Approximate Query Answering using Answer Space Discretization

Max Zwager

Artificial Intelligence, Vrije Universiteit Amsterdam, The Netherlands
m.j.zwager@vu.nl

Abstract. Performing approximate query answering on incomplete Knowledge Graphs (KGs) is a challenging yet important task. Current models use entity embeddings and perform parameterized operations on those embeddings to generate a query representation. However, these methods do not actually perform query answering but rather return a ranking, even when a query does not have any answers. In this paper we present Query Answer Space Embeddings (QASE) that embeds entities as vectors and queries as polyhedral cones and is specifically designed for direct query answering where only a set of entities is returned. Inspired by random projection, we generate multiple cones to allow for a trade-off between precision and recall and be able to classify multiple disjoint clusters of entities simultaneously. We experiment with different model architectures and loss functions and compare their performance to four other baseline models, where we are the first to evaluate them in a binary classification setting. Our results show that QASE is currently outperformed by the baseline models, but that using pretrained entity embeddings can greatly improve performance. Insights show that training well-distributed entity embeddings and generating smaller answer spaces are still challenges to overcome.

Keywords: Approximate query answering · knowledge graphs · machine learning.

Table of Contents

1 Introduction

Knowledge graphs (KGs) are data representations that capture the relations between entities. In a KG, entities are expressed as nodes and their relations as edges. Unlike tabular data, the structure of KGs is not defined beforehand which makes them very flexible and subsequently well-suited for complex and irregular data.

The use of graphs as data structures has received increasing attention in both the public and enterprise domain [17]. Well-known instances of large publicly available graphs include DBpedia [18], YAGO [32] and Freebase [8]. While these graphs contain huge amounts of data, the quality of this data is very dependent on the errors, bias, disagreements and sometimes vandalism of their human contributors [17]. As a result of this KGs are notoriously incomplete, and finding methods for answering complex multi-hop queries on such large graphs efficiently and effectively is difficult and an active field of research.

Traditional approaches for multi-hop query answering involve the use of query languages such as SPARQL [31] or Cypher [14] to traverse and access specific nodes in the graph using simple logical inference. A common method for this is sub-graph matching. This method uses the intuition that conjunctive queries can be represented as Directed Acyclic Graphs (DAGs). This query graph is then matched against the KG in different orientations to search for target entities that satisfy the structure of the graph and its anchor nodes (known entities). A major drawback of this approach is that it cannot work with missing links that block inference between nodes, often resulting in *no answer* returned. While the issue of missing links can be solved by link prediction and graph completion methods, these methods transform the initial sparse graph in a fully-connected probabilistic graph which in turn exponentially increases computational complexity to a point where execution is no longer possible.

Recent alternatives can solve these issues by using graph embedding techniques that account for this missing information (approximate query answering). Here, a query is represented as a computation graph and gets embedded to a lower-dimensional latent space. With the use of anchor nodes and parameterized set operators that act in this latent space, an embedding for the target node is computed by traversing the computation graph. Subsequently a ranking of plausible answers is returned using nearest neighbour search. Further continuations of this work have resulted in the development of geometry-based models that embed queries and entities as geometric shapes (e.g. boxes and cones [27, 38]) and define set operations that act upon these shapes. This allows the use of more expressive logical operators such as unions, intersections, and for some methods negations.

Still, these methods also have their drawbacks. Firstly, they only support limited query structures (see section 3.1). Secondly, these methods lack generalizability and require a variety of query structures for training. Thirdly, current methods do not actually perform query answering, but rather return a ranking of most probable answers. Moreover, these models will always return a ranking containing all entities, even if the query does not have any answers. To return

a finite answer set, a cut-off point or threshold should be determined which requires additional resources.

To address these issues, we introduce **Q**uery **A**nswer **S**pace **E**mbeddings (QASE). We use a message passing approach to learn entity and relation embeddings. Inspired by random projection, we generate a set of polyhedral cones in the entity embedding space that enclose the answers to the query. Because the query embedding does not rely on restrictive operators such as projections and intersections, it allows for more complex and expressive query representations. This expressiveness is further increased by the fact that convex cones allow us to take unions, intersections and negations of these areas. Finally, since entities are still represented as single points in space, this demarcated answer space allows us to return a set of entities instead of a ranking.

We run multiple experiments to determine the performance of QASE, including model variations with different architectures and loss functions. As a baseline we train four other ranking-based query embedding models, and perform distance threshold optimization to allow for binary predictions. We also train QASE on pretrained entity embeddings to determine if this increases performance. During all experiments additional metrics were tracked that could identify flaws and challenges in the current design.

Our results show that QASE is outperformed by the baseline models. Additionally, no clear difference in model performance is observed between architectures and loss functions. Tracked metrics indicate that training well-distributed entity embeddings and generating smaller answer spaces are still challenges to overcome. The first challenge is confirmed by the results of using pretrained entity embeddings, where the performance of QASE approaches those of the baseline models.

This work is an Artificial Intelligence master thesis written under the supervision of Michael Cochez at the Vrije Universiteit Amsterdam.

2 Related work

Graph embeddings There is a significant amount of research on KG embeddings for link prediction, which practically equals to one-hop query answering. One of the first approaches to this is TransE [9]. Here the training objective is to learn entity and relation embeddings that approximate $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ in Euclidean latent space, where $\mathbf{h}$ is the subject entity, $\mathbf{r}$ is the relation and $\mathbf{t}$ is a object entity for which the triple (h, r, t) holds. TransE can model anti-symmetric, inverse and composite relations, but cannot capture symmetric and 1-to-n relations. TransR builds upon this method and uses relation-specific latent spaces to solve these two shortcomings, at the cost of not being able to capture composition relations [20]. Both methods above use L1 (Manhattan) or L2 (Euclidian) distances to calculate the score of each prediction, but other methods such as DistMult and Bilinear can also be used [36]. TransH is a method that models relations as hyperplanes with a translation operation upon it, and is specifically designed to handle one-

to-many, many-to-one, and many-to-many relations [35]. Other link prediction methods include RESCAL [24], HOLE [23] and NTN [30].

Subgraph matching Research on subgraph matching in the context of approximate query answering is limited. Zhang et al. propose a model that embeds query graphs and scores their similarity such that approximate answers are returned [37]. Other research focuses on embedding RDF queries with the aim of query relaxation, e.g. suggesting another query when the current query returns an empty answer set [34].

Query embeddings Because TransE is simple and can handle composite relations it can be retrained for graph query embeddings (GQE) to answer simple path queries [16]. Using learned embeddings, a query graph can be traversed from the anchor nodes across the relations to get a representation of the target embedding. However, because the vector representation of GQE cannot naturally represent sets of entities in latent space there is a significant focus on using geometrical representations that can. As discussed, two prominent examples are Query2Box (in this paper sometimes referred to as Q2B) and ConE that model entities/queries as boxes and cones respectively [27, 38]. These models use two vectors, one that determines the center and one that determines the shape of the geometrical representation. For entities, the representation is often reduced to a single (center) vector that does not have any volume. Other representations include BetaE that models entities and queries as probabilistic beta distributions [28], and Query2Particles that uses an ensemble of models to focus on different disjoint regions in the entity embedding space simultaneously [4].

As discussed we follow a line of research that uses message passing for embedding whole query graphs, such as MPQE [12]. Another related model that uses message passing is StarQE that is designed to handle hyperrelational KGs where relations have additional attributes [2]. Finally there is MPQB that uses a message passing approach to represent both entities and queries as boxes, and considers entities an answer to a query if the query and entity boxes overlap. To our knowledge, this is the only other research discussing direct query answering using query embeddings.

Other methods for query embedding involve using RNNs [11], contextual graph attention layers [21], compositionalization [15] and Complex Query Decomposition [3].

3 Background

In this section we will cover some preliminary concepts, as well as give a short introduction to locality sensitive hashing and random projection, explain how random projection has inspired the idea of answer space discretization, and give a description of the problem that we aim to solve.

3.1 Preliminaries

Knowledge graphs We follow earlier work by defining a knowledge graph as a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{R}, \mathcal{T})$, where the set $\mathcal{V}$ represents the entities, $\mathcal{E}$ contains typed relations between two entities in the form $r(v_i, v_j)$ where $v_i, v_j \in \mathcal{V}$ and relation type $r \in \mathcal{R}$, and function $\tau : \mathcal{V} \to \mathcal{T}$ maps the entity to an entity type [12]. There is a factual relation r going from v_i to v_j if and only if $r(v_i, v_j) \in \mathcal{E}$.

Queries In this paper we will focus on first-order logic queries in conjunctive normal form (CNF). These queries consist of predicates and logical operations such as conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$) and existential quantifiers ($\exists$) and are structured as a conjunction of clauses. These clauses contain one - or a disjunction of more - binary predicates that take a subject entity, relation type and object entity as arguments. For example, consider a KG containing movie data and the query "*Select all actors that played in a Christopher Nolan movie, and were born in Ireland*". We can rewrite this query to CNF resulting in the logical equation:

$$?Actor.\exists ?Movie : \text{directed}(\text{Nolan}, ?Movie) \wedge \text{has\_cast}(?Movie, ?Actor) \\ \wedge \text{land\_of\_birth}(\text{Ireland}, ?Actor), \tag{1}$$

where *?Actor* represents an entity or set of entities that is the answer to the query, and *?Movie* is an unknown variable. Queries written in CNF can be represented as computation graphs (figure 1), that can be used as input for graph neural networks, among other models, for a variety of tasks. The use of CNF for queries does come with constraints: query graphs can only contain one target, can only contain anchor nodes at the leaves of the DAG (no variables or targets) and cannot contain any cycles (e.g. self-loops). Whereas our proposed model is not restricted to handling these query structures exclusively, we use them in this paper to allow comparison to existing models that are.

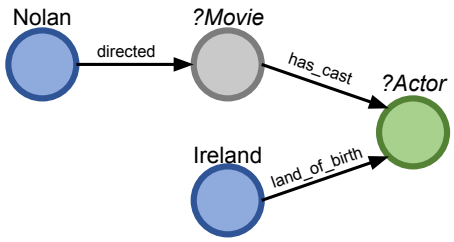


Fig. 1: Query computation graph corresponding to logical equation 1.

