Towards Understanding the Geometry of Knowledge Graph Embeddings

Chandrahas
Indian Institute of Science
chandrahas@iisc.ac.in

Aditya Sharma
Indian Institute of Science
adityasharma@iisc.ac.in

Partha Talukdar
Indian Institute of Science
ppt@iisc.ac.in

Abstract

Knowledge Graph (KG) embedding has emerged as a very active area of research over the last few years, resulting in the development of several embedding methods. These KG embedding methods represent KG entities and relations as vectors in a high-dimensional space. Despite this popularity and effectiveness of KG embeddings in various tasks (e.g., link prediction), geometric understanding of such embeddings (i.e., arrangement of entity and relation vectors in vector space) is unexplored – we fill this gap in the paper. We initiate a study to analyze the geometry of KG embeddings and correlate it with task performance and other hyperparameters. To the best of our knowledge, this is the first study of its kind. Through extensive experiments on real-world datasets, we discover several insights. For example, we find that there are sharp differences between the geometry of embeddings learnt by different classes of KG embeddings methods. We hope that this initial study will inspire other follow-up research on this important but unexplored problem.

1 Introduction

Knowledge Graphs (KGs) are multi-relational graphs where nodes represent entities and typed-edges represent relationships among entities. Recent research in this area has resulted in the development of several large KGs, such as NELL (Mitchell et al., 2015), YAGO (Suchanek et al., 2007), and Freebase (Bollacker et al., 2008), among others. These KGs contain thousands of predicates (e.g., *person*, *city*, *mayorOf(person, city)*, etc.), and millions of triples involving such predicates, e.g., *(Bill de Blasio, mayorOf, New York City)*.

The problem of learning embeddings for Knowledge Graphs has received significant attention in recent years, with several methods being proposed (Bordes et al., 2013; Lin et al., 2015; Nguyen et al., 2016; Nickel et al., 2016; Trouillon et al., 2016). These methods represent entities and relations in a KG as vectors in high dimensional space. These vectors can then be used for various tasks, such as, link prediction, entity classification etc. Starting with TransE (Bordes et al., 2013), there have been many KG embedding methods such as TransH (Wang et al., 2014), TransR (Lin et al., 2015) and STransE (Nguyen et al., 2016) which represent relations as translation vectors from head entities to tail entities. These are *additive models*, as the vectors interact via addition and subtraction. Other KG embedding models, such as, DistMult (Yang et al., 2014), HolE (Nickel et al., 2016), and ComplEx (Trouillon et al., 2016) are *multiplicative* where entity-relation-entity triple likelihood is quantified by a multiplicative score function. All these methods employ a score function for distinguishing correct triples from incorrect ones.

In spite of the existence of many KG embedding methods, our understanding of the geometry and structure of such embeddings is very shallow. A recent work (Mimno and Thompson, 2017) analyzed the geometry of word embeddings. However, the problem of analyzing geometry of KG embeddings is still unexplored – we fill this important gap. In this paper, we analyze the geometry of such vectors in terms of their lengths and conicity, which, as defined in Section 4, describes their positions and orientations in the vector space. We later study the effects of model type and training hyperparameters on the geometry of KG embeddings and correlate geometry with performance.

We make the following contributions:

- We initiate a study to analyze the geometry of various Knowledge Graph (KG) embeddings. To the best of our knowledge, this is the first study of its kind. We also formalize various metrics which can be used to study geometry of a set of vectors.
- Through extensive analysis, we discover several interesting insights about the geometry of KG embeddings. For example, we find systematic differences between the geometries of embeddings learned by additive and multiplicative KG embedding methods.
- We also study the relationship between geometric attributes and predictive performance of the embeddings, resulting in several new insights. For example, in case of multiplicative models, we observe that for entity vectors generated with a fixed number of negative samples, lower conicity (as defined in Section 4) or higher average vector length lead to higher performance.

Source code of all the analysis tools developed as part of this paper is available at `https://github.com/malllabiisc/kg-geometry`. We are hoping that these resources will enable one to quickly analyze the geometry of any KG embedding, and potentially other embeddings as well.

