Design and Development of Murphy System: Generating Meaningful Negative Samples for KGEs

Master Thesis

Reviewer 1: Prof. Jens Lehmann
Reviewer 2: Prof. Andreas Behrend

Semab Ali

July 2022
Declaration of Authorship

I hereby declare that all the work described within this Master thesis is the original work of the author. Any published (or unpublished) ideas or techniques from the work of others are fully acknowledged in accordance with the standard referencing practices.

Semab Ali

Signed: ________________________________

Date: 25-04-2022
This thesis is dedicated to my late mother. She was the best mother a kid could have. I wish you could see me now. You will always live in my heart.
Acknowledgements

First of all, I am really thankful to my mentors Dr. Sahar Vahdati and M.Sc. Mirza Mohtashim Alam for their support and guidance throughout this thesis. Without their technical support and guidance, this work would have been difficult to achieve. I would like to express my sincere gratitude to Dr. Sahar Vahdati for her continuous moral and technical support throughout this thesis. I am particularly thankful to M.Sc. Mirza Mohtashim Alam for the encouragement and moral support besides being always ready to assist me in all aspects whether it’s in form of technical knowledge or in form of general support and guidance.

I am also grateful to M.Sc. Mojtaba Nayyeri for involvement in early discussions. I convey my gratitude also toward M.Sc. Md Rashad Al Hasan Rony for helping me in this topic and providing me with the valuable feedback and support in technical details.

I would like to extend my sincere thanks and convey my gratitude towards Prof. Dr. Jens Lehmann and Prof. Dr. Andreas Behrend for becoming my examiner for this thesis.

Last, but not least, I am grateful to my friends and family, who supported and encouraged me during this work.
Technical Acknowledgement

The idea of using language models for negative sampling (theoretical formulation and technical descriptions highlighted in the online version of results presented as scientific article[1] for Knowledge graph Embedding models and the base implementation of the corresponding framework belongs to Mirza Mohtashim Alam (PhD student at SDA group at the development time of this thesis). The discussions related to language models is related to the research of Md Rashad Al Hasan Rony (PhD student at SDA group at the development time of this thesis).

This thesis has benefited from the available GitHub pages for which we thank for their great job and open access code. All of these pages were online and available at the time writing this thesis in April 2022. The Murphy framework developed in this work is adapted from RotatE[2] and SANS[3] Github repositories. For creating some of the visualizations in this work, the Trained Embedding Visualizer[4] repository is also used in this work.

For the the execution of all experiments that are performed as a part of this work, the HPC System from TU-Dresden is used.

I, Semab Ali, have build up further implementations on top of the base provided ones, and completed it as system that I named it Murphy. I have run the proposed solutions on several datasets and performed an extensive hyperparameter search. I also re-used the developed visualization tools by Mirza Mohtashim Alam and generated what was needed for the research results.

[2]https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding
[4]https://github.com/ColdMist/Trained_Embedding_Visualizer_2.0
Contents

Declaration of Authorship i

Acknowledgements iii

1 Introduction 1
   1.1 Research Questions ........................................ 3
   1.2 Thesis Structure ........................................... 4

2 Conceptual and Technical Background 5
   2.1 Knowledge Graphs ........................................... 5
   2.2 Knowledge Graph Embeddings ............................... 7
   2.3 Score Functions ............................................ 8
   2.4 Loss Functions ............................................. 11
      2.4.1 Pointwise Loss Functions ............................... 11
      2.4.2 Pairwise Loss Functions ............................... 12
      2.4.3 Setwise Loss Functions ............................... 13
   2.5 Negative Sampling ......................................... 13
      2.5.1 Uniform Negative Sampling ............................. 13
      2.5.2 Bernoulli Negative Sampling ............................ 14
      2.5.3 Affinity Dependent Negative Sampling ................. 14
      2.5.4 Structure Aware Negative Sampling .................... 14
      2.5.5 Self-adversarial Negative Sampling .................... 15

3 The Murphy System 16
   3.1 Motivation .................................................. 17
   3.2 Methodology ................................................ 18
      3.2.1 Pretraining Phase ..................................... 19
      3.2.2 Optimization Phase .................................... 24
   3.3 Clustering-based Negative Sampling ....................... 26
      3.3.1 Negative Sampling Using Uniform Trained Embeddings 26
      3.3.2 Negative Sampling Using Language Model Embeddings 28
   3.4 Dynamic Vector-based Negative Sampling .................. 29
      3.4.1 Negative Sampling with Optimized Affinity ............ 30

4 Implementation 32
Contents

4.1 Preparation and Overview ........................................ 33
4.2 Generating the Pretrained Embeddings ......................... 34
4.3 The Overview Of Hyperparameters .............................. 36
4.4 Running the Murphy Framework ................................ 37
4.5 Model Initialization and Forming Clusters ..................... 39
4.6 Generating Negative Triplet ...................................... 43
  4.6.1 Using Language Model Embeddings ......................... 43
  4.6.2 Using Uniform Trained Embeddings ......................... 45
4.7 Negative Sampling with Optimized Affinity .................... 46
4.8 Score and Loss Calculation ..................................... 48

5 Evaluations and Results ........................................... 50
  5.1 Evaluations Matrices ............................................ 51
  5.2 Datasets ....................................................... 51
  5.3 Hyperparameter Settings ...................................... 53
  5.4 Result and Analysis ............................................ 55
    5.4.1 Performance Comparison of Different Methods ............ 55
    5.4.2 Language Model Embeddings Visualization ............... 58
    5.4.3 Trained Embeddings Graphical Comparison ............... 60
    5.4.4 Analysis of Generated Entity Candidates .................. 62

6 Conclusion and Future Work ...................................... 64
  6.1 Contributions ................................................ 65
  6.2 Future Work .................................................. 67

Bibliography .......................................................... 68

List of Figures ....................................................... 74

List of Tables ........................................................ 75

Appendices .......................................................... 76

Appendix Appendix 1 ................................................. 76
Chapter 1

Introduction

With the advent of novel data management technologies, a great improvement in the performance and application of Artificial Intelligence (AI) methods has been realized. This includes industry 4.0, social media, and personalized applications that capture enormous amount of data from our everyday life. Accessibility of such data has also been made easier by initiatives such as 5-Star, and Open Data principles [1]. Nowadays, almost every domain is touched by novel data collection, enrichment, data storages as well as AI approaches for many tasks such as prediction and recommendation tasks. From medical domains, to agriculture, all domains are actively adapting to up to date techniques. Therefore, citizens tend to see the impact all of these created in every sphere of life.

In the last decade, Knowledge Graph (KG) has emerged as the leading technology of knowledge representation and reasoning. Knowledge graphs are a form of semantic networks to express knowledge in form of triples \((\text{head}, \text{relation}, \text{tail})\) shortened as \((h, r, t)\) [2]. The triple in KG looks like this \((Alex, \text{livesIn}, \text{Cologne})\), where \(Alex\) donates the \text{head}, \(\text{livesIn}\) donates the \text{relation} and \(\text{Cologne}\) represents the \text{tail}. The nodes in KG represent entities and the relationship between entities are models in form of edges. Hence, one complete piece of information so called “a fact” can be represented.

Despite all the growing use cases of KGs, they usually have the problem of incompleteness. The missing links in KGs can be predicted with the aid of existing ones, and this method is refereed as knowledge graph completeness (KGC) task. As knowledge graph completeness is vital for solving many problems, this has attracted immense interest in past years. For the KBC there exist handsome amount
of approaches but knowledge graph embedding (KGE) method has been used quite a lot and through which KGC task has been done quite efficiently.

A lot of AI-based tasks of KG exist in a variety of forms that employed the Knowledge Graph Embedding models, some notable mentions are recommendation systems, relation extraction, social network analysis, and link prediction tasks. Knowledge Graph Embedding Models (KGE) are basically low dimensional vector spaces generated from KG i.e by embedding the components (head, relation, tail) into continuous vector space. Embedding models try to learn embeddings based on facts or triples present in KG, one can call them as positive triples or positive facts. Besides the positive triples, For some KG based tasks embedding models also require the negative triplets. Negatives triples are essentially the facts that are not present in KG and they are of paramount importance in embedding’s learning tasks because more robust and generalized embedding’s can be generated by employing them.

Most common negative triple generation methods randomly corrupt the head or tail part of triples \(^3\) and some other uniformly sampled negative triples from a pool of candidate \(^4\). Studies have suggested that if negative examples are generated in a more efficient way results of the embedding models could be improved a lot \(^5\). Mainly the existing approaches follows the philosophy of close world assumption while creating negative facts that is triples which do not exist in KG are considered as false. As many of the existing approaches use only the facts available in KG to generate negative triplets, a lot of other information remain untapped which could be utilized for effective negative triplet generation. One of the important points to consider in negative fact generation is the contextual information of entities encoded in the triples. By leveraging the contextual representation of the knowledge graph entries with the aid of text available as part of entities, more meaningful negative entity candidates could be generated and thus could improve the model performance significantly.

This thesis will shed light on novel approaches that can be used for efficient negative samples generation and also through this work we would investigate and compare the different embedding model performances after employing these approaches. The other important thing that will be under consideration would be the execution time of models since existing models running time on big datasets i.e FB15k-137 and WN18 is quite large therefore keeping in view these facts, work
has been done in this regard to optimize one of the existing methods and therefore this will also be a focus of the work and will be discussed in coming sections.

1.1 Research Questions

In this paper following questions will be analyzed and detailed discussion with proof through experiments will be under consideration.

<table>
<thead>
<tr>
<th>Research Question 1:</th>
<th>Do different negative sampling techniques effect the performance of KGE models?</th>
</tr>
</thead>
</table>

Through this question, we aim to understand the influence of different negative sampling techniques on the performance of the knowledge graph embedding models. To analyze this aspect, we develop three different negative sampling techniques named as Negative Sampling Using Uniform Trained Embeddings, Negative Sampling Using Language Model Embeddings and Negative Sampling with Optimized Affinity in this work. These methods are evaluated and compared with other state of the art approaches such as Structure Aware Negative Sampling [6] and Uniform Negative Sampling [4].

<table>
<thead>
<tr>
<th>Research Question 2:</th>
<th>Can we leverage language models for generating negative samples?</th>
</tr>
</thead>
</table>

To answer this question we utilized embeddings generated directly from the language model. To asses the impact of the embeddings with other randomly obtained embeddings we did clustering and visualize the clusters to see whether the semantically related entities are grouped together or not. To evaluate further we compare the performance of language driven negative sampling with others state of the art sampling methods.

<table>
<thead>
<tr>
<th>Research Question 3:</th>
<th>Does the performance of negative sampling approach could be enhanced by leveraging the optimized method for Computations?</th>
</tr>
</thead>
</table>
The running time of KGE models is very crucial aspect and can be very beneficial if large datasets can be trained in very reasonable amount of time. To address this question we optimized the Affinity Dependent Negative Sampling [7] by redefining the similarity calculation algorithm so that it can be executed on large datasets.

1.2 Thesis Structure

The thesis will follow the following order of chapters:

Chapter 1 gives the introduction to knowledge graphs and describe the relevant detail needed to understand the basics of it. Further in this chapter we defined the research questions in a brief manner and gives the outline of thesis.

Chapter 2 provides the introduction to knowledge graph embedding and to embedding models alongside their various uses in current world scenarios will be discussed also. Moreover, a high-level overview of different scoring methods along with various negative sampling methods will be provided to comprehend technical concepts more easily that are laid down incoming further chapters.

Chapter 3 sheds light on the various novel approaches for triplet generation alongside the implementation details and their strengths and weakness. We also put light on different language models available, and alongside this their pros and cons will also be discussed. Further in his chapter, we will discuss which language models we have opted for this work and why they are chosen for this work.

Chapter 4 describes the concrete implementation details of different negative sampling techniques that in this work we have formulated. It also lay out details on how to run these methods along side with description of hyperparameters that we have used to run the experiments.

Chapter 5 bring forth the evaluations and results of the proposed techniques side by side detail comparisons of results with existing state of the art approaches. In this chapter the detail analysis of the proposed approaches in terms of performance will also be discussed along with facts and figures supporting the analysis.

Chapter 6 concludes the thesis by providing contributions to the research questions. Each question is addressed and answer to each will be lay out. In the end, thesis is concluded by discussing the possible directions for future research.
2.1 Knowledge Graphs

A knowledge graph is a formalism to represent world knowledge in the form of triplets \((h, r, t)\) or \((\text{entity}, \text{relation}, \text{entity})\) \[\text{[8]}\]. The head and tail part corresponds to entities of triple while relation indicates the relationship between head and tail entities as can be seen in figure \[\text{2.1}\]. The nodes of the KG donates the entities while edges represent relations. For instance, the triple \((\text{Mark}, \text{works in}, \text{Berlin})\), \text{Mark} is the entity or head in the triplet and \text{Berlin} is also an entity that donates the tail while \text{works in} indicates the relationship between two entities. The facts or triplets contained in a knowledge graph are primarily considered as true. The one most important fact about KGs is that lot of knowledge that is not contained in KGs can also be inferred by assumptions. Generally, there are two widely known assumptions practiced in the knowledge graph world named as Open World Assumption (OWA) \[\text{[1]}\] and Closed World Assumption (CWA) \[\text{[2]}\].

Open World Assumption basically assumes that triplet or facts do not exist in the current knowledge base can be true or false, like one can’t deny or accept them rather we say we are not assured of them and are considered as unknown. In contrast to OWA, in Closed World Assumption the facts that are outside of the knowledge graph are considered as false. Usually, the OWA is applied to the

\[\text{https://en.wikipedia.org/wiki/Closed-world_assumption}\]
knowledge base which are incomplete and does not have the complete triplets to represent all the knowledge. On the other hand, CWA are applied to the system that is complete and new knowledge that is outside of it perfectly known to be false.

Figure 2.1: Knowledge graph example. Entities are represented as nodes and relations are represented as edges between nodes.

Knowledge graphs consist of a large number of facts but still they possess a challenge that needs to be overcome which is of incompleteness. The missing information poses a great hurdle in reaping the full benefits of the knowledge graph \[9\]. Therefore it is crucial to determine the missing facts to utilize fully the knowledge bases in domain areas. Although most of the knowledge graphs are incomplete we can still exploit the full benefits by inferring the new facts by determining the plausibility of triples.

There exist wide varieties of techniques and methods that can be used for this purpose. The machine learning methods in this regard has shown great success in predicting missing facts which in turn incorporate new knowledge in existing KG. Among the machine learning methods, knowledge graph embedding models have achieved great success in recent years in carrying out various prediction tasks i.e. link prediction, entity resolution, and are also used in recommender systems.


2.2 Knowledge Graph Embeddings

Knowledge graph embedding aims to embed the entities and relationships of a knowledge graph in low-dimensional vector spaces, which can be widely applied to many tasks [10] [11]. These low dimensional representation targets at exploiting the ‘semantic-closeness’ of words based on their context of occurrence in a particular knowledge world to generate a meaningful relationship.

The low-level latent representations are learned using different KGE models. The goal behind generating the embeddings is to create the vector representation by encoding the structural representation of facts present in knowledge graphs and later employing them in real-world applications. As by the fact most of the knowledge graphs are not complete, these KGE are then used for knowledge completeness through a task of link prediction [12].

Knowledge Graph Embedding Models (KGEM) are employed to learn the embeddings by mapping the entities and relations present inside the knowledge graph into low dimensional vector representation such that information about the structure of the KG is preserved. KGEMs are characterized based on their score function, loss function, and negative triplets generation method. Since the goal of KGEM is to rank the triplets of a knowledge graph based on relevance this is accomplished by defining the appropriate scoring function. The score function has the objective of calculating the similarity between entities. The final aim of the KGEM is to generate the embeddings by keeping in view the information that entities that exist in a specific relationship are in a larger sense related to each other. Therefore, it tries to model them in such a way that they should stay closer in latent representation so the prediction tasks could be carried out efficiently.

KGEM can be used in a variety of tasks and has lot of application areas in the real world where they had proven its success in real world applications [13]. Some of them are highlighted below to understand their importance and usage of them in real world scenarios.

**Link Prediction** Link prediction is the task of finding the missing entity in triple given the incomplete fact in hand. Lets assume a given triple \((entity, relation, ?)\) or in the form \((?, relation, entity)\) the task is to predict the missing entity denoted by \(?\). With the help of link prediction, one can infer the missing knowledge
in the knowledge graphs. Given the fact that mostly the knowledge graphs are incomplete, link prediction plays a vital role in knowledge graph completeness [14].

**Recommender Systems** Another important application task of KGEM is in the recommender system [15]. The recommender system primarily recommends user multiple choices based on user preference or user interests by filtering and then inferring the similarities between user and items present inside the knowledge graphs. By doing so it fulfills the requirement of both user and provider at the same time.

**Entity Resolution** Entity resolution also sometimes called as instance matching, is primarily a task to find similarities between entities given the data. KGEM tries to predict related entities based on linkage patterns they co-share with each other. With the help of entity resolution, one could able to identify the underlying base object that multiple entities are referring to by matching the entities through entity resolution [16].

Now after discussing the different uses of KGE models, the light will be shed on different types of scoring functions that are used commonly.

### 2.3 Score Functions

As the main goal of KGE models is to learn the embeddings of entities and relations for accomplishing various prediction tasks. For learning these embeddings the KGEMs utilizes different scoring methods to determine the plausibility of triples. The scoring function takes the vector representation of \((h, r, t)\) and calculates the vector’s similarity, and finally returns at the end plausibility of triples. The plausibility of triples is determined by assigning the higher scores to true facts in comparison to false or invalid facts i.e. higher plausibility results in a higher score. Scoring functions are really important and have a huge impact on embeddings generation and thus on overall model performance.