3.2 LSH

Locality sensitive hashing (LSH) is a technique for approximate nearest neighbour search (NNS). Unlike exact approaches such as linear search or space partitioning, LSH allows for a trade-off between accuracy and efficiency, making NNS significantly faster. The aim of LSH is to maximize the amount of hashing collisions of *similar* objects, which is in contrast with regular hashing techniques [19]. As a consequence, similar objects will often end up in the same hash bucket, which allows for performing NNS in sub-linear time. To prevent false-negatives (e.g. similar items not ending up in the same bucket) the hashing protocol is repeated using different hash functions. Taking into account the probabilistic properties of this approach, a LSH scheme can (for the method described below) be defined as a probability distribution over a family $\mathcal{F}$ of hash functions such that $\mathbf{Pr}_{h\in\mathcal{F}}[h(x) = h(y)] = sim(x, y)$, where $sim(x, y) \rightarrow [0, 1]$ is some similarity function defined on the collection of objects [10].

There are many methods to perform LSH, but the method relevant to this paper is *random projection*. In this method, given a vector representation of a large set of objects, random hyperplanes are generated that divide the object space into segments (figure 2). For each object and hyperplane a binary value indicates at which side of the hyperplane the object is located. The collection of binary indicators for this object is called its *signature*. Similar objects often contain the same signatures which can be used for efficient NNS, for instance using Hamming distance; the amount of bits two binary vectors differ. Using an anchor object and a distance threshold similar items to this anchor object can be returned. To prevent false-negatives (i.e. not returning an object while the object is similar), in practice the signature of an object is often subdivided into multiple bands. For every band the distance is calculated and the document is considered similar if the threshold is reached in at least one of the bands.

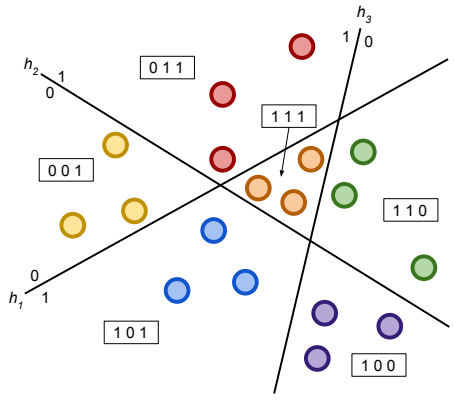


Fig. 2: Locality sensitive hashing using random projection. Similar items often contain similar signatures, which can be used for nearest neighbour search.

3.3 Query answer space discretization

Current models for embedding queries return a latent representation of the query target and subsequently use NNS techniques to return a ranking of best possible answers. Inspired by random projection, we aim to learn a set of hyperplanes that demarcate a convex subspace in the embedding space to perform direct query answering. To do this, we represent the subspace as a polyhedral cone, which can be defined by a finite number of hyperplanes (or half-spaces) that pass through the origin. If we represent these hyperplanes using normal vectors, we can subsequently represent a cone as a matrix $\mathcal{C}^{h\times d} = \{n_1, n_2, ..., n_k \mid n_i \in \mathbb{R}^d\}$, where h is the number of normal vectors and d is the dimension of each normal vector.

Whereas we model the queries as cones, we represent entities as vectors. This allows the generated cones to enclose sets of entities that are the answers to the query. For entities to be inside the cone, they must be on the correct side of each hyperplane, where we differentiate between the positive and negative side of a hyperplane by taking the dot product between its normal vector and the entity embedding. The entity is on the positive side of the hyperplane if and only if the dot product between the entity representation and the corresponding normal vector is positive. Intuitively, for the example in figure 2, we try to learn the set of hyperplanes for which the subspace corresponding to signature [1, 1, 1] contains the answers to the query.

Furthermore, we do not model one, but multiple cones to enclose the query answers. The motivation behind this is the observation that similar entities might not necessarily be close together in the embedding space. To illustrate this, consider the entities *Joe Biden* and *Emmanuel Macron*. Their representations should be similar in the way that they are both presidents, but different since they come from different countries. The result of this is that similar entities group together in disjoint clusters instead of a single region in latent space, which is demonstrated in figure 3. By using multiple cones, each cone can focus on its own cluster of entities.

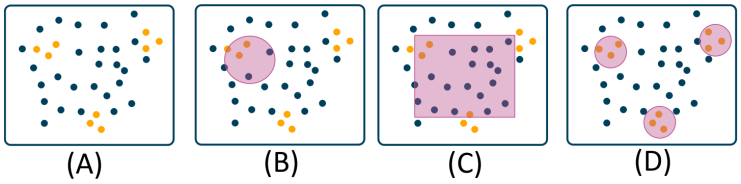


Fig. 3: In this example embedding space, the yellow dots are the answer entities, and the blue dots are the non-answer entities. The purple areas in (B), (C), and (D) demonstrate the neighborhoods of the vector embedding, the box embedding, and the desired query embedding (taken from [4]).

To do this, we generate for each query an answer space $\mathcal{S} \in \mathbb{R}^{c \times h \times d}$, consisting of c cones, h hyperplanes per cone with dimension d. An entity is contained in a cone if the dot product of its embedding and all the normal vectors in this cone are larger than zero. Subsequently, we consider the entity inside the answer space if it is contained in any cone. In other words, we take the union of the sets of entities contained in each cone as our answer set. Notice how this logical structure shows similarities with disjunctive normal form logic, which allows to have a trade-off between preventing false-positives (recall) and false-negatives (precision), similarly to how bands are used in LSH. Mathematically this logic can be represented by equation 2. Here entity v_k is considered an answer if the dot product between all hyperplanes in the cone are positive (such that their product is higher than zero), and this is true for at least one cone (such that the sum of these products is higher than zero).

$$\mathbf{v}_k \in [\![\hat{q}]\!] = \begin{cases} 1 & \text{if } (\sum_{i=0}^{c} \prod_{j=0}^{h} \max(0,\, \mathbf{v}_k \cdot \mathbf{s}_j^i)) > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{2}$$

There are three main distinctions with earlier work on approximate query answering. To start, we are the first to consider modelling queries and entities in different representation spaces. Secondly, we are the first to use a LSH-inspired approach by using multiple cones, simulating bands and allowing a trade-off between recall and precision. Thirdly, we are only the second paper to discuss direct query answering using query embeddings, where the only other explored method is MPQB.

3.4 Problem Statement

For any query q we call the set of entities present in the answer space the *predicted answer set* $[\![\hat{q}]\!]$ [27]. The goal is to find $[\![\hat{q}]\!] \subseteq \mathcal{V}$ such that for all $v_k \in \mathcal{V}$, $v_k \in [\![\hat{q}]\!]$ if and only if v_k is a true answer of that query q. For simplicity, we refer to the true set of answers for query q as *true answer set* $[\![q]\!]$. As we deal with incomplete knowledge graph $\mathcal{G}' \subset \mathcal{G}$ (where $\mathcal{G}$ is the complete KG), we cannot solve this problem by logical inference or sub-graph matching. Instead, we learn entity representations as vectors in some latent space, and generate a convex subspace that encloses entities such that $[\![\hat{q}]\!] \approx [\![q]\!]$.

4 Model

In this section we will describe our model, message passing operations, model architectures and loss functions.

4.1 Message passing

We define a model that learns representations for all entities, including unknown variables and targets, $\mathbf{E} \in \mathbb{R}^{(|\mathcal{V}|+2) \times d}$. The model also learns representations for

regular and inverse relations, and self-loops $\mathbf{R} \in \mathbb{R}^{(2|\mathcal{R}|+1)\times d}$. To acquire our answer space $\mathcal{S} \in \mathbb{R}^{c\times h\times d}$ we need ch normal vectors. These vectors are generated using message passing, where the node representations of the query computation graph are iteratively updated by aggregating neighbouring representations. We use a composition-based multi-relational graph convolution network (CompGCN) as our message passing approach [33]. Here the update equation is defined as:

$$\mathbf{x}_v^{k+1} = f\left(\sum_{(u,r)\in\mathcal{N}_{(v)}} \mathbf{W}_{\lambda(r)}\phi(\mathbf{x}_u^k, \mathbf{z}_r^k)\right),$$

where $\mathbf{x}_u^k, \mathbf{z}_r^k$ represent the initial entity and relation representations respectively at the $k+1$'th message passing step. Unlike earlier methods (e.g. R-GCN [29] or D-GCN [22]) that use relation-specific weight matrices, a composition function $\phi : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is used to make the update relation-aware, which makes CompGCN more parameter-efficient. The updated representations are finally passed through some activation function $f(\cdot)$. $\mathbf{W}_{\lambda(r)}$ is a learnable direction-specific weight matrix for outgoing, incoming and self-loops for which its assignment is defined below:

$$\mathbf{W}_{dir(r)} = \left\{\begin{array}{ll} \mathbf{W}_O, & r \in \mathcal{R} \\ \mathbf{W}_I, & r \in \mathcal{R}_{inv} \\ \mathbf{W}_S, & r = \top(\text{self-loop}) \end{array}\right\}$$

Finally the relation representation is updated with:

$$\mathbf{z}_r^{k+1} = \mathbf{W}_{rel}\mathbf{z}_r^k,$$

where $\mathbf{W}_{rel}$ is a learnable weight matrix.

Within the convolution layer we experiment with three different composition operators: multiplication, circular correlation and complex rotation. Additionally we experiment with regular symmetric message weighting and attention message weighting with 8 attention heads. Within all layers we use batch normalization.

On a higher level, different aggregation functions are tried such as the *max* and *sum* of all query nodes. Furthermore we experiment with the Target Message (TM) function introduced by Daza & Cochez (2020) [12]. Here a number of message passing steps are applied equal to the diameter to the query graph, which ensures that all information from all nodes can reach the target node. Subsequently, the representation of the target node is returned.

4.2 Model architectures

As discussed, message passing can be used to generate an output vector given a query structure and embeddings for the anchor, variable and target nodes. To generate our answer space $\mathcal{S} \in \mathbb{R}^{c\times h\times d}$ where we require more than one output vector, we use a collection of *sub-models* (GNNs) each consisting of 3

message passing layers. Because we are not restricted to using equal input and output dimension for these layers, we can also create larger output vectors which can represent more than one normal vector and reduce the number of learnable parameters in our model. The only restraint here is that the normal vectors require a dimension equal to the entity embedding, and subsequently the larger output vector should be divisible by this dimension.

Given this flexibility, we experiment with different model architectures containing one or more of these sub-models; **hype-wise**, **cone-wise** and **single** (figure 4). For the *hype-wise* model a GNN is used for every normal vector, resulting in a total of ch GNN models with output vector dimension d. For the *cone-wise* method a GNN is used for every cone, resulting in c GNN models with output dimension hd. Finally we have the *single* method where one GNN is used with output dimension chd. For the latter two architectures a linearly increasing output dimension is used for every message passing layer. For all model architectures the output is reshaped into multiple sets of hyperplanes (cones) after the forward pass, resulting in equal output dimensions for all models. Since the *hype-wise* architecture contains the same input and output dimensions for each layer, the sharing of weights between the convolution layers of a single GNN is considered. We differentiate between the model architectures by referring to QASE-h, QASE-c and QASE-s for the *hype-wise*, *cone-wise* and *single* version respectively.