2 Related Work

In spite of the extensive and growing literature on both KG and non-KG embedding methods, very little attention has been paid towards understanding the geometry of the learned embeddings. A recent work (Mimno and Thompson, 2017) is an exception to this which addresses this problem in the context of word vectors. This work revealed a surprising correlation between word vector geometry and the number of negative samples used during training. Instead of word vectors, in this paper we focus on understanding the geometry of KG embeddings. In spite of this difference, the insights we discover in this paper generalizes some of the observations in the work of (Mimno and Thompson, 2017). Please see Section 6.2 for more details.

Since KGs contain only positive triples, negative sampling has been used for training KG embeddings. Effect of the number of negative samples in KG embedding performance was studied by (Toutanova et al., 2015). In this paper, we study the effect of the number of negative samples on KG embedding geometry as well as performance.

In addition to the additive and multiplicative KG embedding methods already mentioned in Section 1, there is another set of methods where the entity and relation vectors interact via a neural network. Examples of methods in this category include NTN (Socher et al., 2013), CONV (Toutanova et al., 2015), ConvE (Dettmers et al., 2017), R-GCN (Schlichtkrull et al., 2017), ER-MLP (Dong et al., 2014) and ER-MLP-2n (Ravishankar et al., 2017). Due to space limitations, in this paper we restrict our scope to the analysis of the geometry of additive and multiplicative KG embedding models only, and leave the analysis of the geometry of neural network-based methods as part of future work.

3 Overview of KG Embedding Methods

For our analysis, we consider six representative KG embedding methods: TransE (Bordes et al., 2013), TransR (Lin et al., 2015), STransE (Nguyen et al., 2016), DistMult (Yang et al., 2014), HolE (Nickel et al., 2016) and ComplEx (Trouillon et al., 2016). We refer to TransE, TransR and STransE as *additive* methods because they learn embeddings by modeling relations as translation vectors from one entity to another, which results in vectors interacting via the addition operation during training. On the other hand, we refer to DistMult, HolE and ComplEx as *multiplicative* methods as they quantify the likelihood of a triple belonging to the KG through a multiplicative score function. The score functions optimized by these methods are summarized in Table 1.

Notation: Let $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ be a Knowledge Graph (KG) where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relations and $\mathcal{T} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is the set of triples stored in the graph. Most of the KG embedding methods learn vectors $\mathbf{e} \in \mathbb{R}^{d_e}$ for $e \in \mathcal{E}$, and $\mathbf{r} \in \mathbb{R}^{d_r}$ for $r \in \mathcal{R}$. Some methods also learn projection matrices $M_r \in \mathbb{R}^{d_r \times d_e}$ for relations. The correctness of a triple is evaluated using a model specific score function $\sigma : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$. For learning the embeddings, a loss function $\mathcal{L}(\mathcal{T}, \mathcal{T}'; \theta)$, defined over a set of positive triples $\mathcal{T}$, set of (sampled) negative triples $\mathcal{T}'$, and the parameters θ is optimized.

We use small italics characters (e.g., h, r) to represent entities and relations, and correspond-

Type	Model	Score Function $\sigma(h,r,t)$
Additive	TransE (Bordes et al., 2013)	$-\left\|\mathbf{h}+\mathbf{r}-\mathbf{t}\right\|_1$
	TransR (Lin et al., 2015)	$-\left\|M_r\mathbf{h}+\mathbf{r}-M_r\mathbf{t}\right\|_1$
	STransE (Nguyen et al., 2016)	$-\left\|M_r^1\mathbf{h}+\mathbf{r}-M_r^2\mathbf{t}\right\|_1$
Multiplicative	DistMult (Yang et al., 2014)	$\mathbf{r}^\top(\mathbf{h}\odot\mathbf{t})$
	HolE (Nickel et al., 2016)	$\mathbf{r}^\top(\mathbf{h}\star\mathbf{t})$
	ComplEx (Trouillon et al., 2016)	$\mathbf{Re}(\mathbf{r}^\top(\mathbf{h}\odot\bar{\mathbf{t}}))$

Table 1: Summary of various Knowledge Graph (KG) embedding methods used in the paper. Please see Section 3 for more details.

ing bold characters to represent their vector embeddings (e.g., $\mathbf{h}$, $\mathbf{r}$). We use bold capitalization (e.g., $\mathbf{V}$) to represent a set of vectors. Matrices are represented by capital italics characters (e.g., M).