Nowadays a wide variety of scoring functions have been used but they can be broadly classified into two categories translational distance models (TDMs) and semantic matching models (SMMs). TDMs used distance based functions to calculate the plausibility of triple. The prominent models that exist in this category are TransE [3], RotatE [4]. While the SMMs use the inner product to calculate
the similarity of a triple in determining the plausibility. The models included in this category are DistMult [17], ComplEx [18], QuatE [19] and SimplE [20]. In the section ahead we will discuss these models in detail along with their working methodology.

**TransE:** TransE stands for translating embeddings for modeling multi-relational data [3] is one of the standard models that uses a translation distance based scoring function. Let’s suppose \((h, r, t)\) donates the embedding vectors of knowledge triplet \((h, r, t)\). TransE transcribes the relations as translation and makes sure that the sum of head embedding vector plus relation embedding vector lies adjacent to tail embedding vector \(h + r \approx t\). Both L1 norm and L2 norm can be utilized to calculate the distance in vector space. Let’s scoring function can be donated as \(f_r\) \((h, t)\), the model scoring function can be defines as:

\[
 f_r(h, t) = -||h + r - t||_{l1/l2} \tag{2.1}
\]

**RotatE:** RotatE [4] in comparison to TransE represents the relation as angle of rotation between \(h\) and \(t\), where \(h\) and \(t\) are k-dimensional embeddings vectors. The model is inspired from Euler’s equation \(e^{i\theta} = \cos \theta + i \sin \theta\). Given the triplet \((h, r, t)\) the relationship between them can be represented using the equation \(t = h \circ r\). where \(\circ\) is an element-wise product. The model scoring function defines as:

\[
 f_r(h, t) = -||h \circ r - t||^2 \tag{2.2}
\]

**DistMult:** DistMult [17] is a bilinear model that models entities as vectors and relations as matrices i.e entity vectors are mapped in \(\mathbb{R}^d\), whereas relations matrices are mapped to \(\mathbb{W}^{d \times d}\). DistMult is different to other bilinear models in the sense is the relation matrix is not an ordinary matrix but a diagonal matrix \(\text{diag}(W_r)\). The scoring function of the model can be described by the following equation:

\[
 f_r(h, t) = h^t W_r t \tag{2.3}
\]

**ComplEx:** ComplEx [18] is primarily an extension of DistMult but different in the sense that it uses complex representation i.e entities and relations are mapped to k-dimensional embedding vectors which constitute of real and imaginary parts \((\text{Re}(z), \text{Im}(z))\). This makes the ComplEx model more robust and powerful compared to DistMult in modeling the different types of relationships [3].
Conceptual and Technical Background

10

\[ f_r(h, t) = \text{Re}(h \odot r \odot t) \]  

(2.4)

**QuatE:** QuatE [19] is a generalization of ComplEx model, it models the entries and relation into hypercomplex valued (quaternion) representation. QuatE also use the Hermitian product instead of Hamilton product while calculating the distance. Quaternion matrices are used to embed entities and relations. Entities embeddings are represented in the matrix of the form \( E \in \mathbb{H}^N \times K \) while relation embeddings have the the form of \( R \in \mathbb{H}^M \times K \). QuatE models the head and tail entity by vector i.e. \( E = a_h + b_h i + c_h j + d_h k \) where \( a_h, b_h, c_h, d_h \in \mathbb{R}^d \) represent real numbers while and \( i, j, k \) are imaginary numbers and donate three different square roots of -1. The score function of QuatE can be defined as:

\[ f_r(h, t) = E_h^t \cdot E_t = \langle a'_h, a_t \rangle \langle b'_h, b_t \rangle \langle c'_h, c_t \rangle \langle d'_h, d_t \rangle \]  

(2.5)

While \( E_h^t = E_h \otimes R_t \) and \( \otimes \) donates the Hamilton product. While it can be further expended to given below form.

\[
E_h \otimes R_t = (a_h \circ p - b_h \circ q - c_h \circ u - d_h \circ v)
+ (a_h \circ q - b_h \circ p - c_h \circ v - d_h \circ u)i
+ (a_h \circ u - b_h \circ v - c_h \circ p - d_h \circ q)j
+ (a_h \circ v - b_h \circ u - c_h \circ q - d_h \circ p)k
\]  

(2.6)

**SimplE:** SimplE [20] is a tensor factorization method and works on the principle of parameter sharing. let \( (h, r, t) \) donates the triple, SimplE defines the two embedding vectors for entities \( h_e, t_e \in \mathbb{R}^d \) while \( e \in E \) and one embedding vector for relation \( r \in \mathbb{R}^d \) i.e. \( r \in R \). Both the embedding vector are learned independently by the model. The score function of the SimplE model is given below.

\[ f_r(h, t) = \frac{1}{2} \left( \langle h'_e, r, t_e \rangle + \langle h_e, r', t_e \rangle \right) \]  

(2.7)

The embeddings learned through SimplE are interpretable, and certain types of background knowledge can also be incorporated into the embeddings through weight tying method [20].
2.4 Loss Functions

KGEM loss function plays a crucial role in model performance but often it is neglected while carry out the training process. Two models having same hyper-parameters but are using different loss function can have substantial difference in their performance which is highlighted by Katsuhiro et al [21]. In KGE models various loss function are used in the optimization process depending on the task in hand. Loss functions can be broadly classified into Pointwise, Pairwise and Setwise loss functions [22] [23]. In the coming section below light will be shed on the most prominent loss functions present in each category.

2.4.1 Pointwise Loss Functions

Pointwise loss functions works on single triple. lets assume set of entities $E$, set of relations $R$ and $l$ donates the label $l \in \{0,1\}$ where value 1 indicates the positive triple and 0 corresponds to negative triple. Let $x \in X$ donates the KG triple, the pointwise loss can be described as by the following equation.

$$L = L(f(x), l(x))$$  \hspace{1cm} (2.8)

where $f(x)$ donates the scoring function and $l(x)$ represents the labeling function. The square error loss, binary cross entropy loss and logistic loss are included in this category and will be discussed briefly.

**Square Error Loss:** Square error loss tries aims to minimize the square difference between the predicted scores and the expected output.

$$L = \frac{1}{2} \sum_{x \in X} (f(x) - l(x))^2$$  \hspace{1cm} (2.9)

Due to square part in loss function it ensure that outliers should be penalized heavily therefore every single bad prediction i.e that deviates considerably from the labels, would have large impact on optimization.

**Binary Cross Entropy Loss:** Binary cross entropy loss is a quite novel approach as it utilizes the whole entities in knowledge set $e \in E$ to calculate the loss for single triple under consideration. For a given triplet $(h, r, t)$ all triplets that satisfy the condition $(h, r, t')$ where $t' \in E$ but $t' \neq t$ are treated as negative
triplets. This makes the binary cross entropy loss computationally very expensive. It can be defined with following equation.

\[ \mathcal{L} = -(l \cdot \log(\sigma(f(x))) + (1 - l) \cdot \log(1 - \sigma(f(x)))) \] (2.10)

where \( l \) indicates the true labels \( l \in \{0,1\} \) and \( \sigma \) donates the logistic sigmoid function.

### 2.4.2 Pairwise Loss Functions

The pairwise loss functions as name indicates applied to pair of triples i.e. positive triple \( x^+ \) and negative triple \( x^- \). Positive triples are true facts that are present in knowledge graph while negative triples are generated by corrupting the positive triple \( (x^+) \) head or tail part. The loss can be defined as below.

\[ \mathcal{L} = L(f(x^+) - l(x^-)) \] (2.11)

For this loss to be used in the optimization of KGE we don’t require labels for triples rather just negative triples are necessary which can be generated at run time. *pairwise hinge loss* and *pairwise logistic loss* are constituents of this category.

**Pairwise Hinge Loss** Pairwise hinge objective is to to maximize the difference between positive score (calculated from positive triplet) and negative score (calculated from negative triplet) by a maximize margin. For this reason it is also known as margin ranking loss.

\[ \mathcal{L} = [\lambda + \exp(f(x) - f(x'))]_+ \] (2.12)

\( f(x) \) donates the score from positive facts while \( f(x') \) donates the score form negative facts and \( \lambda \) is a hyperparameter.

**Pairwise Logistic Loss** Pairwise logistic loss is very similar to pairwise hinge loss, can be defined as following [24].

\[ \mathcal{L} = \log(1 + \exp(f(x) - f(x'))) \] (2.13)
2.4.3 Setwise Loss Functions

Setwise Loss function are distinct to other loss functions in way that they can be applied to set of triple scores and these triples scores can be composed of of positive triplets or negative ones.

**Self-adversarial negative sampling loss**  Self-adversarial negative sampling loss is used in RotatE [4] to sample the negative triplets effectively and to achieve better optimization results. The given below is the probability used as triple weight while selecting the negative triplets.

\[
P((h'_j, r, t'_j)|(h'_i, r, t'_i)) = \frac{\exp(a_f(h'_j, r, t'_j))}{\sum_{i=1}^{n} \exp(a_f(h'_i, r, t'_i))} \quad (2.14)
\]

where \((h_i, r, t_i)\) donates the true facts while \((h'_i, r, t'_i)\) represent the negative facts and \(a\) is the temperature of sampling. The model takes the probabilities generated in equation [2.16] into the loss function which takes the final form as mentioned below:

\[
\mathcal{L} = -\log(\sigma(\gamma + f(h, r, t))) - \sum_{i=1}^{N} p((h', r, t')) \cdot \log(\sigma(-\gamma + f(h'_i, r, t'_i))) \quad (2.15)
\]

2.5 Negative Sampling

Training the popular self-supervised KGE models requires positive facts as well as negative facts for optimization. For these type of learning task the selection of negative sampling distribution plays a vital role in overall model performance. For training graph representation learning models, negative sampling is an efficient and adaptive technique since it does not treat all unobserved data as negative.

2.5.1 Uniform Negative Sampling

The initial negative sampling approaches used the idea of generating negative triplets that are based on random corruption of positive triplet [4]. One of the prominent approach developed that relies on randomly replacing the entity with head or tail of positive triple are known as Uniform Negative Sampling (UNS) was
Conceptual and Technical Background

used in TransE [3]. The choice whether head or tail is to be corrupted is decided uniformly in UNS. To understand this method, let’s assume the given positive triple \((h, r, t)\) present in the batch, the methods tries to generate the negative samples i.e. \(\{(h', r, t) | h' \in \mathcal{E}, h' \neq h\}\) and \(\{(h, r, t') | t' \in \mathcal{E}, t' \neq t\}\) while \(h'\) and \(t'\) are picked uniformly with the condition that generated negative triplets should not exist in knowledge graph \((h', r, t)(h, r, t') \notin \mathcal{G}\).

2.5.2 Bernoulli Negative Sampling

Other than the Uniform Negative Sampling, several different approaches for negative triplet generation exist and have performed really good on benchmark datasets one out of them is Bernoulli Negative Sampling (BNS) [25]. BNS approach focused on utilizing Bernoulli distribution instead of choosing the candidates randomly. The sampling works by assigning corruption probabilities to head and tail based on relationship mapping characteristics and candidates are chosen which have obtained high probabilities. By corrupting head and tail this way it aims for the minimization of false negative labels.

2.5.3 Affinity Dependent Negative Sampling

Another negative sampling method known by the name of Affinity Dependent Negative Sampling shortly as ADNS [7] leverages the closeness of entities with the help of cosine similarity in selecting the negative candidates for the optimization process. The entities that achieves a high cosine similarity scores are perceived as semantically similar and are the best candidates for head or tail corruption. The salient feature of this technique is as the learning process goes further the selection of negative triples also tends to go better as the model takes the recently updated embeddings for similarity score calculation.

2.5.4 Structure Aware Negative Sampling

Recently very efficient negative sampling approach known as Structure Aware Negative Sampling [6] (SANS) technique is proposed. For the negative sampling generation, SANS builds the graph structure based on neighborhood information and
then $K$ number of negative samples are fetched based on neighborhood radius k-hop defined as hyperparameter in the model. The method aims to utilize the graph structural information to access local negative candidates that in turn helps in the creation of hard negative samples. To exploit the structural information of KG, SANS requires preprocessing step that can be computationally extensive based on the size of the dataset and choice of hyperparameters.

2.5.5 Self-adversarial Negative Sampling

Self-adversarial Negative Sampling is used in RotatE \cite{4} to sample the negative triplets and to achieve better optimization results. The method intuitively customized the negative sampling loss and employed probability distribution to sample the negative triplets based on this distribution and this approach is referred to as \underline{Self-adversarial Negative Sampling}.

\[
P((h'_j, r, t'_j)|(h'_i, r, t'_i)) = \frac{\exp(a f(h'_j, r, t'_j))}{\sum_{i=1}^{n} \exp(a f(h'_i, r, t'_i))} \tag{2.16}
\]

where $(h_i, r, t_i)$ donates the true facts while $(h'_i, r, t'_i)$ represent the negative facts and $a$ is the temperature of sampling.
Chapter 3

The Murphy System

This chapter gives the broader level overview of the Murphy System and its underlying components. The objectives behind the creation of this system and the main modules this system comprises of, will also be under the discussion. The section 3.2 describes the methodology of our approaches alongside their architecture diagrams and they broadly classified into two groups: 1) Dynamic Vector based Negative Sampling 2) Clustering-based Negative Sampling. Further in this section, we have the subsections 3.2.1 3.2.2 which describes the individual phases of the Clustering-based approaches. These subsections laid out the details of comprehensive steps involved in each phase with the support of relevant figures and examples.

Subsequently, section 3.3 describes the in depth overview of Clustering-based novel negative sampling approaches conceived as a part of this work. Further in the section each approach under the Clustering-based group is discussed separately in separate subsections 3.3.1 3.3.2 with the support of algorithms. This is done so that we can provide the comprehensive understanding of related concepts that are important to understand our work. The section 3.4 lay out the details of our final negative sampling approach that comes under the umbrella of dynamic vector based approaches, developed under the hood of Murphy system. The part of this work is electronically submitted and are available as arXiv version [26].

\footnote{The author of this thesis is the co-author of this paper. The paper is currently in process of submission in the conference at the time of submitting this thesis.}
3.1 Motivation

In recent years, graph representation learning has gained significant research attention due to the growing number of knowledge graph based applications such as question answering [27], link prediction [13], and recommender system [28]. A huge number of these applications are based on a contrastive learning paradigm. This learning paradigm requires both positive triplet and negative triplet with the goal to rank positive triplet higher than negative triplet that’s why these graph learning methods are also referred to as learning to rank problems.

The negative samples play a crucial role in the optimization of contrastive approaches methods. In recent years various approaches have been developed to effectively create negative triplets and had also been discussed in previous section 2.5 in detail. The foremost problem with these approaches is they tend to have some drawbacks in generating efficient negative samples and therefore this restricts the overall model performance. Like many of others approaches, Uniform Negative Sampling tends to suffer from the problem of nonsensical negative samples generation and hence generated negative triplets generally do not make sense. Let’s take the example from figure 2.1 of the triple \((\text{Alex, livesIn, Cologne})\). Performing tail corruption on this with UNS, the negative triplet generated afterwards could look like as \((\text{Alex, livesIn, bread})\) which does not make any relevance in a real-world scenario and hence will not be useful for model optimization.

Likewise, Bernoulli negative sampling as discussed in section 2.5.2 utilizes the relation mapping properties to generate the negative facts, it still tends to suffer from the problem of generating false negative labels that pose a hindrance in achieving good accuracy. The ADNS [7] although mitigated the issue of generating false negative labels but the approach is not quite scalable to the bigger datasets and faces the runtime issue. While on the other hand SANS [6] which is a quite novel approach and produces quite good results on standard datasets, requires the preprocessing step which is quite time-consuming process.

Therefore keeping in view these points, several negative sampling techniques have been proposed in this work which have achieved good results besides being novel and runtime efficient. Primarily in this work, pretrained models employment with clustering is examined and also the utilization of context information through language models while generating the negative facts is also inspected.
3.2 Methodology

In this section, the complete methodology of the negative samples generation methods will be laid down along with the support of architecture diagrams of the approaches. In the last section, some of the renowned negative sampling methods are highlighted and their shortcomings also discussed briefly. From the analysis, it comes out as from all of the discussed approaches SANS (described in section 2.5.4) and ADNS (described in section 2.5.3) techniques are quite novel and had achieved good performances on various benchmarking datasets. Therefore taking the inspiration from these approaches, multiple approaches have been developed in this work. The approaches that are developed are not only runtime efficient but also achieve state-of-the-art performances on the standard datasets, details of which will be described in upcoming sections.

As stated before, knowledge graphs represent information in form of triples which donates the truth knowledge so we refer to them as positive facts. These positive facts consist of \((\text{head}, \text{relation}, \text{tail})\) shortened as \((h, r, t)\). KGE models discussed in section 2.2 learn the embeddings from knowledge graphs by optimizing a scoring function that assigns higher scores to true facts than invalid facts. So for the process of embedding learning, the model besides the positive facts also requires negative triples which have to be generated from the corruption of positive facts. To generate these negative facts we purpose several techniques for negative sampling generation that can be broadly classified into two categories: 1) Dynamic Vector-based Negative Sampling 2) Clustering-based Negative Sampling.

The architecture diagram of dynamic Vector based approach is shown in figure 3.1. Each step from start to end is highlighted in the figure for better understanding. The method begins by taking KG dataset as input in form of triples as can be seen in the figure 4.3 and employs the BNS to calculate head and tail replacement probabilities. After deciding the position in a triple, the entity embeddings of selected entity and all remaining entities are fetched using the associated entities ids and stored separately. Cosine similarity score is then obtained with the help of using \(\@\) operator by utilizing these embeddings. This process is done on complete batch of triples at once in parallel.
Now moving to second category of developed approaches in this framework, which is the Clustering-based Negative Sampling approaches. The Clustering-based approaches uses a pretrained language model and UNS embeddings and subsequently employs clustering on them for meaningful generation of negative samples. The detailed description of our process is depicted in figure 3.2.