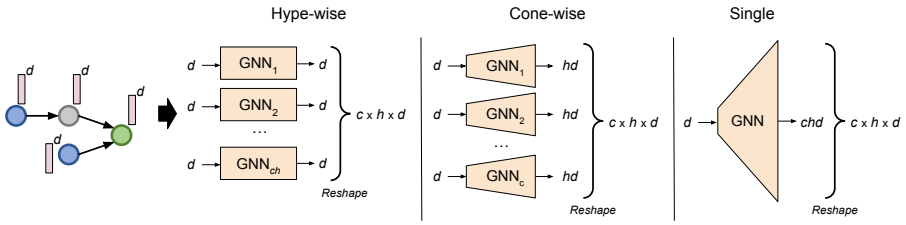


Fig. 4: Given input query with entity embedding dimension d, we experiment with three model architectures – **hype-wise**: $c \times h$ GNN models with output dimension d; **cone-wise**: c models with output dimension $h \times d$; **single**: 1 model with output dimension $c \times h \times d$.

4.3 Loss function

Given a cone and a finite set of positive and negative samples, we want all the positive samples to be inside the cone and all negative samples to be outside of the cone. In other words, for all normal vectors we want to maximize the cosine distance for positive samples and minimize the cosine distance of negative samples. Moreover, there is no need to significantly focus on decreasing the cosine distance for negative samples that are already outside of the cone, since this

will not influence the resulting prediction. Similarly we care less about positive samples already inside of the cone.

With this intuition, the parameters of the model are optimized by performing gradient descent on the contrastive loss function in equation 3. Here for each normal vector we minimize on the negated cosine distance for positive samples and the regular cosine distance for negative samples. Additionally, a ReLU or LeakyReLU activation function is applied to (partially) ignore cosine distances that are already in a favorable domain:

$$\mathcal{L} = \sum_{i=0}^{c} \sum_{j=0}^{h} \frac{1}{ch} \left(\sum_{a \in A_i} \frac{1}{|A_i|} f(-\mathrm{D}_{\cos}(\mathbf{v}_a^+, \mathbf{s}_j^i)) + \sum_{p=0}^{l} \frac{1}{l} f(\mathrm{D}_{\cos}(\mathbf{v}_p^-, \mathbf{s}_j^i)) \right), \tag{3}$$

where $\mathbf{s}_j^i$ is normal vector j of cone i, $\mathbf{v}^+$ and $\mathbf{v}^-$ denote the embedding of a positive and negative sample respectively, l is the negative sample size, f is an activation function in $\{\text{ReLU, LeakyReLU}\}$ and $\mathrm{D}_{\cos} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^1$ defines the cosine distance function:

$$\mathrm{D}_{\cos}(\mathbf{t}, \mathbf{n}) = \frac{\mathbf{t}^\top \mathbf{n}}{\|\mathbf{t}\| \, \|\mathbf{n}\|}$$

We sample negative entities from $v_k \in \mathcal{V} \setminus [\![q]\!]$. However, since we train our model with $\mathcal{G}_{\text{train}} \subset \mathcal{G}_{\text{valid}}$ where edges are missing, the assumed true answer set $[\![q]\!]$ may just be a subset of the ground truth answer set $[\![q]\!]'$. We assume that the number of ground truth answers $[\![q]\!]$ is significantly smaller than $\mathcal{V}$, which means that given $[\![q]\!]' \subseteq [\![q]\!]$ there is only a very small chance that complementary entities are incorrectly labeled as negatives (e.g. the probability of sampling $\mathbf{v}_p^-$ from $\mathcal{V} \setminus [\![q]\!]'$ where $\mathbf{v}_p^- \in [\![q]\!]$ is small). Therefore, we see this as a minor issue with negligible effects, and will not take any action to resolve it. Later obtained statistics on the number of query answers in appendix B support this assumption.

To make use of the flexibility multiple cones provide us with, we only sum over a subset of target vectors for each cone A_i, which maps every target entity to the *closest* cone:

$$A_i = \{a \in [\![q]\!] \mid \forall j \in \{0..c-1\} \setminus \{i\}, \mathrm{D}_{\cos}(\mathbf{v}_a^+, \hat{\mathbf{s}}^i) > \mathrm{D}_{\cos}(\mathbf{v}_a^+, \hat{\mathbf{s}}^j)\}$$

where $\hat{\mathbf{s}}^k$ denotes the average normal vector of cone k. Note that all entities are assigned to a single cone, such that the union of all cone assignments contain all target entities: $\cup_{i=0}^{c} A_i = [\![q]\!]$, and that the assignment sets of two cones never contain the same entities: $\forall n \in [0..c-1] \setminus \{i\}, A_i \cap A_n = \emptyset$. We can also see that in some cases there are cones that without assigned entities. For instance, this always occurs when a query has less answers than the number of cones. In this case where $A_t = \emptyset$ for cone t, there will be no summation over any target vectors and only the negative samples contribute to the loss of cone i.

Looking at equation 3 we see that the loss value for all normal normal vectors in a cone is computed similarly. As a result of this, there exists only one global optimum that is shared by all normal vectors within a cone. This might cause them to converge to the same direction, resulting in answer spaces with high volume. To tackle this, an alternative loss function is implemented.

4.3.1 Alternative loss function As explained, the loss value for normal vectors in computed similarly. However, this similarity in computation is not necessary since we only need the negative samples to be on the negative side of one hyperplane instead of all of them. By focusing on the most promising normal vector to exclude a negative sample (where $\mathrm{D}_{\cos}$ is already low) we break this similarity in loss calculation. We implement an alternative loss function as follows:

$$\mathcal{L} = \sum_{i=0}^{c} \frac{1}{c} \left(\sum_{a \in A_i} \sum_{j=0}^{h} \frac{1}{|A_i|h} f(-\mathrm{D}_{\cos}(\mathbf{v}_a^+, \mathbf{s}_j^i)) + \sum_{p=0}^{l} \sum_{j=0}^{h} f(\mathrm{D}_{\cos}(\mathbf{v}_p^-, \mathbf{s}_j^i) \cdot \mathbf{w}^j) \right),$$

where $\mathbf{w}_j$ is a weight used for computing the Softmin weighted average over all cosine distances between a negative sample and all normal vectors in a cone, which is calculated by:

$$\mathbf{w}_j = \frac{e^{-\mathrm{D}_{\cos}(\mathbf{v}^-, \mathbf{s}_j)}}{\sum_{k=0}^{h} e^{-\mathrm{D}_{\cos}(\mathbf{v}^-, \mathbf{s}_k)}}$$

Using this alternative loss function we hope to generate smaller answer spaces that might give better performance. We differentiate between the two loss functions by referring to the *normal* (n) and *softmin* (s) versions. Supplementary experiments with another loss function are also conducted, which are discussed in appendix G.

5 Experimental set-up

In this section, we discuss the design of various experiments that tackle the following research questions: **RQ1)** How well can we answer knowledge graph queries using hyperplanes? **RQ2)** How does the performance of the three model architectures compare? **RQ3)** How does the performance of the two loss functions compare? **RQ4)** Can we benefit from using pretrained entity embeddings?

5.1 Datasets

We use two publicly available benchmark datasets that have been used in previous related studies: AIFB and MUTAG [12, 29]. The AIFB dataset contains information regarding the research group of the same name and models key entities relevant for typical research communities and the relations between them [7].

The MUTAG dataset is a protein classification dataset where the chemical structure is encoded into the nodes and their relations [13]. Their statistics can be found in appendix A. Whereas related research includes additional datasets such as AM and BIO, we restrained from using these datasets as available resources were not sufficient enough to process them.

5.2 Query generation

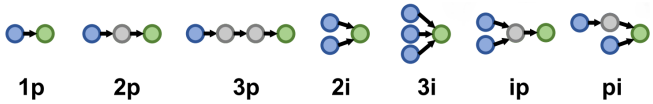


Fig. 5: Query structures (adapted from [27]).

Common query structures demonstrated in figure 5 are considered. Queries are generated with a graph query preprocessing and loading framework[1]. The frameworks allows loading graph datasets from multiple file formats, such as RDF or TSV files, and provides multiple options for data splitting, storage, the creation of mappings and the sampling of queries. It also provides options to filter out high in and out-degree nodes to downsample frequently occurring answers, a data issue that has been ignored in many related work in the past [2]. The framework is flexible as the sampled queries can be exported to multiple formats that can be used as input for existing baseline repositories, and also comes with a Pytorch compatible dataloader.

For our experiment each dataset is split in a train, validation and test graph, where $\mathcal{G}_{\text{train}} \subset \mathcal{G}_{\text{valid}} \subset \mathcal{G}_{\text{test}}$. Splitting is performed using a round-robin method where edges of the complete graph are included in the train, validation and test graphs with probabilities 0.7, 0.8 and 1.0 respectively. Nodes in both datasets are restricted to having a maximum in and out-degree of 50. The resulting query statistics can be found in appendix B.

5.3 Approach

The GNNs in our architecture are implemented using StarQE models containing CompGCN layers, but we use their implementation without hyper-relational qualifiers [12]. This also allows for experimentation on hyper-relational graphs in future work. All model and training implementation is written in Pytorch [26].

To find appropriate hyperparameters we implement a hyperparameter optimization (HPO) pipeline using the Optuna optimization framework [1]. We train and evaluate on 1000 and 500 queries per structure for each respective split. We sampled configurations using a Tree-structured Parzen Estimator algorithm

[1] https://github.com/miselico/graph_query_sampler

and applied early stopping to runs with weighted f1-scores on the validation queries that were below the median for that epoch [6]. We run the optimization pipeline for each dataset and model architecture (six in total) with the normal loss function. The explored hyperparameters can be found in appendix E.1. After optimization we train each combination (including the Softmin version of the loss function) on the best found hyperparameters for three epochs and compare the performance between them, as well as baseline models that we train until convergence.

To investigate the effect of using pretrained entity embeddings, we use trained embeddings from StarQE. Since our model is implemented using StarQE submodels, the embeddings can be reused directly. We first run HPO for the StarQE model for both datasets with embedding dimension 64 and 128. For this we use the authors original optimization setup [2]. After optimization we train StarQE using the best found hyperparameters until convergence. Using the pretrained entity embeddings from StarQE, we train the *cone-wise* model architecture with both loss functions and embedding dimensions 64 and 128 for four epochs. During training, the entity embeddings remain frozen. We chose the *cone-wise* architecture since it is the middle ground in terms of complexity and efficiency between all three architectures. As a baseline we optimize a distance threshold for StarQE (see section 5.4). All QASE experiments are run on a single A100 (40GB) or RTX A6000 (48GB)[2]. All baseline experiments are run on a single NVIDIA GeForce GTX 1080 Ti (11GB) orRTX A4000 (16GB). The code to reproduce the model experiments can be found online [3].