3.1 Additive KG Embedding Methods

This is the set of methods where entity and relation vectors interact via additive operations. The score function for these models can be expressed as below

$$\sigma(h,r,t) = -\left\|M_r^1\mathbf{h}+\mathbf{r}-M_r^2\mathbf{t}\right\|_1 \quad (1)$$

where $\mathbf{h},\mathbf{t} \in \mathbb{R}^{d_e}$ and $\mathbf{r} \in \mathbb{R}^{d_r}$ are vectors for head entity, tail entity and relation respectively. $M_r^1, M_r^2 \in \mathbb{R}^{d_r\times d_e}$ are projection matrices from entity space $\mathbb{R}^{d_e}$ to relation space $\mathbb{R}^{d_r}$.

TransE (Bordes et al., 2013) is the simplest additive model where the entity and relation vectors lie in same $d-$dimensional space, i.e., $d_e = d_r = d$. The projection matrices $M_r^1 = M_r^2 = \mathcal{I}_d$ are identity matrices. The relation vectors are modeled as translation vectors from head entity vectors to tail entity vectors. Pairwise ranking loss is then used to learn these vectors. Since the model is simple, it has limited capability in capturing many-to-one, one-to-many and many-to-many relations.

TransR (Lin et al., 2015) is another translation-based model which uses separate spaces for entity and relation vectors allowing it to address the shortcomings of TransE. Entity vectors are projected into a relation specific space using the corresponding projection matrix $M_r^1 = M_r^2 = M_r$. The training is similar to TransE.

STransE (Nguyen et al., 2016) is a generalization of TransR and uses different projection matrices for head and tail entity vectors. The training is similar to TransE. STransE achieves better performance than the previous methods but at the cost of more number of parameters.

Equation 1 is the score function used in STransE. TransE and TransR are special cases of STransE with $M_r^1 = M_r^2 = \mathcal{I}_d$ and $M_r^1 = M_r^2 = M_r$, respectively.

3.2 Multiplicative KG Embedding Methods

This is the set of methods where the vectors interact via multiplicative operations (usually dot product). The score function for these models can be expressed as

$$\sigma(h,r,t) = \mathbf{r}^\top f(\mathbf{h},\mathbf{t}) \quad (2)$$

where $\mathbf{h},\mathbf{t},\mathbf{r} \in \mathbb{F}^d$ are vectors for head entity, tail entity and relation respectively. $f(\mathbf{h},\mathbf{t}) \in \mathbb{F}^d$ measures compatibility of head and tail entities and is specific to the model. $\mathbb{F}$ is either real space $\mathbb{R}$ or complex space $\mathbb{C}$. Detailed descriptions of the models we consider are as follows.

DistMult (Yang et al., 2014) models entities and relations as vectors in $\mathbb{R}^d$. It uses an entry-wise product ($\odot$) to measure compatibility between head and tail entities, while using logistic loss for training the model.

$$\sigma_{DistMult}(h,r,t) = \mathbf{r}^\top(\mathbf{h}\odot\mathbf{t}) \quad (3)$$

Since the entry-wise product in (3) is symmetric, DistMult is not suitable for asymmetric and anti-symmetric relations.

HolE (Nickel et al., 2016) also models entities and relations as vectors in $\mathbb{R}^d$. It uses circular correlation operator ($\star$) as compatibility function defined as

$$[\mathbf{h}\star\mathbf{t}]_k = \sum_{i=0}^{d-1} \mathbf{h}_i\mathbf{t}_{(k+i)\bmod d}$$

The score function is given as

$$\sigma_{HolE}(h,r,t) = \mathbf{r}^\top(\mathbf{h}\star\mathbf{t}) \quad (4)$$

The circular correlation operator being asymmetric, can capture asymmetric and anti-symmetric relations, but at the cost of higher time complexity