Clustering-based Negative Sampling approaches consist of several steps and therefore are broadly grouped into two phases 1) **Pretraining Phase** 2) **Optimization Phase**. Now each phase will be discussed in detail before explaining the actual negative sampling methods individually.

### 3.2.1 Pretraining Phase

Knowing the fact that negative triplets created by context capturing are more salient negative samples we therefore sought to employ a language model. Language models have been proven useful in various applications and achieve the state
of the art results in various NLP related tasks i.e speech recognition, sentence classification and chatbots [29]. Though they have many benefits but their usability have not been tested in KGs. So keeping this in mind that the language model can efficiently generate the low level vector representation of words based on capturing the context in which word has been employed, the two most prominent models are selected for this work that have already shown great results in the mentioned applications. The models are known as FastText [30] and transformer-based model called as Sentence-BERT [31].

FastText is a language model developed by Facebook and since its availability as an open-source project, it has captured the attention of the research community. FastText differs from other language models in the sense that it does not
treat the whole word as a unit while generating vector representation. FastText strength lies in the idea that words consist of characters so treating the word as n-gram characters it can create meaningful embedding of the words. This strategy of n-grams has a big advantage over others models as it can generate vector representation for words even if they do not exist in the vocabulary. Let’s take an example that two words by mistake are combined together e.g. sunnyweather, if any other method would be used it might throw an error or zero vector since this word do not exist in vocabulary but FastText disembark the word into chunks i.e. sunny and weather, and thus would able to generate meaningful embeddings for this word by combining the individual vectors.

The transformer-based language models have also shown great success in recent years. The models such as BERT [32], GPT [33] and ELECTRA [34] have achieved the state of the art results in language driven tasks. Since language is itself a complex thing i.e. small word can completely change the context of the sentence, therefore earlier language models are designed for the only targeted task. But now with the availability of large corpus publicly available and thanks to transformer-based models this restriction can be easily overcome. There exist a wide variety of transformer-based models that have shown impressive results but in this work we opted for the Sentence-BERT (SBERT) [31] model. SBERT is essentially an optimized version of BERT model and could generate semantically more meaningful embeddings than BERT model. SBERT model is pretrained on the SNLI dataset and is capable to generate rich embeddings with which one can later also compute cosine similarity using them. The SBERT model is more powerful than earlier models due to many reasons but one of them is that it is trained on the large SNLI Stanford dataset which is a collection of nearly 570k human-written English sentences that are labeled manually to limit the errors in data creation.

Using these language models, rich embeddings are generated for entities (subject and predicate) present in the triples of the KG dataset and the approach is named as Negative Sampling Using Language Model Embeddings discussed in section 3.3.2. Let $\mathcal{E}$ donate the set of entities and $\mathcal{E}_{text}$ donate the the set of corresponding textual representation of the entity set $\mathcal{E}$. If we consider the sentence transformer (SBERT) as a function $\Omega_t$ and for FastText $\Omega_f$, the output of $\Omega_t$ is $|\mathcal{E}| \times 786$ and the output of $\Omega_f$ is $|\mathcal{E}| \times 300$ dimensional vector representation.

Now moving to the second variant of Clustering-based negative triplet generation method that is named as Negative Sampling Using Uniform Trained Embeddings
briefly described in section 3.3.1. This method instead of using language model embeddings, utilized the pretrained Uniform embeddings generated using the RotatE framework. These embeddings are obtained by employing Uniform Negative Sampling but are generated without using the adversarial sampling. Although the UNS itself is not quite an elegant method but employing those embeddings as the basis for clustering is rather a useful idea as rich semantics of entities could be gathered after employing clustering over them. Typed Sampling and Nearest Neighbor Sampling are two methods that used the pretrained embeddings and have achieved good performance so taking motivation from these approaches we tend to use pretrained embeddings from Uniform Negative Sampling. If we consider the generated embeddings as a function $\Omega_\text{uns}$, then the output of $\Omega_\text{uns}$ is $|E| \times x$ where $x$ donates the embedding dimension of pretrained embedding model. The generalized equation for this is step is given in the equation 3.1 Where $n_e$ is the number of entities in $E$ and $x$ donates the embedding dimension of pretrained language or UNS model.

$$\Omega(E_{text}) : E_{ST} \in \mathbb{R}^{|E| \times x} \quad (3.1)$$

The output dimension from the generated embeddings are quite large and on large dimension the clustering algorithm suffer the the problem which is known as curse of dimensionality. The curse of dimensionality prohibits the proper use of clustering algorithms in the high-dimensional space. So for this reason, the dimensionality reduction techniques are applied on top of generated embeddings in order to reduce the dimension to size $Z$ which is less than those generated by respective language or UNS methods.

Dimensional reduction is primarily a technique to reduce the number of input features in the input data and only those features are retained that capture the essence of data and others are left out which are not true representatives of data. It not only reduces the number of features but additionally a lot of computation cost could also be saved by lowering the dimensions of data. In this work we utilized three different methods for dimensionality reduction on the embedding generated from the language model these are known as: 1) Principal Component Analysis (PCA) 2) Spectral Embedding 3) T-SNE.

---

2[https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding](https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding)
Principal Component Analysis tries to map the data from high dimensional space to lower with the objective to maximize the variance in low dimensional space. PCA works by finding the linear correlation and as sometimes it is difficult to find one so in some cases it also results in loss of information in the mapping process from high to low dimension depending upon the data in hand. Keeping this in mind, the other methods of dimensionality reduction are also explored. Spectral Embedding is another dimensional reduction method that is primarily a non-linear method that works based on eigen-decomposition. It works by building the k-nearest neighbors for each point in high dimensional space and then construct the undirected graph from these points. As a final step it assign weights to the edges of the graph so that input patterns can be mapped to nearby outputs in low dimensional space. T-SNE is another useful non linear dimensionality reduction algorithm. It operates with the goal to minimize the Kullback-Leibler divergence between low-dimensional embedding and the high-dimensional data so as to preserve the local variance.

Let’s consider the dimensionality reduction function as \( \phi \). The input of this function is the output of \( \Omega \). Additionally \( \Omega \) also requires the desired number of reduced dimension \( \mathcal{Z} \) as a argument. The output is a \(|\mathcal{E}| \times \mathcal{Z}\) dimensional vector namely \( \mathcal{E}_{ST}^Z \in \mathcal{R}^{(|\mathcal{E}| \times \mathcal{Z})} \) which is reflected in the equation 3.2.

\[
\phi(\mathcal{E}_{ST}) : \mathcal{E}_{ST}^Z \in \mathcal{R}^{(|\mathcal{E}| \times \mathcal{Z})} \quad (3.2)
\]

Last but not least, **Elbow method** is employed finally before moving towards the next phase in order to calculate efficiently the optimal number of clusters for the generated embeddings obtained after applying the dimensionality reduction technique. The Elbow method is one of the popular algorithm when it’s about finding the optimal number of K (no of clusters) for clustering. It works by iterating over a range of clusters values provided as argument and calculate the distortion score and inertia at each given value of k after building the clusters. The ideal value of K would be the point where the distortion/inertia started to decrease in a linear fashion as seen in figure 3.3. If the output of the elbow closely looks like an arm then it’s a good indication that the model fits perfectly on the data and therefore the predictions are quite accurate. The dotted line in the output shown in figure 3.3 indicates that setting \( k = 10 \) is a good number to try while doing clustering.
3.2.2 Optimization Phase

The optimization phase comprises of building the neighborhood clusters, optimizing the score function, and performing evaluation on the test set. The clustering is performed by using K-means++ [36] algorithm provided by Scikit-Learn [37] library. K-means++ is applied on the output of dimensionality reduction algorithm $E_{ST}$ to obtain $K$ number of clusters where this number $K$ is estimated using elbow method. The clusters are comprised of entities $\{e\} \subseteq E$ and these entities are linked to respective clusters based on the squared euclidean distance calculated from each entity to centroid $\{c\}$ of each individual cluster. Then each entity takes the membership of that cluster of whom it lies nearest to. This is reflected in the equation 3.3.

$$\text{Argmin}_k \sum_{i=1}^{K} \sum_{e \in E \cap C} \| E_{ST} - \mu_i \|^2$$ (3.3)
The overall cluster forming process can be found in algorithm 1. Finally, after clusters are formed each entity that is mapped to their belonging cluster centroid acts as the representative of the whole cluster. After assigning each entity $e$ to its respective clusters $c_i \in C$, we construct the mapping dictionary $dict$, where the cluster assignment of each individual entity is preserved. In $dict$ the keys are the set of all the entity symbols $E$ and the values are associated representative cluster centroids $C$. The distances of the cluster centroids are also preserved.

**Algorithm 1 Building neighborhood using K-means++**

**INPUT:** Entity embedding $E$ from language model or uniformly pretrained model, Desired number of clusters $K$, dimensionality reduction function $\phi$, reduced dimensionality $Z$.

**OUTPUT:** Cluster Dictionary $dict$ where keys: mapped entities $\{e\} \subseteq E \subseteq C$, values: cluster centroids $\{c\}$, $KM_\theta$ as K-means++ Attributes such as distance to centroids, number of centroids etc.

**Function Build entity clusters($E, K, Z$):**

- $E_\mathbb{Z} \leftarrow \phi(E, Z)$
- $KM \leftarrow$ Initialize KMeans++ ($numberofclusters = K$)
- $D_{E_\mathbb{Z},c}, KMeans++\_attributes = KM(E_\mathbb{Z})$ // For each entities in $E_\mathbb{Z}$ obtain assigned cluster number $C$
- $dict \leftarrow \{\}$
  
  for each entity $e \in E$ do
  
  - $dict\{key : e, value : c\} \leftarrow$ map($\{e\} \subseteq E_\mathbb{Z}, c \in C$) // map the matching entities $\{e\} \subseteq E$ belonging to the centroid $c$

return $dict, KM_\theta$

After building the clusters $dict$ one can able to group the similar entities present in each cluster. In the training step, we can get all entities that are identical to entity $e_i$ by just knowing to which cluster this entity belongs to with the help $dict$ already built. Secondly, we can also compute the nearest $K$ clusters (s.t. within d nearest distances) to the entity in hand so the more number of relevant entities even though present in other clusters but are quite similar based on the distance metric could also be fetched to increase the candidate pool. So using this $dict$ one can retrieve all the relevant candidates for negative triplet generation quite efficiently and effectively. Then any scoring function i.e. TransE [3], RotatE [4], QuatE [19], DistMult [17] and ComplEx [18], can be used and training process can be optimized by employing these negative triplets.
3.3 Clustering-based Negative Sampling

As discussed previously, employing a clustering strategy to build entities neighborhood is a very efficient method for finding relevant entities that are semantically similar to each other and that could also be potentially used for negative triplets creation. Based on the principle of utilizing the clustering approach for negative samples generation, two novel strategies are developed and are named as 1) Negative Sampling Using Uniform Trained Embeddings 2) Negative Sampling Using Language Model Embeddings. These strategies will be discussed in detail in the coming subsections.

3.3.1 Negative Sampling Using Uniform Trained Embeddings

KG embeddings that are trained and optimized using some scoring function can be used to produce rich negative samples and boost the performance of the KGE model. Pretrained KGE has been used in the past to reduce training time [38] and handle data related issues i.e sparsity of data [39] but these generated embeddings can be a great source of finding meaningful correlated entities.

In this approach, the pretrained embeddings are utilized that are trained on a similar dataset but used the Uniform Negative Sampling approach (discussed in section 2.5.1) during the training process and this approach is named as Negative Sampling Using Uniform Trained Embeddings shortly as PMC. As UNS method has the drawback that it does not create meaningful negative samples because it randomly perturbs the head or tail entity to form a negative triple but despite this fact it also has the great advantage of being runtime efficient. Keeping in mind that embeddings obtained using Uniform Negative Sampling although does not achieve the desired model performance itself but can be employed to fetch admissible candidate entities for corrupting head or tail for a given positive triplet. It is therefore decided that the pretrained UNS embeddings can be a good source for the negative triple formation process if utilized in a proper way.

In the PMC method, first of all, the clusters are acquired by using the pretrained UNS embeddings, the process of which is highlighted in algorithm 1 and thereafter the respective entities belonging to individual clusters, mapped in form dict is
Algorithm 2 Negative Sampling Using Uniform Trained Embedding

INPUT: Training set $\mathcal{T}_{h,r,t}$, cluster dictionary $\text{dict}$, Entity set $\mathcal{E}$, Relation set $\mathcal{R}$, Batch size $\mathcal{B}$, Negative sample number $\mathcal{N}$, K-means++ attributes $KM_{\theta}$
OUTPUT: For given batch of triples $\mathcal{T}_{\text{batch}}$ generate batch of negatives $\mathcal{T}'_{\text{batch}}$ where $h', r, t' \in \mathcal{T}'_{\text{batch}}$

Function Sample Negative($\mathcal{T}_{h,r,t}, \text{dict}, \mathcal{R}, \mathcal{B}, \mathcal{N}, \mathcal{H}$):

\[
\begin{align*}
\mathcal{T}_{\text{batch}} & \leftarrow [] \\
\text{for} & \text{ triple } t_{h,r,t} \in \mathcal{T}_{\text{batch}} \text{ do} \\
\text{ corrupt} & \_\text{position} \leftarrow \text{ probability}(h, t, 0.5) \\
\text{ entity} & \_\text{to} \_\text{be} \_\text{corrupted} \leftarrow t_{h,r,t}[\text{corrupt} \_\text{position}] \\
\text{ target} & \_\text{cluster} \leftarrow \text{ fetch} \_\text{cluster}(\text{entity} \_\text{to} \_\text{be} \_\text{corrupted}, \text{dict},KM_{\theta}) \\
\text{ corruption} & \_\text{candidates} \leftarrow \text{ dict}[\text{target} \_\text{cluster}] \\
\text{ corrupted} & \_\text{entities} \leftarrow \text{ random} \_\text{choice}(\text{corruption} \_\text{candidates}, \mathcal{N}) \\
\text{ Negative} & \_\text{Triples} \leftarrow \text{ random} \_\text{choice}(\text{corruption} \_\text{candidates}, \mathcal{N}) \\
\mathcal{T}'_{\text{batch}} & \leftarrow \mathcal{T}'_{\text{batch}} \cup t' \\
\text{return} & \mathcal{T}'_{\text{batch}}
\end{align*}
\]

obtained, using the process described in section 3.2.2 and in algorithm 1. After obtaining the $\text{dict}$, The ultimate process of negative sampling begins. For each triple $(h, r, t)$, firstly the decision about the head or tail corruption is decided uniformly similar to one in UNS technique. Subsequently, deciding at run time for each triple $(h, r, t)$ whether to pick $h$ or $t$ the chosen $h$ or $t$ is then searched in each individual cluster by matching with the entities present in each cluster. By using python\footnote{https://www.python.org/} in\footnote{https://docs.python.org/2/library/stdtypes.html} operator the searching operation can be done very elegantly. After getting the cluster to where the entity belongs, all other entities except the one in hand are pulled out from the fetched cluster. When this step is finished, a pool of entities are available that can act as candidates for replacement. The pool of the entities that have been gathered is semantically similar to the entity that need to be corrupted. Subsequently after acquiring all the entities, randomly $\mathcal{N}$ entities are chosen from the pool and either head or tail as whatever decided earlier is corrupted afterwards. Where $\mathcal{N}$ is a hyperparameter that donates the number of negative samples to choose from the available one’s. After corruption is done these negative samples are passed to the scoring function and then the loss is calculated and backpropagation of loss is performed finally at the end of each step. This process continues till the specified number of $\text{epochs}$ or till the desired level of optimization is not achieved.
3.3.2 Negative Sampling Using Language Model Embeddings

As Language based models can efficiently generate the low level vector representation of words based on capturing the text. Therefore in this approach language models are employed for embedding generation by using the the entities text as initial step and subsequently these embeddings are used as input for clustering method to obtain negative triplets. The approach is named as **Negative Sampling Using Language Model Embeddings (LMC)**. With the help of a language models, semantically rich embeddings could be generated based on the provided KG data set. In this work, as already stated two well known language models *FastText* [30] and *Sentence-BERT* [31] are used. After doing the steps mentioned in the pretraining phase in the section 3.2.1, subsequently, the embeddings are acquired and saved. Then any scoring function i.e. TransE [3], RotatE [4], QuatE [19], DistMult [17] and ComplEx [18] can be used and optimized by employing these negative triplets. Then afterwards these obtained embeddings are passed to the K-means++ algorithm and thereafter clusters are acquired. The number of $K$ clusters that is obtained, $K$ is the output predicted by the elbow method. The no of clusters to be computed can also be set without running the elbow method but based on the experiments it is noticed that the elbow method gives the optimal choice and therefore it is recommended to set this value based on the elbow method predicted value.