5.4 Baseline models

We compare our model versions to four other query embedding models, namely GQE [16], BetaE [28] and Query2Box [27], and a message passing-based model, StarQE [12]. Since for StarQE we only consider edges without qualifiers, the idea behind the model is similar to MPQE but uses a more parameter-efficient and better performing message passing algorithm. All models are first trained using their standard objective, after which a structure-wise distance threshold is optimized using using Bayesian optimization where we maximize the f1-score [25]. This optimization is done on all validation queries, after which the determined structure-wise thresholds are used on the test queries.

For a fair comparison each baseline model is given the same entity embedding dimension of 800. We follow earlier work by giving Query2Box and BetaE half of the embedding dimension for GQE and StarQE, since each entity representation consist of two parameters (Query2Box: center and offset; BetaE: α and β). An additional overview of the number of learnable parameters for each baseline model and QASE is provided in appendix C.

There will be no HPO performed for the baseline models. The motivation for this is that minimal gain is expected while being costly and harmful for the

[2] Hardware specifications are explicitly mentioned as QASE requires significantly more memory for training.

[3] https://github.com/maxzw/QASE

environment. We use the optimized model hyperparameters of their respective papers, which can be found in appendix D.

Because all query structures are used for training, we do not discuss the generalizability of the above models to unseen query structures. For this we refer to previous research [2–4, 27, 28, 38]. The code to reproduce the baseline results can be found online [4] [5].

5.5 Evaluation procedure

In all experiments all models are evaluated in a binary classification setting. Using evaluation formula 2 the predicted answer set $[\![\hat{q}]\!]$ is computed for each query. Subsequently, the f1-score is calculated using equation 4. Whereas other f-scores can be considered, we use the f1-score to put equal emphasis on recall and precision. We calculate performance on the *filtered* answer set, i.e. when evaluating validation/test queries, the focus is on $[\![q]\!]_{\text{valid}} \setminus [\![q]\!]_{\text{train}}$ and $[\![q]\!]_{\text{test}} \setminus [\![q]\!]_{\text{valid}}$ respectively. Metrics are aggregated into structure-wise averages, and a global macro and micro average.

Similar to earlier work we found greatly varying true answer set cardinalities [2]. For example, there are queries where the maximum answer set is 468, whereas the 75 percentile is only 1. These queries are corrected for by weighting the aforementioned metrics of each sample by the inverse cardinality of the corresponding answer set. A detailed analysis of the answers of the sampled queries can be found in appendix B.

$$\mathrm{F}_1 = \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

5.6 Additional metrics

To get more relevant insights on QASE, additional metrics were tracked every evaluation epoch during training:

Metric 1 (Embedding uniqueness) *The embedding uniqueness is the fraction of unique sign signatures, which is the result of transforming real-valued entity embedding vectors to a binary vector where positive values get assigned a 1 and negative values a 0. As embeddings get initiated randomly before training the embedding uniqueness start at 1.0 almost surely, given a large enough dimension size. We also track the embedding uniqueness for StarQE. Since Query2Box and BetaE are trained using Euclidean distances instead of angular distances, the entity embeddings might all lie in the same sign domain without any loss of performance, hence there is no need to track the embeddings for those models. We calculate the embedding uniqueness once every evaluation run.*

[4] https://github.com/maxzw/QASE_baselines_cqd
[5] https://github.com/maxzw/QASE_baselines_mphrqe

Metric 2 (Answer space size) *The answer space size is the volume of the answer space. This value is approximated with a Monte-Carlo approach, where a large number of random vectors are generated (n=10^4), and the fraction of vectors inside the answer space approximates the answer space size. Because this computation requires a lot of resources, we calculate the average answer space size on a sample of batches (n=10^3) of the evaluation data.*

Metric 3 (Answer set size) *The answer set size is the cardinality of the predicted answer set $[\![\hat{q}]\!]$. We calculate the average answer space size of all batches of the evaluation data.*

6 Results

Results of HPO for QASE can be found in appendix E.1 where table 9 shows both the optimal hyperparameters and the corresponding weighted f1-score on the validation queries. Training the *hype-wise* architecture on the full datasets required over 100 hours per epoch. Since current resources were not sufficient, it was not possible to train this architecture within an acceptable time limit. We included the architecture in this research nonetheless, since it shows better performance than the other architectures and it could be trained in future work. The *cone-wise* and *single* architectures were trained on both datasets with both loss functions, taking approximately four and eight hours per epoch respectively. The baseline models were also trained using their standard objective, after which their distance threshold was optimized structure-wise. The results of the distance threshold optimization for the baseline models can be found in appendix F. The final evaluation metrics for QASE architecture and loss function combinations, as well as the baseline models are shown in table 1. The additional metrics tracked for QASE are displayed in figure 6. Analysis showed that the embedding uniqueness of StarQE remained 100% for AIFB and decreased to 95.3% for AIFB. For clarity we excluded these values from figure 6b.

Results of HPO on StarQE can be found in appendix E.2, where table 10 shows the optimal hyperparameters. The final evaluation metrics for QASE using pretrained entity embeddings and the StarQE baseline are shown in table 2. The additional metrics tracked for QASE using pretrained embeddings are displayed in figure 7. The loss values for all runs of QASE both with and without pretrained embeddings are displayed in figure 8.

7 Discussion

7.1 QASE

The results in table 1 show that all baseline models outperform QASE by a large degree, indicating that QASE does not yet work very well. We also see QASE performs better on the AIFB dataset than on MUTAG, which is in line with the baseline models. Regarding the comparison between model architectures and loss functions, there seems to be a difference in performance between

Table 1: Classification performance (f1-score, %) of all baseline models and QASE architectures (c/s) and loss versions (n/s).

Method	1p	2p	3p	2i	3i	ip	pi	Macro	Micro
AIFB									
GQE	**44.51**	**32.01**	**34.43**	**59.27**	72.68	**43.44**	**55.68**	**48.86**	**58.93**
Q2B	41.83	31.06	15.63	59.09	**73.34**	42.23	54.43	45.37	58.25
BetaE	21.35	20.45	19.38	51.87	67.53	34.19	49.37	37.73	52.58
StarQE	17.49	17.32	14.61	51.51	67.59	32.23	47.22	35.42	51.06
QASE-c (n)	4.98	5.48	3.62	9.2	11.52	7.33	8.18	7.19	9.16
QASE-c (s)	5.11	5.67	4.6	9.34	11.38	7.5	8.42	7.43	9.27
QASE-s (n)	3.31	5.11	4.34	6.57	7.48	6.0	4.96	5.4	5.83
QASE-s (s)	3.29	4.8	3.99	6.35	8.05	5.68	4.85	5.29	5.91
MUTAG									
GQE	**8.48**	**32.8**	30.85	**36.63**	70.51	38.85	42.29	37.2	49.11
Q2B	7.74	32.18	**51.52**	31.96	69.11	38.97	42.17	**39.09**	48.31
BetaE	2.26	15.5	26.27	25.94	62.55	38.22	45.71	30.92	47.64
StarQE	0.45	13.37	39.09	18.46	**72.65**	38.75	**47.75**	32.93	**51.09**
QASE-c (n)	0.16	3.99	5.49	1.06	4.27	6.5	5.8	3.9	4.88
QASE-c (s)	0.17	4.61	6.33	0.93	3.73	7.6	6.42	4.26	5.1
QASE-s (n)	0.17	4.71	6.48	1.54	6.07	7.02	6.55	4.65	5.9
QASE-s (s)	0.17	4.11	3.38	0.91	3.71	5.29	5.41	3.29	4.45

model architectures trained on AIFB where the *cone-wise* architecture on average performs 157% better than the *single* architecture. This higher performance can be caused by the larger number of learnable parameters in the *cone-wise* architecture. However, no such difference is observed between architectures trained on MUTAG, which might indicate that this difference in performance is dependent on other factors. No clear difference in performance is observed between different loss functions.

Additional insights on the performance of QASE can be found in figure 6 that shows the f1-score and additional metrics. In subfigure 6b we can see that the embedding uniqueness for QASE decreased over time. This is in contrast with the observations for StarQE, where embedding uniqueness never decreased below 95%. Looking at the subfigures together we see that high-performing QASE models often have a high embedding uniqueness and a small average answer set size, which logically follows since most queries contain only a few answers. Also interesting is that the answer space size shows no clear indication or correlation with performance, and that some QASE models perform relatively well even if over 50% of the entity embedding space is designated as an answer. Whereas taking the union multiple cones might cause this large answer space, the majority of queries only contain one answer so the answer space should still be much lower.

There are a number of possible explanations for the poor performance of QASE. Firstly, it could be that the assignment function creates multiple disjoint optimization problems, since each cone focuses on its own assigned target entities and does not care about other target entities. Because of this, the only way

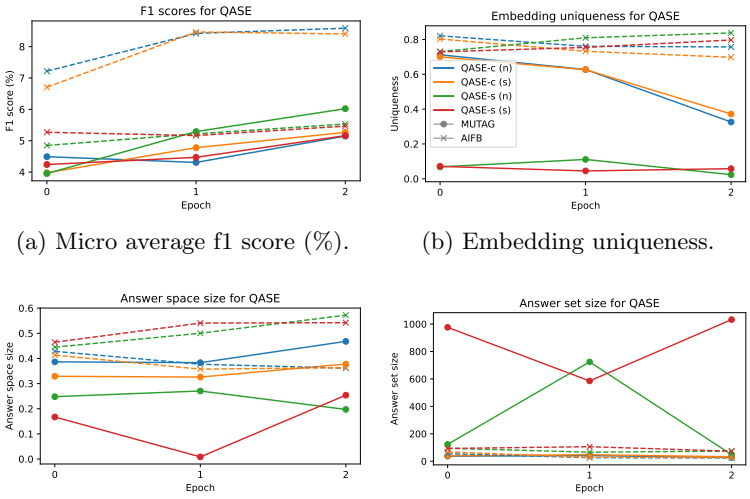
Table 2: Classification performance (f1-score, %) of training on pretrained entity embeddings on QASE-c with normal (n) and softmin (s) loss function, with StarQE as a baseline.