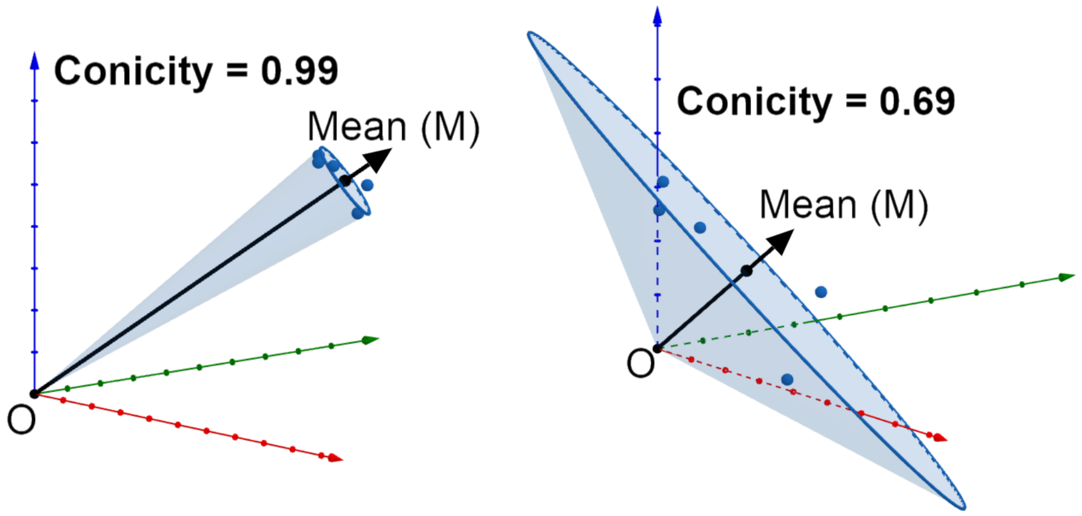


Figure 1: Comparison of high vs low Conicity. Randomly generated vectors are shown in blue with their sample mean vector M in black. Figure on the left shows the case when vectors lie in narrow cone resulting in high Conicity value. Figure on the right shows the case when vectors are spread out having relatively lower Conicity value. We skipped very low values of Conicity as it was difficult to visualize. The points are sampled from 3d Spherical Gaussian with mean (1,1,1) and standard deviation 0.1 (left) and 1.3 (right). Please refer to Section 4 for more details.

($\mathcal{O}(d \log d)$). For training, we use pairwise ranking loss.

ComplEx (Trouillon et al., 2016) represents entities and relations as vectors in $\mathbb{C}^d$. The compatibility of entity pairs is measured using entry-wise product between head and complex conjugate of tail entity vectors.

$$\sigma_{ComplEx}(h,r,t) = \mathbf{Re}(\mathbf{r}^\top(\mathbf{h} \odot \bar{\mathbf{t}})) \qquad (5)$$

In contrast to (3), using complex vectors in (5) allows ComplEx to handle symmetric, asymmetric and anti-symmetric relations using the same score function. Similar to DistMult, logistic loss is used for training the model.

4 Metrics

For our geometrical analysis, we first define a term **'alignment to mean'** (**ATM**) of a vector $\mathbf{v}$ belonging to a set of vectors $\mathbf{V}$, as the cosine similarity[1] between $\mathbf{v}$ and the mean of all vectors in $\mathbf{V}$.

$$\text{ATM}(\mathbf{v}, \mathbf{V}) = \text{cosine}\left(\mathbf{v}, \frac{1}{|\mathbf{V}|}\sum_{\mathbf{x}\in\mathbf{V}} \mathbf{x}\right)$$

We also define '**conicity**' of a set $\mathbf{V}$ as the mean ATM of all vectors in $\mathbf{V}$.

$$\text{Conicity}(\mathbf{V}) = \frac{1}{|\mathbf{V}|}\sum_{\mathbf{v}\in\mathbf{V}} \text{ATM}(\mathbf{v}, \mathbf{V})$$

[1]cosine$(u, v) = \frac{u^\top v}{\|u\|\|v\|}$

Dataset		FB15k	WN18
#Relations		1,345	18
#Entities		14,541	40,943
#Triples	Train	483,142	141,440
	Validation	50,000	5,000
	Test	59,071	5,000

Table 2: Summary of datasets used in the paper.