After setting the desired number of clusters, the mapped *dict* of entities to the respective clusters are obtained based on the steps mentioned in algorithm 1. Lets suppose based on uniform selection, head entity $h \in \mathcal{E}$ is selected for corruption from positive triple $(h, r, t)$. Then the selected head entities embedding vector $h$ is fetched using the associated index and compared with each cluster center that was obtained after executing K-means++ clustering. Following that, the euclidean distances are computed from entity to each cluster centroid. The obtained distances are then afterward sorted in ascending order. Now based on provided argument hop size $\mathcal{H}$ the clusters are selected from the sorted list that are already ranked based on distances from the entity *head* to each cluster centroid. So if hop size $\mathcal{H} = 2$ then the closest two clusters are chosen from the list available that will be the most closest to the entity in consideration. After obtaining the clusters all entities in there are taken and grouped together in one list making the pool
Algorithm 3 Negative Sampling Using Language Model Embeddings

**INPUT:** Training set $T_{h,r,t}$, cluster dictionary $dict$, K-means++ attributes $KM$, Entity set $E$, Relation set $R$, Batch size $B$, Negative sample number $N$, Number of nearest $H$ hops based on $d_{max}$ distance

**OUTPUT:** For given batch of triples $T_{batch}$ generate batch of negatives $T'_{batch}$ where $h', r, t' \in T'_{batch}$

Function `Sample Negative($T_{h,r,t}, d, KM, R, B, N, H$)`:  

```plaintext
$T'_{batch} \leftarrow \emptyset$

for triple $t_{h,r,t} \in T_{batch}$ do

    corrupt_position $\leftarrow$ probability($h, t, 0.5$)

    entity_to_corrupt $\leftarrow t_{h,r,t}[\text{corrupt_position}]$

    distance_to_clusters $\leftarrow$ compute_distance(entity_to_corrupt, $dict, KM$)

    sorted_clusters $\leftarrow$ sort($dict, distance_to_clusters$)

    corruption_candidates $\leftarrow dict[0 : H]$

    corrupted_entities $\leftarrow$ random_choice(corruption_candidates, $N$)

    NegativeTriples$' \leftarrow t_{corrupt\_position} = \text{corrupted\_entities}$

    $T'_{batch} \leftarrow T'_{batch} \cup t'$

return $T'_{batch}$
```

of candidates to choose from. Then $N$ number of negative triples are formed at runtime by randomly picking the entities and replacing the head entity of the positive triple. Where $N$ is the number of negative samples to create for each triple and it is the hyperparameter that one can set before training. Each step of this method is described in detail in algorithm 3. After creating the desired number of negative triplets these are passed into the optimization process and hence relatively scores and loss are calculated and this training process continues till the desired number of epochs.

### 3.4 Dynamic Vector-based Negative Sampling

The embeddings model based on Dynamic Vector-based Negative Sampling (DVNS) has shown great results in the past and has been a center of attention for past years [40] [41]. As similar to most of the other embedding models, these embedding models also requires negative triples for the learning task. The key idea behind these DVNS approaches is that they tend to generate negative candidates that improve as time progresses and with the passage of time more plausible candidates are fetched and hence the model performance tends to get better and better as training goes ahead.
3.4.1 Negative Sampling with Optimized Affinity

Dynamic Vector-based Negative Sampling are uniquely characterized by the fact that it tends to improve with time as the embedding vectors used for negative sample generation got better with time also. By this fact, it is worth to know that although negative triplet generation plays a key role but the optimization time is also a major factor in the success of these approaches as these methods generally require too much time for the processing. The Affinity Dependent Negative Sampling for Knowledge Graph Embeddings (ADNS) \cite{7} is also one of the approach that is based upon DVNS strategy. ADNS intuitively employs the cosine similarity to create more meaningful negative samples compared to other methods but it still possesses the problem of large training time and hence it is not quite scalable to bigger datasets.

**Algorithm 4** Negative Sampling with Optimized Affinity

**INPUT:** Training set $S_{\{h, r, t\}}$, Entity Set $\xi$, Relation set $R$, batch Size $\beta$, Number of Negatives $N$

**OUTPUT:** For Given Batch $\beta_{\{h, r, t\}}$ return negative triples $N_{\{h', r, t'\}}$

**Function Sample Negative**($S_{\{h, r, t\}}$, $\xi$, $R$, $\beta$, $C$):

for triple $t \in \beta_{\{h, r, t\}}$ do

\[ \text{candidate\_positions} = \text{Bern}(t, R) \]

bern negative sampling to decide whether head or tail corruption

\[ \text{entity\_candidates} \xi_t = t_{\text{candidate\_positions}} \]

\[ M_{\text{score}} = \text{Calculate\_Cosine\_Score}(\xi_t, \xi_c) \]

\[ M_{\text{score}} = \max(0, M_{\text{score}}) \]

for $i \in \text{length}(M_{\text{score}})$ do

\[ \text{probability\_fitness}_i = M_{\text{score}}i / \sum_j M_{\text{score}}j \]

generate the fitness vector

\[ \text{selected\_entities} = \text{random\_choice}(\xi_c, C, \text{probability\_fitness}) \]

\[ \text{NegativeTriple}_{\{h', r, t'\}} = \text{FormNegative}(t, \text{selected\_entities}, \text{candidate\_position}) \]

form $N$ negative triples/positive

\[ N_{\{h', r, t'\}} = N_{\{h', r, t'\}} \cup \text{NegativeTriple}_{\{h', r, t'\}} \]

append $N$ negative triples per positive to the total batch negative set

return $N_{\{h', r, t'\}}$
Based on the in-depth analysis of this approach, it comes out as the main step that is taking much of the time in a training process and needs further optimization, is cosine similarity calculation step. In ADNS the similarity score is calculated for each candidate separately by comparing against all other entities in training set and then finally probability fitness vector is generated. Keeping in view this limitation, an enhancement in ADNS method is proposed. We call this method with our added enhancement as **Negative Sampling with Optimized Affinity (OADNS)**.

**Algorithm 5** Calculating the Cosine Similarity

**INPUT:** Entity Set $\xi_c$, Entity Candidates $\xi_t$

**OUTPUT:** Cosine Similarity Score $M_{score}$

Function $\text{Calculate\_Cosine\_Score}(\xi_c, \xi_t)$:


textbf{entities\_norm} = $\|\xi_t\| \cdot \|\xi_c\|$

$M_{score} = \xi_t \odot \xi_c^\top \div \text{entities\_norm}$

return $M_{score}$

The intuition is instead of calculating cosine similarity for each candidate, this step can be done altogether for the whole batch and thus running time could be reduced to a large extent in a very efficient manner. Through the help of this added improvement, this approach becomes scalable to the bigger datasets. For calculating the similarity score for the complete batch, the $\odot$ operator \(^5\) (also referred as \_\_matmul\_\_ operator) is utilized as can be seen in algorithm 5. The operator basically calculates the matrix multiplication by following the principle of parallelization and hence it efficiently used the gpu while doing the calculations and reduce computation time significantly. The returned score is then used to calculate probability fitness vector $F$. Then using the fitness vector $F$, $N$ probable negative candidates are fetched from the set of candidates. These fetch negative samples are the ones that have obtained a higher cosine similarity score during the computation and are therefore most similar to the entity for which perturbation has to be made. Finally, using these candidates the negative samples are generated and passed to score function for plausibility computation of the positive and negative triples and then loss is computed and backpropagated and at the end embeddings are updated respectively.

\(^5\)https://docs.python.org/3/reference/datamodel.html?highlight=matmul\emulating\-numeric-types
Chapter 4

Implementation

This chapter is more of a technical based and will shed light on the low level implementation details of different negative sampling techniques developed as part of this work.

The section 4.1 gives the details about the underlying folder structure alongside it also provides a brief level overview of different libraries and methods used in the implementation of Murphy. Section 4.2 presents the details steps to obtain the language model embeddings i.e. FastText and Sentence-BERT required for executing the LMC approach in the murphy framework. Furthermore, this section also describes the method of acquiring the Uniform embeddings that are used as input in PMC approach.

Section 4.4 provides the overview of how to execute the different negative sampling approaches which are part of Murphy framework. It also lay out the samples commands for the execution of individual approaches. Section 4.3 will list out the details about the hyperparameters used in the approaches alongside the datatye and with the brief explanation of each of the respective hyperparameter.

The details about the negative triplet formation is put forward in the section 4.6. Under this section, subsection 4.6.1 describes the triplet formation using the language models and subsection 4.6.2 describe the negative sampling generation process using the Uniform embeddings. The last section 4.7 of this chapter explains the implementation details of OADNS methods and its working.
4.1 Preparation and Overview

Before explaining the implementation details, the complete information about the libraries used in the project and overall structure of directory will be provided first. Murphy framework is adapted from the implementation of SANS[1] which is implemented using Python language. Apart from this, code from RotatE[2] github repository have also been utilize i.e. some of the embedding models and scoring function that are predefined are directly used while additional scoring methods and embedding models that were required are added beside them. The PyTorch[3], Scikit-Learn[4] and Numpy[5] are the main libraries used to build this project. The PyTorch library which is developed by Facebook is very well accepted for building machine learning and deep learning projects. The Scikit-Learn is a machine learning library and it is also very popular library among researchers community. It contains the huge amount of machine learning algorithms that one can easily use to perform machine related tasks. It also provides tools for data prepossessing, clustering and dimensionality reduction. Numpy is another python library that is widely used for performing efficient numerical computations.

For the preparation of datasets, so they can work with language models, entities text $E_{text}$ have to be fetched or gathered for some datasets that do not contain the corresponding textual representation of entities $E$ e.g. CoDEx (S-M-L), WN18, WN18RR, and FB15k-237. For this purpose the Sparql queries are developed for fetching the textual information in the form of json files. The queries can be seen here on this github repository[6]. These json files are then mapped to corresponding triplets in training, validation and test files. In the case of WN18 and WN18RR entities textual labels are directly obtained and were also mapped already to the corresponding triplets, from this website[7].

Now talking about structure of project there exist a root directory which is named as Murphy which contains all the relevant files needed for the execution of approaches. Under the root folder lies the Codes directory which contains all implementation scripts in form of .py files. The datasets directory is used to contain

---

1https://github.com/kahrabian/SANS
2https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding
3https://pytorch.org/
4https://scikit-learn.org/stable/
5https://numpy.org/
6https://github.com/rashad101/entityid2entitylabel_wikidata
7https://deepai.org/datasets
all datasets and models directory is used to store the models after training. The trained_Embeddings folder is used to store the pretrained embeddings generated from language model and from UNS method which then can be used during training. Where as utils folder contain the python files for generating the embeddings from language model and also executing the elbow method. The structure is visible in figure 4.1

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{folder_structure.png}
\caption{Folder Structure of Murphy System.}
\end{figure}

4.2 Generating the Pretrained Embeddings

In this work, two types of pretrained embeddings are utilized, one that are generated from the language model and the other from using UNS approach. The language model and Uniform trained embeddings has to be generated before training step and must be stored in the trained_embeddings folder before training begins. Since for the experiments two different types of language model embeddings are employed although in final results only transformer-based embeddings are used but the process of both of them will be explained briefly.
For generating the Transformer based embeddings, the sentence-transformers\(^8\) python framework is used. The framework has a rich collection of wide variety of pretrained embeddings for different tasks. From this sentence-transformers framework, Sentence-BERT (SBERT) model is employed for embeddings creation. The relevant code for this part can be seen in figure 4.2. The output of each entity is vector with dimension \((1 \times 768)\).

For FastText language model embeddings, the open-source Gensim library \(^9\) is utilized. It is considered as one of the widely used libraries in natural language processing (NLP) and information retrieval (IR) community. Beside the functionality of training your own models the It also provides the ready to use different pretrained models. From this library, the FastText embedding model is used in order to get the pretrained embeddings vectors for the entities in the dataset. FastText is really powerful model in comparison to other word embeddings models like word2vec as it works on the principal of utilizing the morphological structure of languages. It looks at word not as whole but as the collection of sub words, this makes the model very robust in terms of embedding generation. The model generates the embeddings with dimensions \((1 \times 300)\) for each input entity.

```python
import numpy as np
from sentence_transformers import SentenceTransformer
from gensim.models import fasttext

text = 'This is a sample text'
encoder = SentenceTransformer('distilbert-base-nli-mean-tokens')
vec = fasttext.load_facebook_vectors("data/wiki.simple.bin")

def get_vec(text, type='sentence-transformer'):
    if type=='sentence-transformer'
        return encoder.encode([text])
    else:
        vec.get_vector(text).reshape(1,-1)

if __name__=='__main__':
    embedding_st = get_vec('germany', type='sentence-transformer')
    print("Sentence-Transformers embedding:", embedding_st)

    embedding_ft = get_vec('germany', type='fasttext')
    print("FastText embedding:", embedding_ft)
```

**Figure 4.2:** The methods to generate the FastText and Transformer based embeddings.

---

\(^8\)https://www.sbert.net/

\(^9\)https://radimrehurek.com/gensim/
For conducting the initial experiments, both FastText and Transformer embeddings are employed in the initial experiments and after carefully analyzing the performance comparison of them with with each other, its finally decided to use only the Transformer based embeddings for the final experiments that are presented in chapter 5 mainly based on the following reason. As FastText is based on the idea of modular embeddings s.t. it works by splitting the word into its component, retrieve their corresponding embedding vectors individually and finally generate the final word embedding by putting them together. In our case entities text are mostly composed of lot of noise. The text contains many additional characters beside the nouns such as underscore and dashes. Other than this, most entities consist of multi-word phrases like e.g. neuroreactive Substance or biogenic amine and after carefully examining FastText trained model vocabulary, it comes out that the model does not have these multi-word phrases in them. So keeping this in mind, its finally concluded that Transformer based embeddings (SBERT) are more robust, efficient and more suitable for our task compared to FastText.

And for obtaining the Uniform trained embeddings, the same dataset for which later trained embeddings are required is made to execute with UNS approach utilizing the RotatE framework [4] and then later are stored so that it can be used as input to the framework. The intuition behind using these Uniform trained embeddings is that the execution time required to generate these embeddings are really small and they possess rich semantics which can be exploited using the clustering approach, especially for comparable entities candidates retrieval process. As the candidates retrieval process is a key part of any negative sampling approach thus using these pretrained embedding in this can be proven to be beneficial. Another good thing about them is that these are required to be generated only once for each dataset and hence later can be used recurrently.

4.3 The Overview Of Hyperparameters

The Murphy project is developed in such a way that user can flexibly choose different options easily and can used this module to extend further. Keeping the extensibility in mind, the framework is designed in such a way that the hyperparameters can be set straightforwardly and so the running of different experiments should be quite easy to perform. The run.py is the main script or entry point and
Implementation

it contain all the different arguments that a user can specify while executing. The complete list of whom one can see in table 4.1.

Some of the hyperparameters are mandatory to provide while others are additional and can be left. Some hyperparameters also have default values which are set to optimal values on the basis of conducted experiments if in case user does not provide it.

4.4 Running the Murphy Framework

The Murphy framework can be brought into action by running the run.py file along with arguments provided in table 4.1. One has to provide one out of the three arguments at least for the execution to begin and these are: --do_train --do_test and --do_valid. If no pretrained model directory is specified user has to provide --do_train, the other two flag can only be used with pretrained model or embeddings. For loading the already trained embeddings --init attribute is used to specify embeddings directory from where it can be loaded. The pretrained embeddings loading is vital for performing further training or testing the trained embeddings. An example command to run the clustering based approach is mentioned below:

```bash
python -u codes/run.py --do_train --cuda --do_test --test_batch_size 8 --data_path data/wn18 --model TransE --hidden_dim 500 --negative_sample_size 100 --batch_size 512 --gamma 6 --adversarial_temperature 0.5 -lr 0.00005 --max_steps 80000 --dataset wn18 --reduce_dim_comp 200 --reduce_dim --reduce_dim_type pca --nclusters 10 --khop 3 --save models/TransE
```

While OADNS method could be executed by following command:

```bash
python -u codes/run.py --do_train --cuda --do_test --test_batch_size 8 --data_path data/wn18 --model TransE --hidden_dim 500 --negative_sample_size 100 --batch_size 512 --gamma 6 --adversarial_temperature 0.5 -lr 0.00005 --max_steps 80000 --neg_sampling_method ADNS --save models/TransE
```
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-cuda</code></td>
<td>boolean (true/false)</td>
<td>use gpu or not</td>
</tr>
<tr>
<td><code>-do_train</code></td>
<td>boolean (true/false)</td>
<td>perform training or not</td>
</tr>
<tr>
<td><code>-do_valid</code></td>
<td>boolean (true/false)</td>
<td>do validation or not</td>
</tr>
<tr>
<td><code>-do_test</code></td>
<td>boolean (true/false)</td>
<td>whether to do testing or not</td>
</tr>
<tr>
<td><code>-data_path</code></td>
<td>dataset path (string)</td>
<td>dataset path</td>
</tr>
<tr>
<td><code>-model</code></td>
<td>model name (string)</td>
<td>model name to select</td>
</tr>
<tr>
<td><code>-dataset</code></td>
<td>dataset name (string)</td>
<td>dataset name</td>
</tr>
<tr>
<td><code>-negative_sample_size(N)</code></td>
<td>float/integer</td>
<td>no of negative samples</td>
</tr>
<tr>
<td><code>-hidden_dim(D)</code></td>
<td>float/integer</td>
<td>hidden dimension of model</td>
</tr>
<tr>
<td><code>-gamma(γ)</code></td>
<td>float/integer</td>
<td>the maximum margin</td>
</tr>
<tr>
<td><code>-adv</code></td>
<td>boolean (true/false)</td>
<td>whether to use adversarial sampling or not</td>
</tr>
<tr>
<td><code>-adversarial_temperature(τ)</code></td>
<td>float/integer</td>
<td>value of adversarial temperature to use</td>
</tr>
<tr>
<td><code>-khop(H)</code></td>
<td>float/integer</td>
<td>no of nearest clusters to consider</td>
</tr>
<tr>
<td><code>-nclusters(K)</code></td>
<td>float/integer</td>
<td>no of cluster to use</td>
</tr>
<tr>
<td><code>-reduce_dim</code></td>
<td>boolean (true/false)</td>
<td>whether to use dimensionality reduction or not</td>
</tr>
<tr>
<td><code>-reduce_dim_comp(Z)</code></td>
<td>float/integer</td>
<td>reduce dimension to this value</td>
</tr>
<tr>
<td><code>-reduce_dim_type(R)</code></td>
<td>name of algorithm (string)</td>
<td>name of dimensional reduction algorithm</td>
</tr>
<tr>
<td><code>-batch_size(B)</code></td>
<td>float/integer</td>
<td>batch size to use</td>
</tr>
<tr>
<td><code>-lr(α)</code></td>
<td>float/integer</td>
<td>learning rate</td>
</tr>
<tr>
<td><code>-max_steps</code></td>
<td>float/integer</td>
<td>run training to to these steps</td>
</tr>
<tr>
<td><code>-save</code></td>
<td>director path (string)</td>
<td>save model to this directory</td>
</tr>
</tbody>
</table>

**Table 4.1:** List of the attributes and their description that can be used as arguments for initiating the training process.
4.5 Model Initialization and Forming Clusters

The Murphy framework can be executed by running the run.py script present inside the Codes directory. The user has the ability to choose from set of hyper-parameters listed in table 4.1. When the user executes the run.py file there is a method inside it named as parse_args() that is invoked, which checks whether all essential arguments are provided with the value or not. Besides this it also validates their data types and throw error if data is not provided in desired format.