Dim	Method	1p	2p	3p	2i	3i	ip	pi	Macro	Micro
AIFB										
64	StarQE	15.79	**19.31**	**16.5**	**50.47**	**70.01**	**32.28**	**46.24**	**35.8**	**51.1**
	QASE-c (n)	**17.6**	9.19	7.39	44.18	62.88	12.52	30.53	27.78	40.72
	QASE-c (s)	16.53	8.38	5.96	43.28	60.57	11.15	33.91	27.29	41.83
128	StarQE	16.47	**17.09**	**15.72**	50.2	71.34	**32.59**	44.4	**35.4**	50.35
	QASE-c (n)	**18.21**	10.3	8.09	**51.35**	**72.4**	18.08	**46.23**	32.55	**52.67**
	QASE-c (s)	17.53	10.22	6.8	49.65	70.74	18.66	45.69	31.47	51.74
MUTAG										
64	StarQE DT	**0.89**	**13.97**	**32.05**	**21.5**	**58.08**	**42.34**	**46.52**	**30.76**	**46.47**
	QASE-c (n)	0.45	12.87	12.13	5.56	20.36	18.35	22.65	13.22	20.04
	QASE-c (s)	0.49	9.57	12.52	3.71	11.16	20.31	24.86	11.97	18.84
128	StarQE DT	**1.32**	**16.84**	**31.63**	**27.34**	**65.44**	**35.49**	**48.24**	**32.33**	**49.91**
	QASE-c (n)	0.39	11.65	3.02	7.09	43.35	24.49	38.62	19.05	36.36
	QASE-c (s)	0.4	15.06	6.2	9.72	56.41	27.33	44.97	23.73	44.23

similar entities that are assigned to different cones end up in the same region is if they get pushed there by negative samples. We hypothesize that the chances of this happening effectively is very small, which results in an entity distribution in the embedding space where similar entities are still far away from each other, and a large answer space is needed to include them all in our predictions.

Secondly, it could be that our loss function in practice still suffers from similarity in computation for different normal vectors within a cone. Because we use a ReLU or LeakyReLU activation function to (partially) ignore values already in a favorable domain, cosine distances for negative samples might not get much lower than zero. This can result in the weighing factor $\mathbf{w}^j$ being approximately the same for each normal factor, making the whole weighing component redundant. Looking at subfigures 6c and 7b we can see that there is no clear difference between the two loss functions on the answer space, supporting this theory.

Figure 8 shows that the loss has converged to negative values for all runs, indicating that on average the cosine distance for positive samples is higher than zero, and lower than zero for negative samples. However, this average is no guarantee for good performance, as positive samples need to be on the correct side of all hyperplanes and both sides of the contrastive loss can compensate for the other. With this in mind, it is probable that the loss functions used in this research do not accurately reflect the intended learning goal of the model. In fact, both of the aforementioned explanations for the poor performance of QASE are caused by the loss function. Experimentation with other loss functions therefore seems essential when building further upon this work.



(a) Micro average f1 score (%).

(b) Embedding uniqueness.

(c) Answer space size.

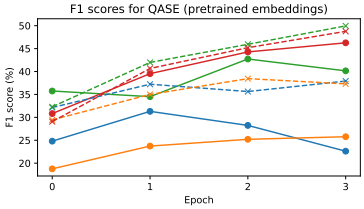
(d) Answer set size.

Fig. 6: Micro average f1 score and additional metrics per epoch, tracked for QASE models on the validation queries.

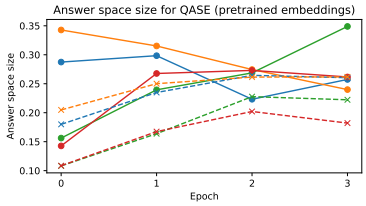
7.2 QASE using pretrained embeddings

The results in table 2 show that using pretrained entity embeddings results in better performance in all cases, one even outperforming the baseline model. Looking at the f1-score and additional metrics for QASE using pretrained embeddings in figure 7, we see that the answer space size and subsequently the answer set size has decreased for all models in comparison with figure 6. We hypothesize this is due to a better distribution of entity embeddings, eliminating the need for a large answer space to contain all query answers, which supports the first possible explanation for the poor performance of QASE. We also observe lower answer space and answer set sizes for well performing models, similarly to figure 6.

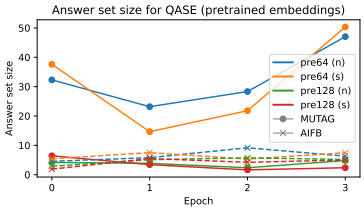
Comparing QASE without and with pretrained entity embeddings, we see that even though QASE does not yet work very well, using pretrained embeddings can greatly increases performance. Current challenges therefore include training these embeddings correctly, as well as decreasing the answer space size to allow for more precise predictions. The second could be achieved by increasing the number of hyperplanes, or using different loss functions where the normal vectors are actively diversified to make the answer space more round, of which an example implementation is discussed in the supplementary experiments in appendix G.



(a) Micro average f1 score (%).



(b) Answer space size.



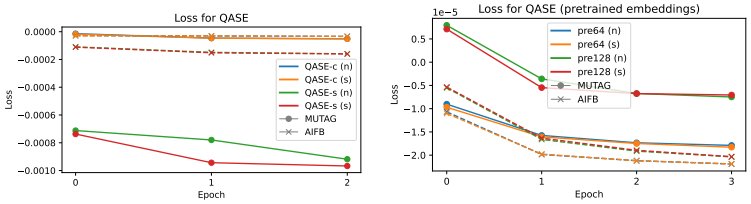
(c) Answer set size.

Fig. 7: Micro average f1 score and additional metrics per epoch, tracked for QASE models trained with pretrained entity embeddings on the validation queries.

7.3 Limitations and future work

In this section we discuss multiple limitations of this work, and possible future research directions.

7.3.1 Resources The first limitation of our work is the lack computational resources, which caused multiple other disadvantages. Firstly, other larger datasets could not be included in our analysis as resources were not enough to process these datasets. Secondly, whereas the *hype-wise* architecture performed best during HPO (see table 9 in appendix E.1), resources were not sufficient enough to train the architecture on the whole dataset within an acceptable time limit. This is also due to the large amount of parameters of the QASE architectures. As can be seen in table 7 in appendix C, the amount of parameters for QASE is much higher than for the baseline models. A third disadvantage is our restriction to low numbers for some hyperparameters during optimization such as batch size, number of cones, number of hyperplanes per cone and the embedding dimension, and the exclusion other variables. Given more resources higher values and more variables could have been included which might have led to better results. Finally, the lack of resources restricted us to only documenting single runs. Therefore we emphasize that even though the main experiments show good indications for the performance of QASE, they hold no statistical power. Future research might circumvent all these issues by having more resources available. Another way to prevent these issues in the future is by the development of more efficient methods for data processing, model training and hyperparameter optimization.



(a) Loss of QASE.

(b) Loss of QASE with pretrained embeddings.

Fig. 8: Running loss during training for both QASE without and with pretrained entity embeddings.

7.3.2 Queries The second limitation of our work is the imbalance in generated queries within the datasets. As discussed in appendix B there is a large imbalance where the two most frequently occurring structures *3i* and *pi* account for over 85% of all queries. Looking at the results of the baseline models in table 1, we also see that the performances of these query structures are much higher than for others. By better balancing the number of queries per structure, better results might be achieved in the future. During processing the datasets, we experimented with multiple values for in and out-degree filtering. No clear relation was found between the filter value and the resulting number of queries per structure, and finding an acceptable balance might just be the result of trial and error. Another way of balancing the number of queries is by using sampling or taking only a subset of queries for structures that are overrepresented.

7.3.3 Loss function In section 7.1 multiple flaws of the current loss functions are discussed. Future research can focus on these flaws to find alternative loss functions that might prove more effective. Main challenges regarding the loss function include diversifying the normal vectors to generate smaller cones, and acquiring better entity embeddings. The first challenge could be tackled by including a diversification component in the loss function, as discussed in the supplementary experiments in appendix G. The second challenge could be tackled by preventing the disjoint optimization problem that is also described in section 7.1. This could be cone by using a weighted assignment function instead of a discrete one, to make cones focus on their assigned entities while also taking into account the distribution of other entities assigned to other cones. This approach might help to move similar entities assigned to different cones closer together. However, since the cone takes into account target entities that might be in a different direction, this can also negatively influence the generated cones in direction or shape.

7.3.4 Other future work Aside from QASE and MPQB, we see other possible methods for direct query answering using query embeddings. Similar to MPQB that uses the box embedding introduced by Query2Box, one could also try to perform direct query answering by using the cones introduced by ConE. As entities are still represented as vectors (or cones with aperture 0), the resulting query cone representations can *contain* the entity representations and subsequently return this enclosed set.

Another interesting research direction is the exploration of other use cases that have a different emphasis on recall and precision. Whereas in this work we focus on their harmonic mean by using the f1-score, there might be use cases where recall and precision are not equally important. Ideally, there would be a hyperparameter in some loss function that could control the focus on these two metrics.

Finally, future work can investigate how QASE scales to hyper-relational graphs. As QASE is built using StarQE sub-models, hyper-relational queries can be handled naturally and require minimum changes.

8 Conclusion

In this work we have introduced our model QASE and demonstrated its performance on the AIFB and MUTAG datasets by comparing it to four baseline models. In the context of approximate query answering, we are the first to consider evaluating these baseline models in a binary classification setting. Our results show that all baseline models outperform QASE, indicating that QASE does not yet work very well. Moreover, we see no indication for a difference in performance between model architectures and loss functions. We have shown that using pretrained entity embeddings can be used to increase performance to the baseline level, which suggests that finding correct entity embeddings is a challenge still to overcome. By tracking additional metrics we identified flaws of the model such as a low embedding uniqueness and high answer space and set size. Future work could use these metrics to track and analyse future model performance.

Negative impacts of this work. This work could have negative societal impacts in fields where the correctness of query answers is of vital importance, such as in the medical domain. Because our model does not return a ranking but instead returns a finite answer set, false negatives can easily be overlooked. On the other hand, our model can have a positive impact in places where a lot of time is spent checking answers but where many invalid queries are submitted. The applicability of our model is therefore very dependent on the use case and requires careful consideration.

Acknowledgements

I would like to thank Michael Cochez for his supervision during the writing of this thesis. I also thank Dimitrios Alivanistos, Daniel Daza and Peter Bloem for their insightful contributions during various brainstorming sessions. Finally I thank the SURF and NWO/NCF (the Netherlands Organization for Scientific Research) and participating universities for the support in using the Lisa and DAS-6 (The Distributed ASCI Supercomputer 6) compute clusters [5].