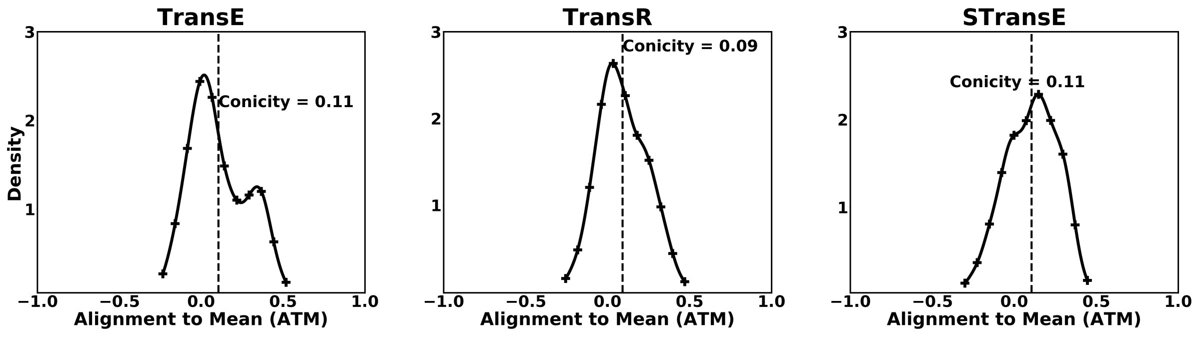
By this definition, a high value of Conicity($\mathbf{V}$) would imply that the vectors in $\mathbf{V}$ lie in a narrow cone centered at origin. In other words, the vectors in the set $\mathbf{V}$ are highly aligned with each other. In addition to that, we define the variance of ATM across all vectors in $\mathbf{V}$, as the '**vector spread**'(VS) of set $\mathbf{V}$,

$$\text{VS}(\mathbf{V}) = \frac{1}{|\mathbf{V}|}\sum_{\mathbf{v}\in\mathbf{V}} \Big(\text{ATM}(\mathbf{v}, \mathbf{V}) - \text{Conicity}(\mathbf{V})\Big)^2$$

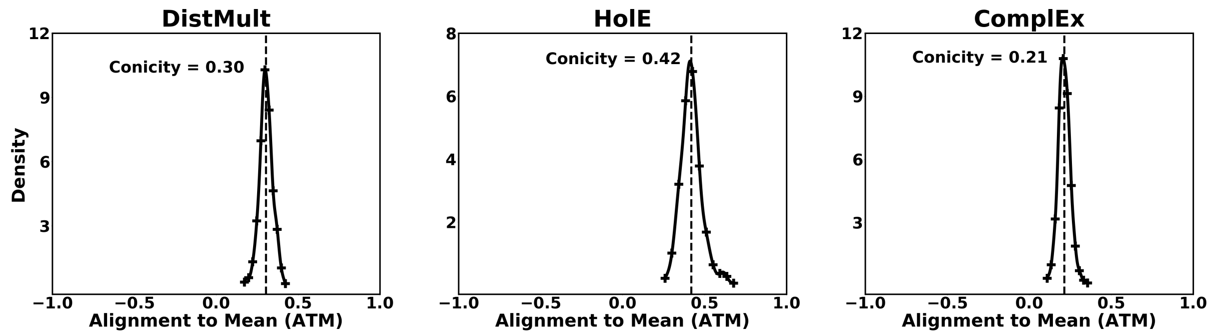
Figure 1 visually demonstrates these metrics for randomly generated 3-dimensional points. The left figure shows high Conicity and low vector spread while the right figure shows low Conicity and high vector spread.

We define the length of a vector $\mathbf{v}$ as L_2-norm of the vector $\|\mathbf{v}\|_2$ and **'average vector length'** (**AVL**) for the set of vectors $\mathbf{V}$ as

$$\text{AVL}(\mathbf{V}) = \frac{1}{|\mathbf{V}|}\sum_{\mathbf{v}\in\mathbf{V}} \|\mathbf{v}\|_2$$



(a) Additive Models



(b) Multiplicative Models

Figure 2: Alignment to Mean (ATM) vs Density plots for entity embeddings learned by various additive (top row) and multiplicative (bottom row) KG embedding methods. For each method, a plot averaged across entity frequency bins is shown. From these plots, we conclude that entity embeddings from additive models tend to have low (positive as well as negative) ATM and thereby low Conicity and high vector spread. Interestingly, this is reversed in case of multiplicative methods. Please see Section 6.1 for more details.