Subsequently, after parsing the arguments control is transferred to the main() method where entities.dict and relations.dict files are loaded from the data path provided by the user. The code section relevant to this can be seen in the figure 4.4. After reading the entities and relation from the files, the triples for training, validation and testing are subsequently loaded afterwards. It is worth to mention here that the data in form of i.e. train, valid and test files should be stored in .txt files while entities and relations should be mapped to integer keys in form a python dictionary with .dict extension. The glimpse of how training data triples looks like is demonstrated in the figure 4.3.

```
trade_name.n.01  _member_of_domain_usage  metharribal.n.01
call.v.03  _verb_group  call.v.09
united_kingdom.n.01  _member_of_domain_region  facer.n.01
blatodore.n.01  _member_meronym  cockroach.n.01
rickettialcasee.n.01  _hypernym  bacteria_family.n.01
antoprocta.n.01  _hypernym  phyllum.n.02
position.n.01  _hypernym  point.n.02
drill.n.01  _has_part  chuck.n.03
breadfruit.n.01  _hypernym  fruit_tree.n.01
rest.v.02  _derivationally_related_form  breath.n.03
transformation.n.01  _derivationally_related_form  shift.v.05
sympathizer.n.01  _hypernym  communicator.n.01
ration.n.02  _derivationally_related_form  ration.v.01
transfix.v.02  _hypernym  pierce.v.04
chromosome.n.01  _has_part  chromatin.n.01
surface.n.02  _hypernym  boundary.n.01
accentuation.n.01  _hypernym  stress.n.01
lexicology.n.01  _derivationally_related_form  lexicographer.n.01
sensual.n.02  _derivationally_related_form  sensuality.n.01
vanish.v.02  _derivationally_related_form  disappearance.n.02
sophisticated.s.01  _also_see  worldly.s.01
```

**Figure 4.3:** A glimpse of train data file from WN18 dataset. The triplets \((h, r, t)\) can be seen separated by tab.

In case of LMC and PMC methods, beside the model initialization and loading of data files there is one more step is performed before training process begins. This step is basically responsible for formation of clusters and it is mandatory step in
both the approaches except for OADNS method. In the clusters formation, first of all the pretrained embeddings from the trained_Embeddings folder are loaded. Since the embeddings are stored based on dataset names so they can be fetched by matching with the dataset name provided in the arguments.

After the loading phase of pretrained embeddings is finished, latterly these embeddings are passed to the dimensionality reduction algorithm if the user has set --reduce_dim argument to true. The dimensionality reduction is applied primarily to overcome one of the main problem that occurs with large embeddings size, which is known as the curse of dimensionality. Curse of dimensionality problem in clustering domain essentially means that points belonging to the same cluster can be rather far apart in the high-dimensional space and projecting the space before clustering results in loss of information and possible mixing of separated clusters [35]. The impact of this problem is that it restricts the proper use of clustering algorithms in a very high-dimensional space. Since in our case the dimensions of pretrained embeddings is quite large, it is vital to perform this step to achieve the desired results after employing the clustering. There exist three different algorithms available to choose from for the purpose of dimensionality reduction that one can select with help of --reduce_dim_type argument. The algorithms can be selected by their names i.e. PCA, T-SNE and Spectral_Embedding.

```
with open(os.path.join(args.data_path, 'entities.dict')) as fin:
    entity2id = dict()
    for line in fin:
        eid, entity = line.strip().split('	')
        entity2id[entity] = int(eid)

with open(os.path.join(args.data_path, 'relations.dict')) as fin:
    relation2id = dict()
    for line in fin:
        rid, relation = line.strip().split('	')
        relation2id[relation] = int(rid)
```

**Figure 4.4:** The loading of entities and relations in the Main method.

After the dimensionality reduction, these embeddings are subsequently passed to K-means++ clustering method and based on provided argument --nclusters, $K$ number of clusters are formed. As clusters in K-means++ algorithm is formed in such a way that each point is associated with nearest most cluster so after this step each entity is going to be associated to at least and at most one nearest cluster that lies in its neighborhood.
Figure 4.5: The Method for Dimensionality Reduction and Clustering.

Eventually, after forming the clusters the results are mapped to python dictionary \( \text{dict} \) which contain the cluster centroids \( \mathcal{C} \) as keys and entities \( \mathcal{E} \) that belong to that cluster as items (equation 4.1). By mapping this way we can later able to fetch entities belonging to each single cluster in a very convenient way. The dictionary is also used for the sorting purposes. The clusters in the dictionary are afterwards sorted based on their distances \( d \) to the target entity from respective centroids.

\[
dict : \mathcal{E} \rightarrow \mathcal{C} \tag{4.1}
\]

When data loading and clustering phase is finished then comes the turn to instantiate the \textit{KGEModel}. As the \textbf{Murphy} is adapted from RotatE framework, it defines one general model in the form of \textit{KGEModel} and implements different scoring functions on top of it which makes it easy to use. Besides this, it is straightforward to plug in any new model, next to the existing ones. When the model is being initialized, in parallel the entity and relation embedding are also initialized based on the provided embedding dimensions as arguments. While talking about entity and relation embeddings dimension, it depends largely upon the scoring method in focus. Some scoring functions requires the entity embeddings to be of double in size which must be then set by providing \textit{double_entity_embeddings} argument value as \textit{true}. Models such as RotatE require the entity embeddings dimension to be double in size, so that the model can map the entities and relations to the complex vector space \([4]\). Beside RotatE, ComplEx and QuatE models also require both entity and relation both to be set as double in size which can be achieved by setting \textit{double_entity_embeddings} and \textit{double_relation_embeddings} value
to true. Beside these, other scoring models do not have this requirement, such as TransE and DistMult. These kind of mandatory checks are also performed in the model initialization phase so that all requirements for desired model should me met before the execution begins.
4.6 Generating Negative Triplet

This section will shed light on negative triplet formation of clustering based approaches PMC and LMC in detail. After the initial step of cluster formation process which is required for these two approaches (briefly discussed in section 4.5) is concluded, and the procedure of negative triplet formation comes in action. The negative triplet formation process begins as soon as the \texttt{TrainDataset()} function loads the positive triplet from the directory. The negative triplet generation is required because of the fact that the scoring function of KGE learning task required both positive and negative facts together. By utilizing these facts, the model afterwards generate the embeddings by ranking the positive triplets higher than than false ones. The formation of negative samples varies in both of the approaches so each will be discussed separately.

4.6.1 Using Language Model Embeddings

In this approach, when each positive triplet is loaded by the dataloader in the special method called as \_\_getitem\_\_(), head or tail entity corruption of the triple is decided uniformly by using the mode variable i.e. head-batch and tail-batch similar to the UNS. If mode is head-batch then tail corruption would be performed otherwise head entity of the triple would be corrupted. After choosing the entity that needs to be perturb, the associated embedding vector are fetched from entity embedding of the model. Now this fetched embedding are reshaped to the desired one so that subsequent operation like distance calculation could be performed efficiently using the Numpy python package.

Afterwards, in the next step, euclidean distance is computed from the embedding to each cluster centroid. This is done primarily so that the clusters can be sorted based on proximity or closeness to the entity that needs to be perturb. After sorting, The $H$ number of clusters will be selected from sorted list based upon the provided khop argument. If the $H$ (khop) is set to two then nearest two clusters would be chosen for each positive triple corruption. The whole process of negative triple generated in terms of python code is given in the figure 4.7.
After the selection of clusters the entities present in those clusters are grouped together in a Numpy array. These groups of entities that are gathered are possibly the best candidates for the entity that needs perturbation as we assume that they are the most similar entities obtainable in the entire dataset for that given entity. Then subsequently from these collected entities randomly $N$ number of negative samples are chosen based on the provided argument.

After grouping the entities certain checks are performed afterwards to filter the entities further. One of them is to examine whether the negative triple formed after replacing the candidate’s entities at the selected position $(head, tail)$ exists in the training set or not. If the triple exists in the training set then the generated negative triple is to be rejected and only those which are not present are making their way to the final stage which is score and loss calculation. The `forward()` function send triples to corresponding score function of the KGE model to obtain the scores and finally the back-propagation is performed on the mini-batches which is demonstrated in figure 4.11.

**Figure 4.7:** The Negative Triplet Generation Process in LMC Approach.
4.6.2 Using Uniform Trained Embeddings

The method of generating negative samples using a clustering approach with Uniform trained embeddings is referred to as PMC in this framework. This approach is slightly different from LMC approach in the negative sample generation process mainly in two aspects: First difference is that instead of employing the language model PMC utilizes the uniformly trained embeddings and the second difference will be highlighted in coming below paragraphs.

In the initial phase using a similar method to as of LMC i.e. decision of whether the head or tail corruption is to be performed are decided in a uniform fashion. In contrast to the LMC approach instead of finding the nearest $H$ clusters, the chosen entity is looked through in each individual cluster for its presence. During the searching process, when the entity is found to be present in some cluster, all entities from that cluster are taken out and are stored in a list. So the fetched entities act as candidate entities for the entity to be corrupted. Then out of all these entities, $N$ randomly negative samples are chosen and finally, negative samples are generated for the triple.

The generated negative samples are then passed through several different checks. Each generated negative sample is inspected to see whether the generated negative sample is already present in training data as a positive triple. If it exists then that newly generated negative sample is discarded from the list. The rest of the negative samples are also examined one by one and at the end only those would be left that do not present in the training set. The other thing that is considered while generating negative samples is that these generated samples should not be repeated as it would affect the optimization process.

In the end, the ones that passed through all checks made their way to the scoring function along with positive triples. The detailed step by step procedure which is described can be seen in figure 4.8 in form of code. This second difference that exist in this approach comparison to LCM approach is that in this technique, for each of the positive triple to be corrupted only one cluster is taken into account where the entity itself lies and entities from that single cluster take part in the negative sample generation. If, the entities in that cluster are less than the required negative sample size, the candidate entities for corruption are fetched randomly from all the entities available in KG.
Searching and retrieving the entities from matched clusters

Figure 4.8: The negative triplet generation process using the Uniform Trained Embeddings (PMC) method.

4.7 Negative Sampling with Optimized Affinity

Negative Sampling with Optimized Affinity is a dynamic vector based method that works based on cosine similarity score. In the initial step, for each positive triple Bernoulli sampling is utilized to decide whether to perform head or tail corruption. This process is repeated for all positive triples present in a given batch. Then positions (head or tail) in each triple that need to be corrupted are stored in the PyTorch tensor data structure. This process can be seen in figure 4.9.

After this step, the embedding vectors for all entities that need to be corrupted are obtained in a matrix based on the indices value of each entity. The newly formed
Use of Bernoulli sampling to decide head/tail corruption

Getting the embedding vectors for entities

Calculating Cosine Score and Affinity vector

Figure 4.9: The Negative Triplet Formation in OADNS Method.

matrix has the dimension equal to \((batchSize \times embeddingDim)\). The embedding vectors for all entities present in the dataset are also fetched and stored in a different matrix based on the entities index values. The shape of this formed matrix is equivalent to \((NoEntites \times embeddingDim)\). Then using the python @ operator cosine score is calculated for the whole batch simultaneously. Using this method for score calculation saves a lot of time because this operation uses GPU in very efficient manner to fasten up the computing. The dimensions of the resulted score matrix are equal to \((batchSize \times NoEntites \times embeddingDim)\).

When cosine calculations are finished, then the cosine score matrix is iterated through and the affinity vector for each triple is calculated. The affinity vector is
implementation

Checking if generated negative samples exist in Dataset or not

Figure 4.10: The method to filter the generated negative samples.

calculated by dividing each score present in a matrix row by the sum of all scores in that row. The computed affinity vectors represents the probability vectors for the entities available. Then using the affinity vectors as the base probability vector in Numpy `random.choice()` method, randomly C replaceable entities are pulled out utilizing the already computed affinity probabilities [7]. Then the earlier selected head or tail corruption positions are replaced by these fetched entities and N negative samples are formed. Finally, these generated negative samples are checked whether the formed negative triples already exist in the dataset or not. This operation is done efficiently and collectively so that processing time could be minimized. The code for this method is shown in figure 4.10.

4.8 Score and Loss Calculation

After the generation of negative triples with the help of positive triples, everything is ready for calculating the score of the triplets. The `train_step()` method is responsible to pass the generated positive and negative triplets to `forward()`
method. In forward method first embedding vectors for each triple are obtained using the indices from the entity and relation embeddings. Then these obtained entity vectors are then send to chosen scoring function for score calculation. The score function is called from within the forward method and computed scores are stored in a variable.

```python
def train_step(model, optimizer, train_iterator, args):
    model.train()
    optimizer.zero_grad()
    positive_sample, negative_sample, subsampling_weight, mode = next(train_iterator)
    if args.cuda:
        positive_sample = positive_sample.cuda()
        negative_sample = negative_sample.cuda()
        subsampling_weight = subsampling_weight.cuda()

    negative_score = model((positive_sample, negative_sample), mode=mode)
    if args.adversarial_sampling:
        negative_score = F.softmax(negative_score = args.adversarial_temperature, dim=1).detach()
        negative_score = F.logsigmoid(-negative_score).sum(dim=1)
    else:
        negative_score = F.logsigmoid(-negative_score).mean(dim=1)

    positive_score = model(positive_sample)

    positive_score = F.logsigmoid(positive_score).squeeze(dim=1)

    if args.unlabeled:
        positive_sample_loss = - positive_score.mean()
        negative_sample_loss = - negative_score.mean()
    else:
        positive_sample_loss = - (subsample_weight * positive_score).sum() / subsampling_weight.sum()
        negative_sample_loss = - (subsample_weight * negative_score).sum() / subsampling_weight.sum()

    loss = (positive_sample_loss + negative_sample_loss) / 2
```

![Call to forward function](Call to forward function)

![Calculation of loss](Calculation of loss)

Figure 4.11: The iterative training process: Score and Loss calculation.

Some of the scoring functions are available that can be used in the framework are discussed in section 2.3. For the sake of simplicity, some of the popular scoring methods are presented and discussed. The resulted positive and negatives scores are received back in `train_step()` function. As a next step these scores are used to calculate overall loss of the triplets. Based on selected loss function as described in section 2.4 the total loss is computed. A regularization would also be applied afterwards if in case it is provided. Then finally gradients of loss with respect to each parameter is computed by invoking `loss.backward()` method available in Autograd module\(^{10}\) of PyTorch. Last but not the least the calculated gradients are then used to update the each parameter value when `optimizer.step()` is invoked.

\(^{10}\)https://pytorch.org/docs/stable/autograd.html
Chapter 5

Evaluations and Results

This chapter focuses on the evaluations results of proposed negative sampling techniques with different scoring functions on standard benchmark datasets. The chapter also addresses the research question and analyzes each of them with facts and findings. The section 5.1 describes the evaluations metrics used to assess and compare the model performances to other standard approaches. Section 5.2 put light on different benchmark datasets that are used to execute experiments. The details and characteristics of each benchmark dataset are also separately exhibited in the table 5.1. Section 5.3 describes the hyperparameter settings used to obtain the results for each approach on all the individual datasets. Section 5.4 is dedicated to present the results and performances of individual approaches along with the comparison to some of the other standard methods that exist in the negative sampling domain. Further in this section, subsection 5.4.1 brings out the comprehensive performance comparison of our proposed three approaches with existing ones with the help of standard evaluation matrices. while subsection 5.4.2 highlights the motives of choosing the language model and sheds light with the help of illustration on the strength of the language model used in this work. Subsequently, subsection 5.4.3 compares the LMC approach to a UNS approach with the help of employing the clustering method on the generated embeddings from each approach. Comparing the clusters in this way shows the robustness of the LMC approach against the Uniform sampling approach. Lastly, further in this chapter the generated negative samples of LMC, PMC, and OADNS approaches are compared with SANS and UNS approaches and are depicted in the tables for the detailed comparison.
5.1 Evaluations Matrices

The standard evaluation criterion for knowledge graph embedding models consists of five different metrics these are: mean rank (MR), mean reciprocal rank (MRR), hits@10, hits@3, and hits@1. Mean rank is the average rank assigned to the true triple over all test triples. Mean reciprocal rank is the average of the reciprocal rank assigned to the true triple [43]. While hit@k indicates the percentage of occurrence of true triples ranked against the corrupted triples at top k places. In this work, after the training process is finished the models are evaluated on test data triples and subsequently, all the above mentioned matrices are computed. The target of the knowledge graph embedding task is to get a high mean reciprocal rank and hits@k with a low mean rank score.