References

1. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 2623–2631 (2019)
2. Alivanistos, D., Berrendorf, M., Cochez, M., Galkin, M.: Query Embedding on Hyper-Relational Knowledge Graphs. In: International Conference on Learning Representations (2021)
3. Arakelyan, E., Daza, D., Minervini, P., Cochez, M.: Complex Query Answering with Neural Link Predictors. In: International Conference on Learning Representations (2020)
4. Bai, J., Wang, Z., Zhang, H., Song, Y.: Query2Particles: Knowledge Graph Reasoning with Particle Embeddings. arXiv preprint arXiv:2204.12847 (2022)
5. Bal, H., Epema, D., de Laat, C., van Nieuwpoort, R., Romein, J., Seinstra, F., Snoek, C., Wijshoff, H.: A medium-scale distributed system for computer science research: Infrastructure for the long term. Computer **49**(5), 54–63 (2016)
6. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. Advances in neural information processing systems **24** (2011)
7. Bloehdorn, S., Sure, Y.: Kernel methods for mining instance data in ontologies. In: The Semantic Web, pp. 58–71. Springer (2007)
8. Bollacker, K., Tufts, P., Pierce, T., Cook, R.: A platform for scalable, collaborative, structured information integration. In: Intl. Workshop on Information Integration on the Web (IIWeb'07). pp. 22–27 (2007)
9. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. Advances in neural information processing systems **26** (2013)
10. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. pp. 380–388 (2002)
11. Das, R., Neelakantan, A., Belanger, D., McCallum, A.: Chains of Reasoning over Entities, Relations, and Text using Recurrent Neural Networks. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers. pp. 132–141 (2017)
12. Daza, D., Cochez, M.: Message passing query embedding. arXiv preprint arXiv:2002.02406 (2020)
13. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. Journal of medicinal chemistry **34**(2), 786–797 (1991)

14. Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., Taylor, A.: Cypher: An evolving query language for property graphs. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1433–1445 (2018)
15. Guu, K., Miller, J., Liang, P.: Traversing Knowledge Graphs in Vector Space. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 318–327 (2015)
16. Hamilton, W., Bajaj, P., Zitnik, M., Jurafsky, D., Leskovec, J.: Embedding logical queries on knowledge graphs. Advances in neural information processing systems **31** (2018)
17. Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G.d., Gutierrez, C., Kirrane, S., Gayo, J.E.L., Navigli, R., Neumaier, S., et al.: Knowledge graphs. Synthesis Lectures on Data, Semantics, and Knowledge **12**(2), 1–257 (2021)
18. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al.: Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic web **6**(2), 167–195 (2015)
19. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of massive data sets. Cambridge university press (2020)
20. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: Twenty-ninth AAAI conference on artificial intelligence (2015)
21. Mai, G., Janowicz, K., Yan, B., Zhu, R., Cai, L., Lao, N.: Contextual graph attention for answering logical queries over incomplete knowledge graphs. In: Proceedings of the 10th International Conference on Knowledge Capture. pp. 171–178 (2019)
22. Marcheggiani, D., Titov, I.: Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. pp. 1506–1515 (2017)
23. Nickel, M., Rosasco, L., Poggio, T.: Holographic embeddings of knowledge graphs. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 30 (2016)
24. Nickel, M., Tresp, V., Kriegel, H.P.: Factorizing yago: scalable machine learning for linked data. In: Proceedings of the 21st international conference on World Wide Web. pp. 271–280 (2012)
25. Nogueira, F.: Bayesian Optimization: Open source constrained global optimization tool for Python (2014–), https://github.com/fmfn/BayesianOptimization
26. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)
27. Ren, H., Hu, W., Leskovec, J.: Query2box: Reasoning Over Knowledge Graphs In Vector Space Using Box Embeddings. In: International Conference on Learning Representations (ICLR) (2020)
28. Ren, H., Leskovec, J.: Beta embeddings for multi-hop logical reasoning in knowledge graphs. Advances in Neural Information Processing Systems **33**, 19716–19726 (2020)
29. Schlichtkrull, M., Kipf, T.N., Bloem, P., Berg, R.v.d., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: European semantic web conference. pp. 593–607. Springer (2018)

30. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. Advances in neural information processing systems **26** (2013)
31. Steve Harris, A.S., Prud'hommeaux, E.: SPARQL 1.1 Query Language, W3C Recommendation 21 March 2013. W3C Recommendation. World Wide Web Consortium. (2013), https://www.w3.org/TR/2013/REC-sparql11-query-20130321/
32. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A large ontology from wikipedia and wordnet. Journal of Web Semantics **6**(3), 203–217 (2008)
33. Vashishth, S., Sanyal, S., Nitin, V., Talukdar, P.: Composition-based Multi-Relational Graph Convolutional Networks. In: International Conference on Learning Representations (2019)
34. Wang, M., Wang, R., Liu, J., Chen, Y., Zhang, L., Qi, G.: Towards empty answers in SPARQL: approximating querying with RDF embedding. In: International semantic web conference. pp. 513–529. Springer (2018)
35. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 28 (2014)
36. Yang, B., Yih, S.W.t., He, X., Gao, J., Deng, L.: Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In: Proceedings of the International Conference on Learning Representations (ICLR) 2015 (2015)
37. Zhang, L., Zhang, X., Feng, Z.: TrQuery: An embedding-based framework for recommanding sparql queries. arXiv preprint arXiv:1806.06205 (2018)
38. Zhang, Z., Wang, J., Chen, J., Ji, S., Wu, F.: ConE: Cone embeddings for multi-hop reasoning over knowledge graphs. Advances in Neural Information Processing Systems **34** (2021)

Appendix

A Dataset Statistics

Table 3 shows the number of entities, entity types, relations and relation types for both datasets. The number of entities for both AIFB and MUTAG may differ with current literature, as blank nodes have been assigned to new entity instances instead of being excluded from the graph.

Table 3: Dataset statistics for AIFB and MUTAG.

	AIFB	**MUTAG**
Entities	2,835	22,540
Entity types	6	4
Relations	39,436	81,332
Relation types	49	8

B Query Statistics

Table 4 shows the number of queries for each dataset, split and structure. Some queries structures are notably underrepresented, which is clearly shown in table 5 that contains the distribution of query structure in percentages. Also interesting is the difference between the amount of generated queries for each dataset, which is over 4 times larger for AIFB than it is for MUTAG, even though the latter dataset contains almost 10 times as much entities and twice as much relations. We hypothesize this is due to the difference in edge distribution, where the in and out-degree filtering protocol excludes nodes from the graph that should otherwise have generated a lot of queries.

Table 4: Number of queries for each query structure.

Dataset	**Split**	**1p**	**2p**	**3p**	**2i**	**3i**	**ip**	**pi**	**Total**
AIFB	Train	3,222	6,453	348	27,807	170,505	58,924	506,918	774,177
	Valid	756	3,404	164	8,537	74,321	40,542	246,654	374,378
	Test	1,338	5,502	327	21,266	243,241	87,700	555,803	915,177
MUTAG	Train	10,105	1,472	203	19,052	41,952	3,733	106,723	183,240
	Valid	2,654	656	66	6,373	19,252	1,739	32,135	62,875
	Test	4,508	1,059	114	14,432	53,252	4,979	103,713	182,057

Table 5: Percentage (%) of each query structure for all datasets and splits.

Dataset	Split	1p	2p	3p	2i	3i	ip	pi
AIFB	Train	0.42	0.83	0.04	3.59	22.02	7.61	65.48
	Valid	0.2	0.91	0.04	2.28	19.85	10.83	65.88
	Test	0.15	0.6	0.04	2.32	26.58	9.58	60.73
MUTAG	Train	5.51	0.8	0.11	10.4	22.89	2.04	58.24
	Valid	4.22	1.04	0.1	10.14	30.62	2.77	51.11
	Test	2.48	0.58	0.06	7.93	29.25	2.73	56.97

Table 6 shows the answer set statistics for each dataset and structure. To save space, we combined the results of all splits. We see that in general AIFB queries contain more answers than MUTAG queries. An explanation for this could be that MUTAG contains smaller graph networks (molecules), since it is mainly used for graph classification. In combination with the large amount of unique entities present this might cause the presence of very specific queries that contain less answers.

Table 6: Query answer statistics (splits combined for simplicity).

Structure	mean	std	min	25%	50%	75%	max
AIFB							
1p	3.33	13.67	1	1	1	2	683
2p	10.04	44.03	1	1	3	7	876
3p	2.50	2.67	1	1	1	2	27
2i	1.25	1.85	1	1	1	1	128
3i	1.03	0.31	1	1	1	1	41
ip	10.93	54.65	1	1	2	6	876
pi	2.11	6.00	1	1	1	1	468
MUTAG							
1p	3.26	6.27	1	1	1	2	175
2p	5.28	8.91	1	1	1	6	168
3p	1.17	0.46	1	1	1	1	6
2i	1.06	0.18	1	1	1	1	2
3i	1.00	0.00	1	1	1	1	1
ip	1.00	0.05	1	1	1	1	2
pi	1.06	0.97	1	1	1	1	135

C Parameter analysis

To compare the complexity of representations for entities, as well as relations, projections, intersections and convolutions, the sets of learnable parameters for QASE and the used baseline models are displayed in table 7 as $\mathbf{E}$, $\mathbf{R}$, $\mathbf{P}$, $\mathbf{I}$ and $\mathbf{C}$ respectively. Here $|\mathcal{V}|$ represents the number of entities, $|\mathcal{R}|$ the number of relation types, $|\Gamma|$ the number of entity types, d the embedding dimension, h_d the hidden dimension for MLP networks (for BetaE), L the number of convolution layers (for StarQE and QASE), c the number of cones and h the number of hyperplanes per cone (for QASE). For simplicity, the formulas for Query2Box and BetaE are based on the amount of MLP layers used in their papers (one and three respectively). For QASE, only the *hype-wise* architecture is displayed as a worst-case example.

Table 7: The number of parameters for each baseline model and QASE.