5.2 Datasets

The experiments are performed on standard benchmark datasets which consist of Nations, UMLS, WN18RR, WN18, and CoDEx (S-M-L).

**WN18** is derived from WordNet [1] WordNet contains large lexical information of English language. It stores the information in a unique graph structure comprise of synsets that represents various parts of speech i.e. Nouns, verbs, adjectives and adverb. They are interlinked with the help of lexical relations. WordNet groups the various lexical variant of same concept into one umbrella which makes it a really powerful lexical collection available in current time. WN18 is a subset of WordNet and was prepared by Bordes et al [44]. WN18 consist of 18 unique relation and has about 41,024 entities.

**WN18RR** is another very popular benchmark dataset. The dataset was prepared by Dettmers et al [45]. It is a subset of WN18 and is created by removing the inverse relations from WN18. By creating this dataset they overcome the test leakage problem present in WN18 that were existing in terms of inverse relations. The dataset with test leakage problem poses a great challenge in knowledge graph based learning tasks as the model simply tries to learn the very straightforward rule i.e. which relations are inverse of each other. WN18RR contains 40,943 entities and has 11 relations.

**CoDEx** dataset is extracted from Wikidata [46] a basically a sister project of Wikipedia. Wikidata since its inception has gained tremendous popularity and is

---

[1] https://wordnet.princeton.edu/
one of the largest collections of free knowledge available to research communities comprising of more than 400 million statements and about more than 45 million entities [47]. The reason of the popularity of CoDEx is that the other available benchmark datasets are derived from Freebase and they have the problem of data leakage while in CoDEx all inverse relations are eliminated to prevent the problem of data leakage. CoDEx dataset essentially consists of three datasets families CoDEx-S, CoDEx-M, CoDEx-L. CoDEx-S is relatively smaller among the other two and has about 36k triplets. It can be used for testing purposes and also for hyperparameter tuning. CoDEx-M is bigger in size and has about 17034 entities and 206k triplets. CoDEx-L which is in size comparable to FB15k [3] dataset is the biggest among the others and can be used for training and evaluations.

**FB15k-237** is created by Toutanova et al [48]. This was prepared to address the issue of data leakage present in FB15K. The dataset is prepared by pulling out the most frequently used relation and then afterwards through examination only one set of inverse or duplicate relations are kept and rest of data was filtered out. The dataset that afterwards comes into existence consist of 237 relations and 14,541 entities.

**Domain Specific Datasets** Nations [49] and UMLS [50] are very small datasets in size compared to the above mentioned ones and are very domain specific. These datasets although are small but they pose a great challenge towards embedding learning based tasks and therefore it is vital to test the model robustness using these datasets. The complete information about all datasets can also be seen in table 5.1.

<table>
<thead>
<tr>
<th>Datasets</th>
<th></th>
<th></th>
<th>Total Triples</th>
<th>Train Triples</th>
<th>Validation Triples</th>
<th>Test Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>WN18</td>
<td>40943</td>
<td>18</td>
<td>151,442</td>
<td>141,442</td>
<td>5,000</td>
<td>5,000</td>
</tr>
<tr>
<td>WN18RR</td>
<td>40943</td>
<td>11</td>
<td>93,003</td>
<td>86,835</td>
<td>3,034</td>
<td>3,134</td>
</tr>
<tr>
<td>Nations</td>
<td>14</td>
<td>55</td>
<td>1,992</td>
<td>1,592</td>
<td>199</td>
<td>201</td>
</tr>
<tr>
<td>UMLS</td>
<td>135</td>
<td>46</td>
<td>6,529</td>
<td>5,216</td>
<td>652</td>
<td>661</td>
</tr>
<tr>
<td>FB15k-237</td>
<td>14,541</td>
<td>237</td>
<td>310,116</td>
<td>272,115</td>
<td>17,535</td>
<td>20, 466</td>
</tr>
<tr>
<td>CoDEx-S</td>
<td>2,034</td>
<td>42</td>
<td>36,543</td>
<td>32,888</td>
<td>1,827</td>
<td>1,828</td>
</tr>
<tr>
<td>CoDEx-M</td>
<td>17,050</td>
<td>51</td>
<td>206,205</td>
<td>185,584</td>
<td>10,310</td>
<td>10,311</td>
</tr>
<tr>
<td>CoDEx-L</td>
<td>77,951</td>
<td>69</td>
<td>612,437</td>
<td>551,193</td>
<td>30,622</td>
<td>30,622</td>
</tr>
</tbody>
</table>

**Table 5.1:** The datasets statistics i.e. the number of entities and relations in each dataset, as well as the sizes of training, validation, and test sets in terms of no of triples.
5.3 Hyperparameter Settings

The preparation and implementation background of the Murphy framework is discussed in detail in the section 4.1. Here the hyperparameter settings corresponding to different models used with various datasets is discussed and for which evaluation results can be seen in the tables 5.2 5.3 5.4 5.5. Initially to determine the optimal range of parameters the grid search technique is executed, over the range of hyperparameters for each dataset against each scoring model. The hyperparameters range that is grid searched are given below:

- Dimension $\in \{ x \mid x \in N, \, 100 \leq x \leq 1000\}$.
- Batch size $\in \{ x \mid x \in N, \, 64 \leq x \leq 1024\}$.
- Gamma $\in \{ x \mid x \in N, \, 4 \leq x \leq 200\}$.
- Learning rate $\in \{ x \mid x \in R^+, \, 0.0005 \leq x \leq 0.01\}$.
- Number of Clusters $\in \{ x \mid x \in N, \, 4 \leq x \leq 50\}$.
- Hop Size $\in \{ x \mid x \in N, \, 2 \leq x \leq 40\}$.
- Negative Sample Size $\in \{ x \mid x \in N, \, 3 \leq x \leq 1024\}$.

After performing the grid search, the best hyperparameters settings are estimated. Talking about hyperparameters that are finally used to run the experiments, for WN18 and WN18RR and FB15k-237 many of the hyperparameters are taken from the best settings mentioned in RotatE paper [4] and their github repository [2].

In WN18 for RotatE model, an embedding dimension $D = 500$ is used and gamma $\gamma = 12.0$, adversarial temperature (for training with adversarial negative sampling) $\tau = 0.5$, learning rate $\alpha = 0.0001$, batch size $B = 512$, and number of negative samples $N$ is set to 100. Additionally, our models hyperparameters are: total number of clusters $K = 10$, max hop $H = 3$. In WN18RR for RotatE model we are using the same settings except the gamma $\gamma = 6$, the learning rate $\alpha = 0.00005$, and the number of negative samples $N = 100$ (for adversarial case $N = 50$). In FB15k-237 RotatE model we used same hyperparameters as stated above except the gamma is set to $\gamma = 9.0$, adversarial temperature (for training with adversarial negative sampling) $\tau = 1.0$, learning rate $\alpha = 0.00005$ and hop size $H = 5$, is used in both cases with and without adversarial sampling.

[2]https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding
In WN18 with TransE model, an embedding dimension $D$ is set to 500 and gamma $\gamma = 12.0$, adversarial temperature (for training with adversarial negative sampling) $\tau = 0.5$, learning rate $\alpha = 0.0001$, batch size $B = 512$, number of negative samples $N = 100$ (without adversarial negative sampling), 50(with adversarial negative sampling). Using adversarial negative sampling the number of negative samples is $N = 50$, the number of clusters $K = 10$, and hop size $H = 3$. The number of negative samples is $N = 100$, the number of clusters $K = 20$, and hop size $H = 2$ is used when no adversarial sampling is employed. With WN18RR dataset using TransE model we use embedding dimension $D = 400$, batch size $B = 1024$, number of clusters $K = 10$, learning rate $\alpha = 0.0005$, number of negative samples $N = 50$, $\gamma = 6$, hop size $H = 2$. With TransE model in FB15k-237 dataset we use embedding dimension $D = 500$, batch size $B = 512$, number of clusters $K = 10$, learning rate $\alpha = 0.00005$, number of negative samples $N = 50$, hop size $H = 5$.

In WN18 with DistMult model, an embedding dimension $D$ is set to 1000 and gamma $\gamma = 200.0$, adversarial temperature $\tau = 1.0$, learning rate $\alpha = 0.001$, batch size $B = 512$, number of negative samples $N = 100$, number of clusters $K = 10$, and hop size $H = 5$ and regularization $= 0.00001$ is used in both adversarial and non adversarial cases. With WN18RR dataset using DistMult model same hyperparameters are used except learning rate $\alpha$ is set to 0.002.

In all the above detests with LMC and PMC approach the reduced dimension hyperparameter for PCA is set to $Z = 200$.

In case of Nations and UMLS datasets for TransE, RotatE, DistMult, ComplEx and QuatE model we have used an embedding dimension $D = 100$, gamma $\gamma = 12.0$, learning rate $\alpha = 0.01$, batch size $B = 64$, number of negative samples $N = 3$ is used for Nations and in case of UMLS dataset number of negative samples $N$ is set to 10.

Talking about additional hyperparameters used in LMC, PMC and SANS approach, In Nations dataset with TransE, ComplEx models we set the number of clusters $K = 4$ and hop size $H = 3$ and for RotatE model number of clusters $K$ is set to 6 and hop size $H = 5$. For QuatE number of clusters $K = 8$ and hop size $H = 3$ and in DistMult number of clusters $K = 7$ and hop size $H = 5$ is used. With UMLS dataset, in the case of TransE we set the number of clusters $K = 8$ and hop size $H = 6$, with RotatE model number of clusters $K = 8$ and hop size $H = 4$. In case of QuatE and ComplEx number of clusters $K = 5$ and hop size $H = 4$ and with DistMult number of clusters $K = 8$ and hop size $H = 7$ has been used. The reduced dimension hyperparameter for PCA is set to $Z = 2$ in all cases for
In CoDEx-S, CoDEx-M and CoDEx-L with all scoring function the embedding dimension $D = 500$ is used and gamma $\gamma = 9.0$, learning rate $\alpha = 0.00005$ (for CoDEx-S $\alpha$ is set to 0.0005), batch size $B = 512$, and number of negative samples $N = 50$ has been used. In case of CoDEx-S dataset with TransE the number of clusters $K = 8$ and hop size $H = 7$, for RotatE, DistMult, QuatE and ComplEx the number of clusters $K = 10$ and hop size $H = 9$ and for DistMult the number of clusters $K = 8$ and hop size $H = 6$ has used. With CoDEx-M dataset with TransE, DistMult, ComplEx and QuatE models the number of clusters $K = 8$ and hop size $H = 7$ is used and for RotatE the number of clusters $K = 12$ and hop size $H$ is set to 9. In CoDEx-L with TransE, DistMult, ComplEx and RotatE models the number of clusters $K = 10$ and hop size $H = 9$ is used and for QuatE the number of clusters $K = 10$ and hop size $H$ is set to 4. The reduced dimension hyperparameter for PCA in case of CoDEx-S is set to $Z = 100$ and for CoDEx-M and CoDEx-L reduced dimension hyperparameter $Z = 200$ has been used. For executing the experiments with Uniform RW-SANS on Nations, UMLS, Codex-S, CoDEx-M and CoDEx-L datasets, The $-nrw$ hyperparameter is set to 1000 and all results on these datasets are obtained without using negative adversarial sampling.

5.4 Result and Analysis

In this section evaluation results of the proposed techniques are presented and they also compared with other standard approaches on the various benchmarks datasets. This section will also shed light on the merits and demerits of the approaches with the help of results computed on various datasets. The results are also analyzed with various visualizations that are solely generated to back the facts and findings.

5.4.1 Performance Comparison of Different Methods

The proposed approaches i.e. LMC, PMC, and OADNS are evaluated against Uniform, SANS, NSCaching, and KBGAN. The benchmark datasets that are used for experiments are FB15k-237, WN18, WN18RR, CoDEx, Nations, and UMLS. The results for FB15k-237, WN18, and WN18RR are computed in two different settings: along with negative adversarial sampling and also without using the
adversarial negative sampling. The result are shown in the tables \ref{table:5.2} and \ref{table:5.3} respectively. Before analyzing the results, It is important to know that for all datasets the complete entity labels are fetched from different sources except for FB15k-237. In the case of FB15k-237, the labels for all the entities in the dataset could not be retrieved and for almost 35 entities the labels could not be fetched, and hence around 4926 train triples have to be skipped during the training. Similarly, test triples for which labels were not present are also skipped during testing.

Looking at the results obtained using adversarial negative sampling shown in table \ref{table:5.3} illustrates that LMC with RotatE and TransE have achieved significant improvements and beat other methods of negative sampling i.e. SANs, Uniform, NSCaching, and KGBAN most of the time in almost all performance metrics. It is also important to mention here that with RotatE all proposed approaches LMC, PMC, and OADNS get significantly higher Hit@10 scores than the rest of the approaches as depicted in the table. On the contrary, our all approaches perform poorer with DistMult on all datasets except WN18RR where LMC beats the others with a slight margin.

<table>
<thead>
<tr>
<th>KGE model</th>
<th>Algorithm</th>
<th>WN18</th>
<th>WN18RR</th>
<th>FB15K-237</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MRR H@10</td>
<td>MRR H@10</td>
<td>MRR H@10</td>
</tr>
<tr>
<td>RotatE</td>
<td>Self-Adv. Uniform*</td>
<td>0.9498</td>
<td>96.05</td>
<td>0.4760</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. SANS*</td>
<td>0.9494</td>
<td>95.85</td>
<td>0.4745</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. RW-SANS*</td>
<td>0.9496</td>
<td>96.09</td>
<td>0.4805</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. LMC (Ours)</td>
<td>0.9469</td>
<td>95.93</td>
<td>\textbf{0.4770}</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. PMC (Ours)</td>
<td>0.9476</td>
<td>96.04</td>
<td>0.4710</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. OADNS (Ours)</td>
<td>\textbf{0.9498}</td>
<td>95.97</td>
<td>0.4746</td>
</tr>
<tr>
<td>DistMult</td>
<td>Self-Adv. Uniform*</td>
<td>0.6837</td>
<td>92.94</td>
<td>0.4399</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. SANS*</td>
<td>0.7561</td>
<td>93.04</td>
<td>0.3684</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. RW-SANS*</td>
<td>0.6634</td>
<td>91.08</td>
<td>0.3836</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. LMC (Ours)</td>
<td>0.6824</td>
<td>92.90</td>
<td>0.4374</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. PMC (Ours)</td>
<td>0.4659</td>
<td>55.25</td>
<td>0.4054</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. OADNS (Ours)</td>
<td>0.1817</td>
<td>75.32</td>
<td>0.3492</td>
</tr>
<tr>
<td>TransE</td>
<td>Self-Adv. Uniform*</td>
<td>0.7722</td>
<td>92.02</td>
<td>0.2232</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. SANS*</td>
<td>0.7136</td>
<td>84.06</td>
<td>0.2249</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. RW-SANS*</td>
<td>0.7429</td>
<td>88.51</td>
<td>0.2273</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. LMC (Ours)</td>
<td>\textbf{0.7947}</td>
<td>\textbf{94.92}</td>
<td>0.2262</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. PMC (Ours)</td>
<td>0.6368</td>
<td>\textbf{95.40}</td>
<td>0.1927</td>
</tr>
<tr>
<td></td>
<td>Self-Adv. OADNS (Ours)</td>
<td>\textbf{0.8256}</td>
<td>\textbf{95.49}</td>
<td>\textbf{0.2388}</td>
</tr>
</tbody>
</table>

Table 5.2: Results of KGE models with adversarial sampling. Results for models marked by * are taken from \cite{6}.

Results of negative sampling methods with adversarial sampling are described in the table \ref{table:5.2}. From the table, it is evident that the evaluation trend with DistMult scoring method is similar to that of observed in the table \ref{table:5.3} such that all of the
### Evaluations and Results

<table>
<thead>
<tr>
<th>KGE model</th>
<th>Algorithm</th>
<th>WN18</th>
<th>WN18RR</th>
<th>FB15K-237</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR</td>
<td>H@10</td>
<td>MRR</td>
<td>H@10</td>
</tr>
<tr>
<td>KGE model</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RotatE</td>
<td>Uniform*</td>
<td>0.9474</td>
<td>96.09</td>
<td>0.4711</td>
</tr>
<tr>
<td></td>
<td>Uniform SANS*</td>
<td>0.9499</td>
<td>95.97</td>
<td>0.4769</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS*</td>
<td>0.9489</td>
<td>96.07</td>
<td>0.4796</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.9481</td>
<td>96.18</td>
<td>0.4732</td>
</tr>
<tr>
<td></td>
<td>PMC (Ours)</td>
<td>0.9480</td>
<td>96.06</td>
<td>0.4708</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.9496</td>
<td>95.93</td>
<td>0.4745</td>
</tr>
<tr>
<td>DistMult</td>
<td>KBGAN*</td>
<td>0.7275</td>
<td>93.08</td>
<td>0.2039</td>
</tr>
<tr>
<td></td>
<td>NSCaching*</td>
<td>0.8306</td>
<td>93.74</td>
<td>0.4128</td>
</tr>
<tr>
<td></td>
<td>Uniform*</td>
<td>0.4689</td>
<td>81.39</td>
<td>0.3938</td>
</tr>
<tr>
<td></td>
<td>Uniform SANS*</td>
<td>0.7553</td>
<td>93.19</td>
<td>0.4025</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS*</td>
<td>0.6235</td>
<td>89.80</td>
<td>0.4071</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.4786</td>
<td>82.93</td>
<td>0.3953</td>
</tr>
<tr>
<td></td>
<td>PMC (Ours)</td>
<td>0.4665</td>
<td>81.09</td>
<td>0.4054</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.3464</td>
<td>77.62</td>
<td>0.2789</td>
</tr>
<tr>
<td>TransE</td>
<td>KBGAN*</td>
<td>0.6606</td>
<td>94.80</td>
<td>0.1808</td>
</tr>
<tr>
<td></td>
<td>NSCaching*</td>
<td>0.7818</td>
<td>94.63</td>
<td>0.2002</td>
</tr>
<tr>
<td></td>
<td>Uniform*</td>
<td>0.6085</td>
<td>95.53</td>
<td>0.2022</td>
</tr>
<tr>
<td></td>
<td>Uniform SANS*</td>
<td>0.8228</td>
<td>95.09</td>
<td>0.2254</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS*</td>
<td>0.8195</td>
<td>95.22</td>
<td>0.2317</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.7646</td>
<td>95.57</td>
<td>0.2145</td>
</tr>
<tr>
<td></td>
<td>PMC (Ours)</td>
<td>0.6361</td>
<td>95.42</td>
<td>0.1928</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.8258</td>
<td>95.45</td>
<td>0.2390</td>
</tr>
</tbody>
</table>

Table 5.3: Results of KGE models without adversarial sampling. Results for models marked by ⋆ are taken from [6].

Approaches LMC, PMC, and OADNS performed poorer on almost every dataset laid out in the table. But in case of TransE model, LMC and OADNS performs really good and beat all others on WN18 and WN18RR datasets by fair margin. Especially looking at the the results of WN18 dataset solely with TransE and RotatE scoring methods, the performance of the the methods is quite good overall in all metrics.

Tables 5.4 5.5 depicts the results on Nations, UMLS and CoDEx datasets. The results for Uniform, Uniform SANS, and Uniform RW-SANS are obtained by setting the common hyperparameters fixed that is obtained by performing the grid search with LMC, PMC, and OADNS methods, which was mainly conducted to find the best combination of hyperparameters. For Uniform-SANS and Uniform RW-SANS, $−khop$ hyperparameter values are grid searched while keeping other hyperparameters locked. Additionally with Uniform RW-SANS the hyperparameter $−nrw$ no of random walks is set to 1000 in all the experiments that are executed with this approach.

It can be observed that generally, all three negative sampling methods performed
better with RotatE, TransE, and DistMult models across all datasets. While on DistMult and QuatE, except for the two occasions on Nations and CoDEx-M where LMC and OADNS beat others by a little margin, other than that the sampling methods did not perform well. By far look, the trend that is worth noticing is that LMC, PMC, and OADNS negative sampling techniques tend to perform generally better on small datasets compared to large ones represented in this table. As in the case of the Nations dataset with TransE method, on Hit@1 and MRR metrics all three approaches achieves really good results and beats others methods by a large margin. Almost similar trend can be seen with UMLS dataset with TransE scoring method. Generally, on Nations and UMLS datasets, all three approaches with all scoring functions performed at par in comparison to other negative sampling methods as evaluation numbers mostly are not too far behind in comparison to others. However looking at the CoDEx datsets results in table 5.5, LMC, PMC, and OADNS approaches beats others most of the times on CoDEx-M and CoDEx-L except RW-SANS approach. On CoDEx-S all the three approaches performs under par and beats others just only two times. RW-SANS approach although achieved really good performance on CoDEx datasets it is really resource hungry approach and the preprocessing time of this approach is significantly higher on large datasets.

5.4.2 Language Model Embeddings Visualization

Figure 5.1 depicts the clustering performed using the Sentence-BERT transformer-based model. For this analysis, the UMLS dataset is used due to its exclusivity. It can be seen from the figure that most of the entities are grouped properly in their respective groups which depicts the usefulness of the language model. If we analyze more thoroughly, it can be seen that entities such as 'human', mammal’ and 'reptile' are correctly classified to the same group as they are semantically similar. The identical pattern could be observed in other clusters as well. Entities like 'bacterium’, 'fish’, and 'animal’ in other groups are also related to each other and have been therefore placed in a matching group.

However, it is also observed that some of the entities are not clustered properly as well. There are entities exist also that are miss-classified as well or placed in different clusters e.g. 'virus’ and 'bacterium’ are associated with each other but have been placed in two different groups. But nevertheless from this figure, it is clearly visible that using the language model for finding the related entities makes much more sense as opposed to fetching randomly from all the entities.
<table>
<thead>
<tr>
<th>KGE model</th>
<th>Algorithm</th>
<th>Nations</th>
<th>UMLS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>MRR H@1 H@3 H@10</td>
<td>MRR H@1 H@3 H@10</td>
</tr>
<tr>
<td>RotatE</td>
<td>Uniform</td>
<td>0.6375 47.26 74.62 99.25</td>
<td>0.8599 74.05 97.80 99.69</td>
</tr>
<tr>
<td></td>
<td>Uniform SANS</td>
<td>0.6672 50.24 80.09 99.25</td>
<td>0.8616 74.28 98.03 99.62</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.6994 50.74 78.35 99.50</td>
<td>0.8691 76.02 97.80 99.62</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.6474 47.26 <strong>77.11</strong> 99.75</td>
<td><strong>0.8713 77.91</strong> 95.99 99.84</td>
</tr>
<tr>
<td></td>
<td>PMC (Ours)</td>
<td>0.5708 36.06 71.14 1.00</td>
<td>0.7596 63.76 85.32 96.67</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.6021 40.79 73.88 99.75</td>
<td><strong>0.8908 81.99</strong> 97.73 99.77</td>
</tr>
<tr>
<td></td>
<td>Uniform SANS</td>
<td>0.5688 36.31 70.14 99.50</td>
<td>0.5149 34.56 62.63 85.70</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.5842 37.81 73.88 98.75</td>
<td>0.5026 32.90 61.72 84.26</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.5142 27.36 68.40 <strong>99.75</strong></td>
<td>0.4327 25.71 51.21 80.63</td>
</tr>
<tr>
<td></td>
<td>PMC (Ours)</td>
<td>0.5523 33.58 69.40 99.50</td>
<td>0.2399 8.01 24.35 69.74</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.5614 31.59 <strong>76.61</strong> <strong>99.75</strong></td>
<td>0.5070 30.55 63.99 84.45</td>
</tr>
<tr>
<td>ComplEx</td>
<td>Uniform</td>
<td>0.6832 53.48 77.11 99.25</td>
<td>0.4518 29.87 52.79 77.53</td>
</tr>
<tr>
<td></td>
<td>Uniform SANS</td>
<td>0.6829 54.47 75.37 98.75</td>
<td>0.4556 30.25 53.10 78.29</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.6805 53.48 77.11 99.25</td>
<td>0.4543 30.86 51.96 77.63</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.6429 47.76 75.37 <strong>99.75</strong></td>
<td>0.4055 24.13 47.35 <strong>77.76</strong></td>
</tr>
<tr>
<td></td>
<td>PMC (Ours)</td>
<td>0.6625 51.99 73.88 99.00</td>
<td>0.2656 12.85 27.45 63.08</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.6728 52.98 74.87 99.00</td>
<td>0.3934 22.84 46.44 76.02</td>
</tr>
<tr>
<td>DistMult</td>
<td>Uniform</td>
<td>0.5185 31.59 63.43 97.51</td>
<td>0.5066 32.75 61.87 83.66</td>
</tr>
<tr>
<td></td>
<td>Uniform SANS</td>
<td>0.4892 27.61 63.68 95.52</td>
<td>0.5163 35.77 61.11 83.73</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.5150 29.10 65.92 97.01</td>
<td>0.4904 32.45 59.60 84.26</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.4402 21.89 51.99 <strong>98.50</strong></td>
<td>0.3752 19.88 46.14 79.27</td>
</tr>
<tr>
<td></td>
<td>PMC (Ours)</td>
<td>0.4788 25.87 59.70 <strong>98.25</strong></td>
<td>0.1833 5.44 15.20 54.76</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.4792 25.37 62.68 97.01</td>
<td>**0.5207 31.69 64.18 <strong>88.72</strong></td>
</tr>
<tr>
<td>QuatE</td>
<td>Uniform</td>
<td>0.4211 6.46 73.63 98.75</td>
<td>0.7145 52.19 95.99 99.16</td>
</tr>
<tr>
<td></td>
<td>Uniform SANS</td>
<td>0.4235 6.71 73.38 98.75</td>
<td>0.7376 51.05 96.97 99.47</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.4168 5.72 73.38 98.75</td>
<td>0.7027 55.90 96.74 99.47</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td><strong>0.4425 11.69</strong> 71.89 99.25</td>
<td><strong>0.7689 57.33</strong> 96.21 <strong>99.62</strong></td>
</tr>
<tr>
<td></td>
<td>PMC (Ours)</td>
<td>0.4264 9.70 70.14 <strong>98.75</strong></td>
<td>0.5977 32.07 86.00 96.44</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.4036 <strong>7.21</strong> 67.41 97.76</td>
<td>0.7160 47.50 95.23 <strong>99.09</strong></td>
</tr>
</tbody>
</table>

| TransE    | Uniform      | 0.4168 5.72 73.38 98.75 | 0.7027 55.90 96.74 99.47 |
|           | Uniform SANS | 0.4235 6.71 73.38 98.75 | 0.7376 51.05 96.97 99.47 |
|           | Uniform RW-SANS | 0.4168 5.72 73.38 98.75 | 0.7027 55.90 96.74 99.47 |
|           | LMC (Ours)   | **0.4425 11.69** 71.89 99.25 | **0.7689 57.33** 96.21 **99.62** |
|           | PMC (Ours)   | 0.4264 9.70 70.14 **98.75** | 0.5977 32.07 86.00 96.44 |
|           | OADNS (Ours) | 0.4036 **7.21** 67.41 97.76 | 0.7160 47.50 95.23 **99.09** |

Table 5.4: Results of different negative sampling techniques evaluated on Nation and UMLS datasets. **Bold** indicates our approaches beat others methods and bold* shows that our approach beat all others excepts Uniform RW-SANS.

Figure 5.1: Sentence-BERT transformer-based model embedding visualization.
### Evaluations and Results

#### Table 5.5: Results of different negative sampling techniques evaluated on CoDEx-S, CoDEx-M and CoDEx-L datasets. **Bold** indicates our approaches beat others methods and **bold*** shows that our approach beat all others excepts Uniform RW-SANS.

<table>
<thead>
<tr>
<th>KGE model</th>
<th>Algorithm</th>
<th>CoDEx-S</th>
<th>CoDEx-M</th>
<th>CoDEx-L</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR H@1 H@3 H@10</td>
<td>MRR H@1 H@3 H@10</td>
<td>MRR H@1 H@3 H@10</td>
<td></td>
</tr>
<tr>
<td>RotatE</td>
<td>Uniform</td>
<td>0.4727 35.66 53.25 69.17</td>
<td>0.3914 31.07 42.64 54.13</td>
<td>0.4142 34.52 44.94 54.23</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.4631 34.62 52.57 68.62</td>
<td>0.4097 34.07 43.78 54.46</td>
<td>0.4228 35.37 45.67 55.14</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.4738 36.07 53.11 69.36</td>
<td>0.3832 32.59 43.77 53.38</td>
<td>0.4063 34.84 45.03 54.42</td>
</tr>
<tr>
<td></td>
<td>PCM (Ours)</td>
<td>0.4807 38.78 52.13 67.12</td>
<td>0.4081 33.53 42.93 52.16</td>
<td>0.4372 37.89 46.54 54.64***</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.4038 28.90 47.04 63.70</td>
<td>0.3267 27.48 42.89 54.83</td>
<td>0.3563 30.22 43.24 54.53</td>
</tr>
<tr>
<td>CompEx</td>
<td>Uniform</td>
<td>0.4812 38.24 54.02 68.27</td>
<td>0.3315 26.41 35.85 46.23</td>
<td>0.3642 30.44 38.97*** 48.06***</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.4757 38.18 54.61 66.66</td>
<td>0.3201 27.09 35.45 45.87</td>
<td>0.3277 28.30 29.69 38.84</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.4806 37.22 53.51 65.48</td>
<td>0.3412 27.46 36.37 45.17</td>
<td>0.3412 28.41 36.59 45.36</td>
</tr>
<tr>
<td></td>
<td>PCM (Ours)</td>
<td>0.2273 21.63 23.27 34.37</td>
<td>0.3123 26.01 33.40 41.32</td>
<td>0.3111 27.90 35.47 43.24</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.2621 17.36 26.44 46.94</td>
<td>0.3267 27.29 35.80 45.92</td>
<td>0.3642 30.44 38.97*** 48.06***</td>
</tr>
<tr>
<td>DistMult</td>
<td>Uniform</td>
<td>0.4521 34.95 49.69 65.67</td>
<td>0.3108 24.59 33.40 43.93</td>
<td>0.3173 25.98 33.90 43.05</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.4561 35.96 49.39 65.72</td>
<td>0.3123 24.60 33.65 44.16</td>
<td>0.3337 27.21 36.07 45.35</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.4051* 35.74* 50.00 65.31</td>
<td>0.3092 24.49 33.08 43.77</td>
<td>0.3085 26.16 34.45 43.14</td>
</tr>
<tr>
<td></td>
<td>PCM (Ours)</td>
<td>0.2161 19.14 22.26 36.03</td>
<td>0.3125 26.01 33.40 41.32</td>
<td>0.3111 27.90 35.47 43.24</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.2621 17.36 26.44 46.94</td>
<td>0.3125 26.01 33.40 41.32</td>
<td>0.3111 27.90 35.47 43.24</td>
</tr>
<tr>
<td>QuatE</td>
<td>Uniform</td>
<td>0.5009 41.05 54.07 68.13</td>
<td>0.2860 28.30 36.96 45.24</td>
<td>0.3455 28.46 36.86 45.24</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.4822 39.22 52.16 66.05</td>
<td>0.3958 32.78 42.64 52.37</td>
<td>0.4288 36.83 43.52 53.95</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.4516 38.26 51.23 63.70</td>
<td>0.3461 28.58 37.44*** 46.16***</td>
<td>0.3461 29.28 37.20 45.17</td>
</tr>
<tr>
<td></td>
<td>PCM (Ours)</td>
<td>0.3324 29.99 34.40 44.57</td>
<td>0.3249 27.44 34.70 42.45</td>
<td>0.3301 30.79 38.13 45.80</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.3779 27.84 40.70 58.20</td>
<td>0.3689* 30.59* 39.49* 48.81*</td>
<td>0.3915* 33.22* 41.72* 50.31***</td>
</tr>
<tr>
<td>TransE</td>
<td>Uniform</td>
<td>0.4381 31.18 50.27 67.45</td>
<td>0.3743 30.09 40.64 51.39</td>
<td>0.4000 33.19 41.48 52.66</td>
</tr>
<tr>
<td></td>
<td>Uniform RW-SANS</td>
<td>0.4223 31.01 50.98 67.77</td>
<td>0.3753 30.28 40.67 51.28</td>
<td>0.3737 29.30 29.69 38.84</td>
</tr>
<tr>
<td></td>
<td>LMC (Ours)</td>
<td>0.4285 30.27 49.12 67.17</td>
<td>0.3871 31.10 42.26 52.82</td>
<td>0.4180 35.02 45.17 54.32</td>
</tr>
<tr>
<td></td>
<td>PCM (Ours)</td>
<td>0.3324 29.99 34.40 44.57</td>
<td>0.3249 27.44 34.70 42.45</td>
<td>0.3301 30.79 38.13 45.80</td>
</tr>
<tr>
<td></td>
<td>OADNS (Ours)</td>
<td>0.3531 21.74 41.71 62.71</td>
<td>0.3886 31.22 42.43 53.04</td>
<td>0.4196 35.47 44.93*** 54.28***</td>
</tr>
</tbody>
</table>

#### 5.4.3 Trained Embeddings Graphical Comparison

This section will demonstrate the difference between generated embeddings from two different negative sampling techniques. For the comparison achieved through the aid of the clustering method, visualization of two different trained embeddings is done, one embedding is obtained using LMC based approach and it is compared against embeddings that use a uniform negative sampling approach. Using the figure 5.2 we try to analyze how trained embeddings visualization actually could support or complement the results depicted in the tables 5.3 and 5.2.

For running this demonstration, the scikit-learn [37] library is used which is available in python. The UMLS dataset is used for this experiment [50]. First of all, the K-means++ algorithm is executed to obtain the desired clusters from the trained embeddings. Then with the use of the T-SNE algorithm which is also present in scikit-learn, the dimensionality of data is reduced so that it can effortlessly be visualized in two dimensions. The no of clusters K is set to 5 and a total of 20 entities are plotted in the figure.

It’s worth mentioning here that these embeddings are specifically used for visualization and may not be trained for the optimal number of steps but for both of
them, the same hyperparameters are used so that the comparison can be done side by side. For the clusters visualization, Seaborn [51] library is utilized as it is quite renowned for interactive data representation and informative illustrations. Finally, the Convex hull is drawn around the cluster data points with the aid of the Scipy library [52] method to depict the boundary of clusters. These visualizations are generated with the help of the Trained_Embedding_Visualizer code repository available on GitHub.

![Image of visualizations showing clustering results]

**Figure 5.2:** The clustering results of trained Embeddings employing Uniform and LMC based negative sampling technique on the UMLS dataset.