Method	**entities**	**relations**	**projections**	**intersections**
GQE	$\mathbf{E} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times d}$	-	$\mathbf{P} \in \mathbb{R}^{\lvert\mathcal{R}\rvert \times d \times d}$	$\mathbf{I} \in \mathbb{R}^{2\lvert\Gamma\rvert \times d \times d}$
Q2B	$\mathbf{E} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times d}$	-	$\mathbf{P} \in \mathbb{R}^{\lvert\mathcal{R}\rvert \times 2d}$	$\mathbf{I} \in \mathbb{R}^{2 \times 2d \times d}$
BetaE	$\mathbf{E} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times 2d}$	-	$\mathbf{P} \in \mathbb{R}^{\lvert\mathcal{R}\rvert \times 4d \times h_d}$	-
StarQE	$\mathbf{E} \in \mathbb{R}^{(\lvert\mathcal{V}\rvert+2) \times d}$	$\mathbf{R} \in \mathbb{R}^{(2\lvert\mathcal{R}\rvert+1) \times d}$	$\mathbf{C} \in \mathbb{R}^{\lvert L\rvert \times (4d \times d + 2d)*}$	
QASE (h)	$\mathbf{E} \in \mathbb{R}^{(\lvert\mathcal{V}\rvert+2) \times d}$	$\mathbf{R} \in \mathbb{R}^{(2\lvert\mathcal{R}\rvert+1) \times d}$	$\mathbf{C} \in \mathbb{R}^{c \times h \times \lvert L\rvert \times (4d \times d + 2d)*}$	

* StarQE and QASE do not use projection and intersection operators. Instead, we define their learnable convolution parameters as $\mathbf{C}$.

D Baseline models hyperparameters

Table 8 shows the hyperparameters used for the baseline models, which are the hyperparameters described in their respective papers. For StarQE, the optimal entity embedding dimension of 192 has been changed to 800 for a fair comparison between the baseline models.

Table 8: Hyperparameters used for the baseline models.

Method	**dim**	**lr**	**batch size**	**neg samples**	**margin**
GQE	800	0.0001	512	128	30
Q2B	400	0.0001	512	128	30
BetaE	400	0.0001	512	128	60
StarQE	800	0.0007	64	all	-

E Hyperparameter optimization

E.1 QASE

Hyperparameters for QASE are explored for each model architecture and dataset combination (6 in total) where the normal loss function is used. For optimization we consider the entity embedding dimension in (32, 128), the number of cones in {1, 2, 4}, the number of hyperplanes per cone in (32, 128), weight dropout in (0, 0.5), activation function in {LeakyReLU, Identity, ReLU}, composition operators in {multiplication, complex rotation}, edge dropout in (0, 0.5), batch size in (8, 256) and learning rate in $(10^{-4}, 10^{-2})$. Additionally, for the *hype-wise* architecture we explored with the sharing of weights between the convolution layers of a single GNN. We also implemented cone-dropout, where with probability p one random cone is dropped out, which was optimized for values of p in (0, 0.5). To decrease the number of variables for optimization we used for all model versions: bias in the convolution layers, the TM graph pooling operator, symmetric message weighting, LeakyRelu loss activation, a negative sample size of 1000 and the AdamW optimizer. These constants have shown better performance during initial experimentation. The resulting best hyperparameters that have been used for training are shown in table 9.

E.2 StarQE

Table 10 shows the resulting best hyperparameters for StarQE that have been used for training, which are optimized for both datasets and embedding dimensions 64 and 128.

Table 10: Best hyperparameters used for pretraining with StarQE.

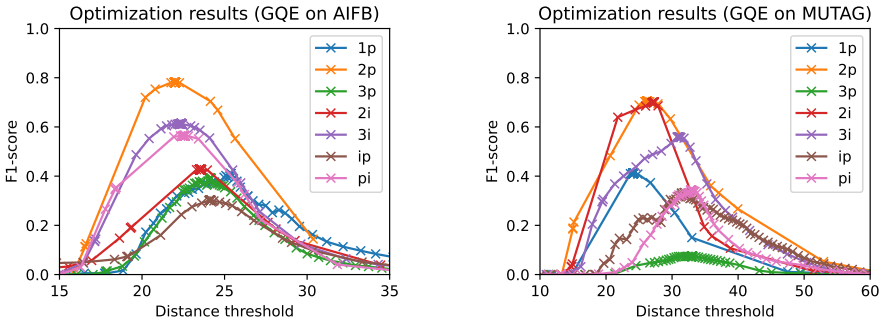
Dataset	dim	layers	lr	batch_size	activation	weighting	dropout	bias
AIFB	64	3	0.001	64	leakyrelu	attention	0.1	True
	128	3	0.0011	128	prelu	symmetric	0.0	False
MUTAG	64	3	0.0011	32	identity	attention	0.3	False
	128	3	0.003	32	relu	attention	0.0	True

Table 9: Best hyperparameters as chosen after running optimization pipeline, including best weighted f1-score on the validation queries (%).

Model	f1-score	embed dim	num cones	num planes	dropout	activ-ation	compos-ition	edge dropout	batch size	lr	share weights	cone dropout
AIFB												
QASE-h	6.93	96	2	128	0.1	LReLU	CompRot	0.179	128	0.0011	False	0.1
QASE-c	6.81	128	1	96	0.3	LReLU	CompRot	0.21	256	0.00028	-	0.0
QASE-s	4.73	96	2	96	0.2	LReLU	CompRot	0.12	64	0.00057	-	0.0
MUTAG												
QASE-h	3.57	96	4	128	0.5	LReLU	CompRot	0.007	32	0.0004	False	0.1
QASE-c	3.21	128	2	96	0.4	ReLU	CompRot	0.23	256	0.00024	-	0.0
QASE-s	1.58	128	2	96	0.1	ReLU	mult	0.26	16	0.00057	-	0.0

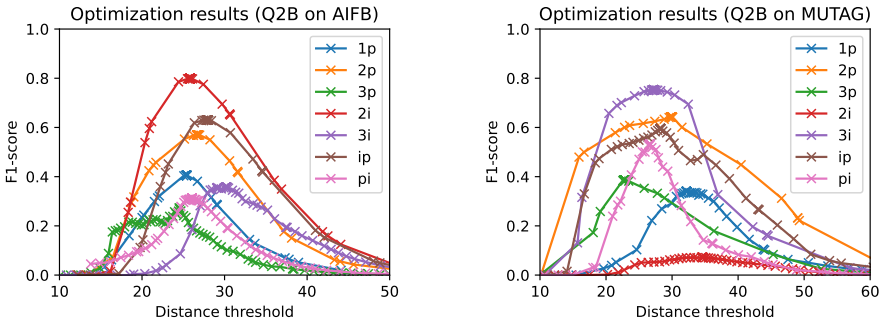
F Threshold optimization results

Figures 9-12 show the structure-wise distance threshold optimization results for GQE, Query2Box, BetaE and StarQE respectively. Note that the actual distance threshold range used for the optimization does not match the range of the x-axis in the figures below, which has been adjusted for clarity. The crosses represent the explored threshold values. The lines between the crosses are filled in to give an approximation of the f1-score for interposing thresholds.



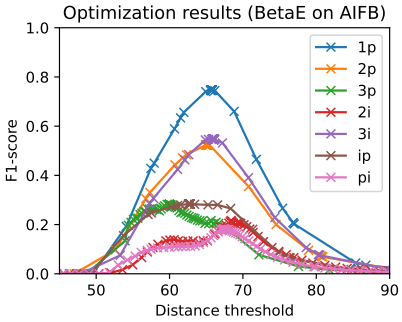
(a) Optimization on AIFB. (b) Optimization on MUTAG.

Fig. 9: Structure-wise threshold optimization of GQE on the validation queries of AIFB and MUTAG.

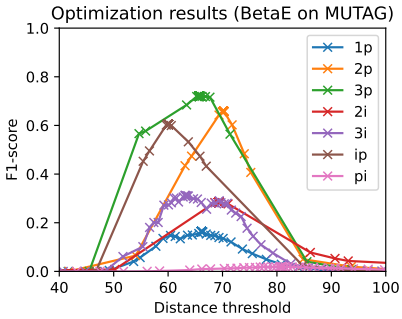


(a) Optimization on AIFB. (b) Optimization on MUTAG.

Fig. 10: Structure-wise threshold optimization of Q2B on the validation queries of AIFB and MUTAG.

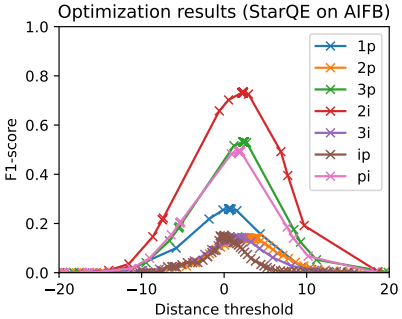


(a) Optimization on AIFB.

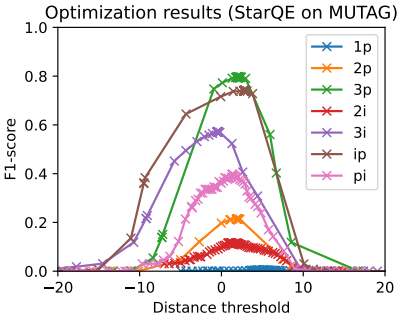


(b) Optimization on MUTAG.

Fig. 11: Structure-wise threshold optimization of BetaE on the validation queries of AIFB and MUTAG.



(a) Optimization on AIFB.



(b) Optimization on MUTAG.

Fig. 12: Structure-wise threshold optimization of StarQE on the validation queries of AIFB and MUTAG.

G Further experimentation

Besides the loss functions introduced in the paper, there has been experimentation with a third more complex loss function. Here, the parameters of the model are optimized by performing gradient descent on a loss function consisting of 3 distinct components – A **contrastive** component that moves the normal vectors in the direction of the target embeddings, and away from negative sampled embeddings; A **push-back** component that pushes the normal vectors away from the target vector to narrow down the answer space; A **diversification** component that actively distributes the normal vectors equally around the target representation to enforce roundness of the cone. The following subsections will discuss their implementation and effects in detail.

Contrastive loss

Given answer space $\mathcal{S} \in \mathbb{R}^{c \times h \times d}$, we iterate over all cones and normal vectors in those cones. The loss value consists of the average negative cosine distance between the normal vector and the target vectors, and the average cosine distance between the normal vector and a collection of negative samples:

$$\mathcal{L}_{\text{CON}} = \sum_{i=0}^{c} \sum_{j=0}^{h} \frac{1}{ch} \left(- \sum_{a \in A_i} \frac{1}{|A_i|} \text{D}_{\cos}(\mathbf{v}_a^+, \mathbf{s}_j^i) + \sum_{p=0}^{l} \frac{1}{l} \text{D}_{\cos}(\mathbf{v}_p^-, \mathbf{s}_j^i) \right)$$

To illustrate the effects of the CON loss component in combination with the assignment function, consider the toy example in figure 13. In this example there are three random target embeddings in green, and two cones, blue and red, each represented by 16 random hyperplanes, all in $\mathbb{R}^3$. Note that the three target entities in green do not cover a single region in the latent space. Other entities have been left out for clarity.