[^3]: https://github.com/NIMI-research/Trained-Embedding-Visualizer
Looking at the figure 5.2, it clearly shows that the clustering results of LMC are quite meaningful compared to the UNS one. It also depicts that the groups of data points obtained using the language model-based negative sampling approach are more coherent against the uniform embeddings. In LMC embeddings the placement of three semantically related entities bacterium, virus, and animal is correctly made in the same cluster showing that they are somehow associated with each other. In comparison, Uniform negative sampling based embeddings are showing quite different behavior and the three mentioned entities are put in three different clusters as opposed to placing them in the individual cluster. This evidently illustrates that LCM based embeddings are more robust and more enriched and hence therefore, K-means++ algorithm is able to cluster the entities in a better way. Whereas in Uniform sampling based embeddings clustering of entities makes less sense as entities although related to each other but has positioned in separate clusters. These also reflects in the results obtained using UMLS dataset in table 5.4, as LMC approach performs much better in comparison to the Uniform Negative Sampling.

5.4.4 Analysis of Generated Entity Candidates

To compare the Uniform, SANS, and proposed approaches in this work named as LMC, PMC and OADNS more rigorously, the negative samples generated by each method are compared, analyzed and subsequently presented in the tables 5.6, 5.7. For carrying out these evaluations, the WN18RR dataset is used. The $K$ in the case of the LMC approach refers to the no of hop size $H$ taken into consideration for the triplet formation. All results of these approaches are without employing the negative self-adversarial negative sampling. The results of SANS and Uniform negative sampling are obtained from [6].

<table>
<thead>
<tr>
<th>Anchor Nodes</th>
<th>Uniform</th>
<th>Uniform SANS</th>
<th>Uniform LMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>arachnoid</td>
<td>dimer</td>
<td>arachnidae</td>
<td>vertebrata</td>
</tr>
<tr>
<td></td>
<td>refusal</td>
<td>biology</td>
<td>pteractylidae</td>
</tr>
<tr>
<td>landscape</td>
<td>arthropod</td>
<td>arachnoid</td>
<td>myxiniformes</td>
</tr>
<tr>
<td>rise</td>
<td>wolf spider</td>
<td>subkingdom</td>
<td>sphyngidae</td>
</tr>
<tr>
<td>namer</td>
<td>garden spider</td>
<td>placodermi</td>
<td>elopidae</td>
</tr>
<tr>
<td>empathy</td>
<td>beach pos</td>
<td>sympathy</td>
<td>regular</td>
</tr>
<tr>
<td></td>
<td>rattus</td>
<td>expectation</td>
<td>succeed</td>
</tr>
<tr>
<td></td>
<td>albinus</td>
<td>passion</td>
<td>promiser</td>
</tr>
<tr>
<td></td>
<td>microcrini</td>
<td>commiserate</td>
<td>ilessness</td>
</tr>
<tr>
<td></td>
<td>banking industry</td>
<td>commiserate Left</td>
<td>pleasurable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pride</td>
<td>bloesedness</td>
</tr>
<tr>
<td>wheat</td>
<td>lend</td>
<td>have a bun in the oven</td>
<td>existent</td>
</tr>
<tr>
<td></td>
<td>align</td>
<td>empathy</td>
<td>humaness</td>
</tr>
<tr>
<td></td>
<td>dooclad</td>
<td>regular</td>
<td>parturienity</td>
</tr>
<tr>
<td></td>
<td>mismanage</td>
<td></td>
<td>oxide</td>
</tr>
<tr>
<td></td>
<td>semiconductor device</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>buckwheat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Candidate Nodes</th>
<th>Uniform</th>
<th>Uniform SANS</th>
<th>Uniform LMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>arachnid</td>
<td>dimer</td>
<td>arachnidae</td>
<td>vertebrata</td>
</tr>
<tr>
<td>biologic</td>
<td>rostracoda</td>
<td>pectognathis</td>
<td>pteractylidae</td>
</tr>
<tr>
<td>rastracoda</td>
<td>arachnidl</td>
<td>antidae</td>
<td>myxiniformes</td>
</tr>
<tr>
<td>subkingdom</td>
<td>arachnidae</td>
<td>amphiban_family</td>
<td>sphyngidae</td>
</tr>
<tr>
<td>placodermi</td>
<td>arachnidae</td>
<td>placoides</td>
<td>elopidae</td>
</tr>
<tr>
<td>scyphons</td>
<td>amphiban_family</td>
<td>categories</td>
<td>elopidae</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>elopidae</td>
</tr>
</tbody>
</table>

Table 5.6: Example set of candidate entities to form a negative triplet given a entity by Uniform, SANS and LMC approaches.
It can be observed by looking at table 5.6 that negative samples generated by LMC are generally more meaningful as compared to SANS and Uniform on all $K$ values. Distinctly at $K=2$ the generated samples from the LMC approach are semantically more related to each other in comparison to other methods and make more sense when looked at in resemblance to each other. We also observed the meaningfulness tends to decrease as we increase the hop size $H$ from 2 to 4. By looking at LMC samples in the table 5.6 it can clearly be noticed that at $K=4$ the candidates generated tend to be less similar as opposed to generated using hop size 2. So from here, two things can be concluded first that the LMC approach final results (as depicted in tables 5.2, 5.4, 5.3) are better in comparison to others because of the fact that the samples created by LMC are more relevant in contrast to others. Secondly, the no of hop size $H$ in the LMC approach has a crucial role in determining the model performance.

<table>
<thead>
<tr>
<th>Anchor Nodes</th>
<th>Uniform</th>
<th>Uniform SANS</th>
<th>Uniform PMC</th>
<th>Uniform OADNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>arachnoid</td>
<td>arachnida</td>
<td>biological</td>
<td>pleopodite</td>
<td>theropoda</td>
</tr>
<tr>
<td></td>
<td>refusal</td>
<td>rostracoda</td>
<td>nereidae</td>
<td>opisthobranchia</td>
</tr>
<tr>
<td></td>
<td>landscape</td>
<td>subkingdoms</td>
<td>amphipodium</td>
<td>annelida</td>
</tr>
<tr>
<td></td>
<td>nurse</td>
<td>placoidea</td>
<td>pelecypodiformes</td>
<td>amphineura</td>
</tr>
<tr>
<td></td>
<td>garden spider</td>
<td>scyphidea</td>
<td>categorina</td>
<td>cestida</td>
</tr>
<tr>
<td>empathy</td>
<td>sympathetic</td>
<td>cheerlessness</td>
<td>sorrowful</td>
<td>cerebro</td>
</tr>
<tr>
<td></td>
<td>sympathy</td>
<td>expectation</td>
<td>altruistic</td>
<td>merginase</td>
</tr>
<tr>
<td></td>
<td>feeling</td>
<td>passion</td>
<td>isolationist</td>
<td>giant</td>
</tr>
<tr>
<td></td>
<td>commiserate</td>
<td>pride</td>
<td>pride</td>
<td>galilean</td>
</tr>
<tr>
<td></td>
<td>banking industry</td>
<td>commiseration</td>
<td>attribute</td>
<td></td>
</tr>
<tr>
<td>wheat</td>
<td>lentil</td>
<td>fastfood</td>
<td>Edirne</td>
<td>rachius</td>
</tr>
<tr>
<td></td>
<td>align</td>
<td>salad</td>
<td>United States</td>
<td>amnt_grass</td>
</tr>
<tr>
<td></td>
<td>doodad</td>
<td>mess</td>
<td>fixings</td>
<td>setaria</td>
</tr>
<tr>
<td></td>
<td>mimmange</td>
<td>stodge</td>
<td>form</td>
<td>alopecurus</td>
</tr>
<tr>
<td></td>
<td>semiconduct device</td>
<td>meal</td>
<td>Iraqi Kurdistan</td>
<td>cereal</td>
</tr>
</tbody>
</table>

**Table 5.7:** Example set of candidate entities to form a negative triplet given a entity by Uniform, SANS, PMC and OADNS approaches.

Now coming towards the second table 5.7. It shows the comparison of Uniform, SANS with our two remaining approaches PMC and OADNS. For this evaluation also, we used the same WN18RR dataset and also we select the same entities used chosen in table 5.6 as anchor nodes for triplet generation so that we can compare and analyze results more rigorously. by looking at the candidate’s entities generated, It can be observed that the PMC candidates are slightly more coherent and meaningful in relation to OADNS and SANS and Uniform approaches. But in the case of OADNS, we discover that entities do not make much sense when looked at thoroughly in comparison to other methods. We recognize this can be due to the fact that since OANS is a dynamic vector based approach so the candidates generated in the beginning are less meaningful but as training progress, the candidates tend to get better with time.
Chapter 6

Conclusion and Future Work

In this thesis work, three novel approaches for negative sample generation are developed and presented. Besides this, the employment of the language models for negative sampling are also investigated. Based on our knowledge this is the first time that language models usability has been tested in the domain of negative sampling. Furthermore, the clustering strategy is probed on top of pretrained models in order to obtain meaningful groups of entities in form of clusters. This is done to leverage the semantic relationship between entities and to generate better negative candidates.

To examine the performance of the developed approaches, the approaches are tested with different scoring functions: RotatE, QuatE, ComplEx, DistMult, and TransE. The multiple benchmark datasets are used to carry out the experiments, to fully test the efficacy of the approaches. Last but not least, the performances of the proposed approaches are compared to other standard negative sample generation methods to analyze them further.

That being said, this chapter which is the last one of this work is dedicated to present and summarize the main contributions of the work. The focus of the thesis are initially laid down in form of a research questions. So in the chapter, the main contributions against each research question will be put forward and discussed by providing the answers to those question.

So section 6.1 will put light on the research questions and discussion on each individual question will be put forward. While section 6.2 gives the outlook on the future work that can be possible further after this thesis research work.
6.1 Contributions

In the beginning, the discussion was started by forming three research questions. For sake of concluding the things, those question will be redefined here.

**Research Question 1:** Do different negative sampling techniques effect the performance of KGE models?

In the exploration of this research direction, an extensive study has been done on understanding the existing negative sampling approaches. Furthermore, they have been tested on top of the benchmark knowledge graphs and compared to the proposed approach. The results of the evaluations shows the high impact of negative sampling on the performance of KGE models. On top, apart from the performance, the meaningfulness of the negative sampling approach play a great role which was deeply studied in this thesis.

**Research Question 2:** Can we leverage language models for generating negative samples?

To answer this questions we conducted systematic research that also covers the first research question, three different negative sampling techniques are developed under the umbrella of Murphy framework. The first technique called as LMC, in which language models are employed along with clustering for negative instance generation. In the second approach, Uniform pretrained embeddings are tested and examined alongside the clustering approach for the negative sampling generation procedure. The last approach that is developed in the Murphy framework is named OADNS, in which the existing ADNS approach described in section 2.5.3 is optimized and the computation time of the approach is squeezed, enabling the approach to be tested on larger datasets. For each of the approaches, extensive experiments are executed on the standard benchmark datasets to check their performances.

From the evaluation results, it comes out as the LMC approach that employed language models generally outperforms the other two methods in most of the experiments and the other two methods PMC and OADNS mostly perform at par in comparison to other approaches. For the sake of comparison, the proposed methods are tested and analyzed to other standard approaches such as Uniform
and SANS for better analysis. From inspection, it becomes evident that with RotatE the LMC method outperforms the other approaches on the two benchmark datasets (WN18, WN18RR), in the cases when no adversarial sampling is utilized. The same trend exists with the TransE model when no adversarial sampling is used.

In the experiments done with adversarial sampling, it emerges out as UNS performs better on WN18 and FB15k-237 with RotatE model and while LMC performs best on WN18 and WN18RR with the TransE model. Since on FB15k-237, entity labels for all the entities could not be retrieved for this dataset (as 4926 train triples are skipped) therefore all the proposed approaches slightly underperform on FB15k-237 in both cases, with and without adversarial negative sampling which is also visible in the tables 5.2, 5.3. In some cases, the performance of Clustering-based approaches is just comparable with other negative sampling approaches and based on the more in depth observation it comes out as it is mainly due to the structural dominance attribute of the underlying KGs. In those cases it is observed that the textual information tends to deviate from underlying symbolic structural information.

The third question is solely focused on optimizing the existing negative sampling approach ADNS and has been stated as:

**Research Question 3:** Does the performance of negative sampling approach could be enhanced by leveraging the optimized method for Computations?

To investigate this question, the revised strategy by employing the optimized method of computation is implemented using ADNS and this new proposed approach is named as OADNS. The strategy of this approach is discussed briefly in a separate section 4.7. By using the optimized matrix python operator and using a batch processing strategy, it is clearly evident through evaluations that it is possible to scale the approach to bigger benchmark datasets i.e. WN18, WN18RR, FB15k-237 and CoDEx datasets.

After the implementation, the experiments are performed in different settings that are briefly discussed in section 5.3, then a comparison to the existing state of the art approaches is made to determine the efficacy of this approach. From the evaluations, it comes out that OADNS performs really better on small datasets (Nations, UMLS, Kinship, and CoDEx-S) compared to larger datasets. On all
Conclusion and Future Work

bigger datasets except FB15k-237 which includes: WN18, WN18RR, CoDEx-M, and CoDEx-L the approach slightly underperforms with all the different scoring functions tested in the experiments. However, in the case of FB15k-237, the results are quite better, and with TransE and RotatE this method outmatch all other approaches.

6.2 Future Work

In the future, there are various paths still available that could be explored. In this work, the language models are employed with the clustering method to fetch the admissible entities for the negative sampling. However, there exists another possible track where for example language models along with the OADNS method could be utilized to create negative samples from the pool of more relevant entities through the aid of language models. Also, using language models will add more benefits as it will also mitigate the issue of OADNS which is taking more time in convergence. This is because, in the beginning, the fetched candidates are less similar to the ones retrieved later since embeddings are not trained enough in the initial epochs. By employing language models semantically more similar candidates could be retrieved in the early part of the training process and thus allowing OADNS to converge faster. Another possible future direction that can be explored is of using various distance metrics in finding the nearest clustering neighbors for potential negative candidates searching. In this thesis, the euclidean distance is used for finding $K$ hops or nearest clusters from where the entity itself is located. In adjacent to this, one possible direction would be trying the other language models besides sentence-Bert and FastText. Knowledge encoded in the language models has a great impact on finding good candidates since they utilized the pretrained knowledge and therefore they have much larger impact on the overall performance of the model. So trying out other language models which can leverage the entities textual information in a better way and could generate more meaningful candidates would certainly boost the overall performance of negative sample approaches.
Bibliography


# List of Figures

2.1 Knowledge graph example. Entities are represented as nodes and relations are represented as edges between nodes ........................................ 6

3.1 The architecture diagram of Dynamic Vector-Based Negative Sampling approach ................................................................. 19

3.2 The overall process flow of Clustering-based Negative Sampling generation methods. The process flow is divided into two main phases i) Pretraining Phase ii) Optimization Phase ........................................ 20

3.3 The Elbow method predicting the optimal number of clusters for UML dataset ................................................................. 24

4.1 Folder Structure of **Murphy** System .................................................. 34

4.2 The methods to generate the FastText and Transformer based embeddings ................................................................. 35

4.3 A glimpse of train data file from WN18 dataset. The triplets \((h, r, t)\) can be seen separated by tab .................................................. 39

4.4 The loading of entities and relations in the Main method ........................................ 40

4.5 The Method for Dimensionality Reduction and Clustering ........................................ 41

4.6 Initialization of KGE Model and Embeddings ........................................ 42

4.7 The Negative Triplet Generation Process in LMC Approach ........................................ 44

4.8 The negative triplet generation process using the Uniform Trained Embeddings (PMc) method ........................................ 46

4.9 The Negative Triplet Formation in OADNS Method ........................................ 47

4.10 The method to filter the generated negative samples ........................................ 48

4.11 The iterative training process: Score and Loss calculation ........................................ 49

5.1 Sentence-BERT transformer-based model embedding visualization ........................................ 59

5.2 The clustering results of trained Embeddings employing Uniform and LMC based negative sampling technique on the UMLS dataset ........................................ 61

1 The clusters visualization of Uniform negative sampling embeddings on CoDEx-M dataset ........................................ 76

2 The clusters visualization of S-BERT transformer embeddings on CoDEx-M dataset ........................................ 76
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>List of the attributes and their description that can be used as arguments for initiating the training process.</td>
</tr>
<tr>
<td>5.1</td>
<td>The datasets statistics i.e. the number of entities and relations in each dataset, as well as the sizes of training, validation, and test sets in terms of no of triples.</td>
</tr>
<tr>
<td>5.2</td>
<td>Results of KGE models with adversarial sampling. Results for models marked by $\ast$ are taken from [6].</td>
</tr>
<tr>
<td>5.3</td>
<td>Results of KGE models without adversarial sampling. Results for models marked by $\ast$ are taken from [6].</td>
</tr>
<tr>
<td>5.4</td>
<td>Results of different negative sampling techniques evaluated on Nation and UMLS datasets. <strong>Bold</strong> indicates our approaches beat others methods and <strong>bold</strong> shows that our approach beat all others except Uniform RW-SANS.</td>
</tr>
<tr>
<td>5.5</td>
<td>Results of different negative sampling techniques evaluated on CoDEx-S, CoDEx-M and CoDEx-L datasets. <strong>Bold</strong> indicates our approaches beat others methods and <strong>bold</strong> shows that our approach beat all others except Uniform RW-SANS.</td>
</tr>
<tr>
<td>5.6</td>
<td>Example set of candidate entities to form a negative triplet given a entity by Uniform, SANS and LMC approaches.</td>
</tr>
<tr>
<td>5.7</td>
<td>Example set of candidate entities to form a negative triplet given a entity by Uniform, SANS, PMC and OADNS approaches.</td>
</tr>
</tbody>
</table>
Appendix 1

Figure 1: The clusters visualization of Uniform negative sampling embeddings on CoDEx-M dataset.

Figure 2: The clusters visualization of S-BERT transformer embeddings on CoDEx-M dataset.