The normal vectors of the toy example are trained on loss component $\mathcal{L}_{\text{CON}}$ using standard gradient descent with a fixed learning rate of 1 for 250 epochs. The result of this can be seen in figure 14. Here each entity is assigned to the closest cone, where the lower left entity is assigned to one cone, and the two upper right entities are assigned to the other cone; the most optimal distribution. All normal vectors of each cone (blue and red) are pointing in the same direction, which means that the corresponding answer space of each individual cone is approximating a single half-space that will likely contain many false-positives if other entities were present.

Push-back loss

To narrow down the cone and reduce the number of false-positives, an additional loss component is required. Since there is already a *pulling* factor in the first loss component, a *pushing* factor is needed to increase the angle between the target and normal vectors. To do this, for each cone a hyperplane is imagined

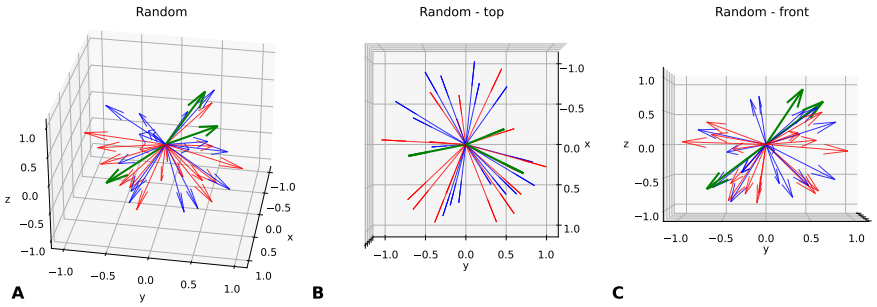


Fig. 13: Random generated target embeddings (n=3, green) and randomly generated normal vectors (n=16) for two cones (blue and red). **A** shows an elevated view, **B** a top view and **C** a front view.

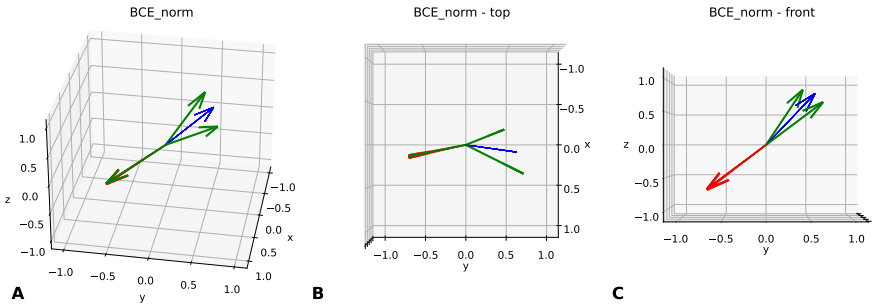


Fig. 14: The result of training on $\mathcal{L}_{\text{CON}}$. **A** shows an elevated view, **B** a top view and **C** a front view.

that has the average assigned target entity representations as its normal vector (approximating the direction of normal vectors trained on $\mathcal{L}_{\text{CON}}$). To increase the angle between the target entities and the normal vectors, the normal vectors of this cone are projected onto this hyperplane and their L2 (Euclidian) norm is maximized. Intuitively, one could see it as a shadow from the sun; a shadow cast on the ground will be longer when the object in question is laying down. We implement this using formula:

$$\mathcal{L}_{\text{norm}} = \sum_{i=0}^{c} \sum_{j=0}^{h} \frac{1}{ch} \left\| \text{rej}(\mathbf{s}_j^i, \hat{\mathbf{t}}_i^+) \right\|^2,$$

where $\text{rej}(\mathbf{n}, \mathbf{t}) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is a rejection function:

$$\text{rej}(\mathbf{n}, \mathbf{t}) = \mathbf{n} - \frac{\mathbf{n} \cdot \mathbf{t}}{\|\mathbf{t}\|^2} \mathbf{t},$$

and $\hat{\mathbf{t}}_i^+$ is the average vector of all assigned target entities of cone i:

$$\hat{\mathbf{t}}_i^+ = \sum_{a \in A_i} \frac{1}{|A_i|} \mathbf{v}_a^+$$

The loss component rejects normal vector $\mathbf{n}$ from average target embedding vector $\mathbf{t}$ such that $\text{rej}(\mathbf{n}, \mathbf{t}) \cdot \mathbf{t} = 0$. In other words, the function returns a new vector that is projected onto the hyperplane defined by $\mathbf{t}$, and the resulting vector is therefore perpendicular to $\mathbf{t}$. The loss value consists of the average L2 (Euclidian) norm for all normal vectors.

Maximizing these vector norms in combination with the CON component ensures that the angle between the normal vectors and some target entity will increase, but never be pushed beyond 90 degrees where the target entity is on the *negative* side of the corresponding hyperplanes. The toy example will now be trained on loss components $\mathcal{L}_{\text{CON}} - \alpha\mathcal{L}_{\text{norm}}$, where α is a hyperparameter that determines the degree to which the normal vectors are pushed back. The results of training with $\alpha = 2$ can be seen in figure 15.

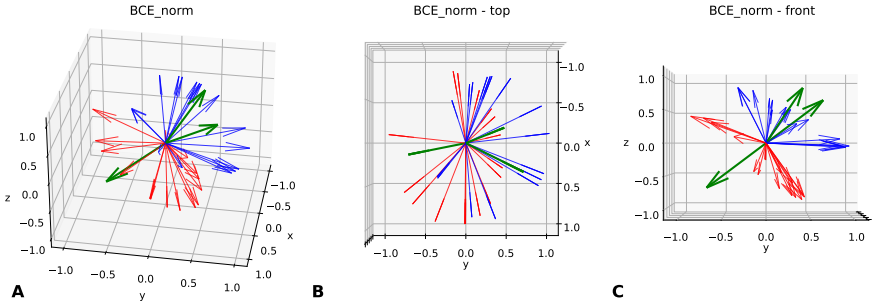


Fig. 15: The result of training on $\mathcal{L}_{\text{CON}} - \alpha\mathcal{L}_{\text{norm}}$, with $\alpha = 2$. **A** shows an elevated view, **B** a top view and **C** a front view.

Experimentation showed that the average angle between the normal vectors of a cone and the average target embedding increases with α, but slower as α increases (figure 16). Because of this we recommend using an α between the values of 0 and 10, preferably at the high end.

Diversification loss

To narrow down the answer space of an individual cone even further, a third and final loss component is introduced. As can be seen in figure 15, normal vectors might not be distributed evenly around the assigned target entity representations. This can result in cones that have angular characteristics instead of being evenly round. To enforce the roundness of cones, the projected norms are actively diversified around the average target entity embeddings by calculating the cosine distance between randomly paired rejected normal vectors:

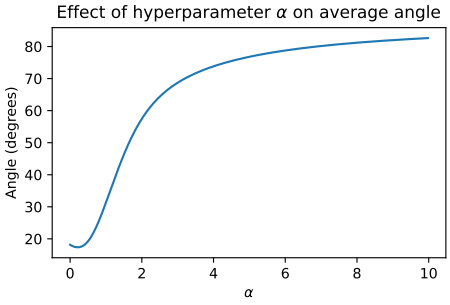


Fig. 16: The average angle between the normal vectors of a cone and the average target embedding as a function of hyperparameter α.

$$\mathcal{L}_{\text{div}} = \sum_{i=0}^{c} \sum_{j=0}^{h} \frac{1}{ch} \text{D}_{\cos}(\text{rej}(\mathbf{s}_j^i, \hat{\mathbf{t}}_i^+), \text{rej}(\mathbf{s}_\chi^i, \hat{\mathbf{t}}_i^+)),$$

where $\chi \overset{\text{R}}{\leftarrow} \{0, 1, ..., h-1\} \setminus \{j\}$ (χ is randomly sampled).

Using the three loss components described above, the final loss function can be defined: $\mathcal{L} = \mathcal{L}_{\text{CON}} - \alpha\mathcal{L}_{\text{norm}} + \beta\mathcal{L}_{\text{div}}$, where β is another hyperparameter. We also experimented with the effect of hyperparameter β on the orientation of the hyperplanes. The difference between the results of training the toy example with a β in range $(0, 10]$ is visually imperceptible. Therefore a value of β in range $(0, 1)$ is recommended to cause as little noise as possible in the learning process. The result of training the toy example on $\mathcal{L}$ with $\alpha = 2, \beta = 0.5$ can be seen in figure 17. The figure shows that the normal vectors are distributed more evenly around the target entity representations, closing some gaps and making the resulting answer space for each individual cone more round.

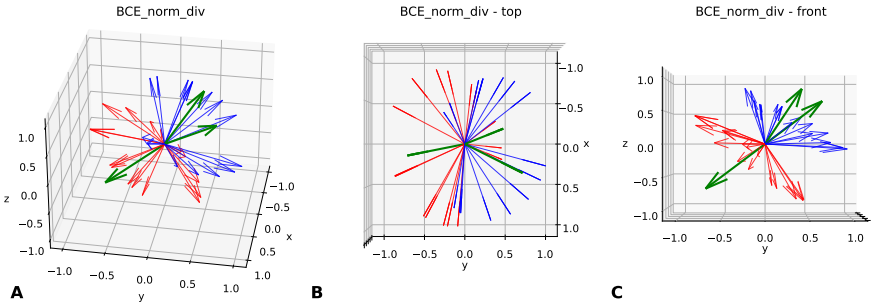


Fig. 17: The result of training on $\mathcal{L}_{\text{CON}} - \alpha\mathcal{L}_{\text{norm}} + \beta\mathcal{L}_{\text{div}}$, with $\alpha = 2, \beta = 0.5$. **A** shows an elevated view, **B** a top view and **C** a front view.

Experimentation results

Training the model with this loss function on the whole dataset using the optimized hyperparameters resulted in weighted f1-scores below 0.1% for both datasets. In addition to this low performance, an important limitation of this approach is that the width of the cone is not dependent on the position of positive and negative samples, but solely on hyperparameter α. Additionally, we hypothesize that the implementation suffers from negative sample embeddings getting stuck inside the cone. Looking at figure 17 we can see that for negative samples inside the answer space the optimal direction would be to move further inside the cone instead of moving towards the edge. Because of this we decided not to continue with this line of experimentation.

Similar to the loss functions introduced in the paper, we also experimented with focusing on the most promising normal vectors to exclude negative samples by using the Softmin weighted average or minimum of the cosine distances between a negative sample and all normal vectors in a cone. These alternative implementations did not yield better performances.

Finally we experimented with adding only the diversity loss component to the loss functions described in the paper, with the aim of increasing the roundness of the answer space. The model did not converge when trained on this combined loss function.