5 Experimental Setup

Datasets: We run our experiments on subsets of two widely used datasets, viz., Freebase (Bollacker et al., 2008) and WordNet (Miller, 1995), called FB15k and WN18 (Bordes et al., 2013), respectively. We detail the characteristics of these datasets in Table 2.

Please note that while the results presented in Section 6 are on the FB15K dataset, we reach the same conclusions on WN18. The plots for our experiments on WN18 can be found in the Supplementary Section.

Hyperparameters: We experiment with multiple values of hyperparameters to understand their effect on the geometry of KG embeddings. Specifically, we vary the dimension of the generated vectors between $\{50, 100, 200\}$ and the number of negative samples used during training between $\{1, 50, 100\}$. For more details on algorithm specific hyperparameters, we refer the reader to the Supplementary Section.[2]

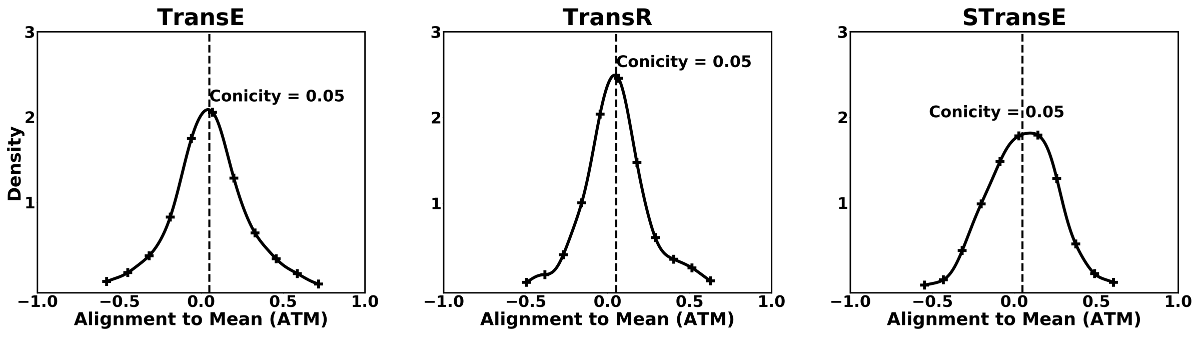
Frequency Bins: We follow (Mimno and Thompson, 2017) for entity and relation samples used in the analysis. Multiple bins of entities and relations are created based on their frequencies and 100 randomly sampled vectors are taken from each bin. These set of sampled vectors are then used for our analysis. For more information about sampling vectors, please refer to (Mimno and Thompson, 2017).

6 Results and Analysis

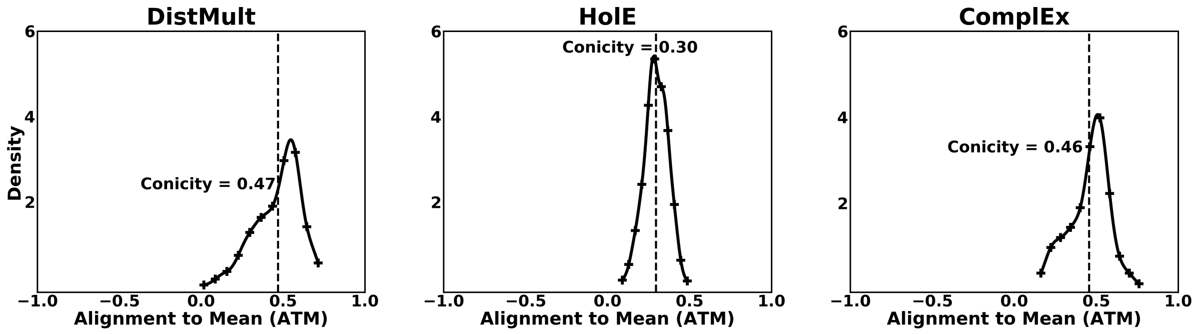
In this section, we evaluate the following questions.

- Does model type (e.g., additive vs multiplicative) have any effect on the geometry of embeddings? (Section 6.1)

[2]For training, we used codes from `https://github.com/Mrlyk423/Relation_Extraction` (TransE, TransR), `https://github.com/datquocnguyen/STransE` (STransE), `https://github.com/mnick/holographic-embeddings` (HolE) and `https://github.com/ttrouill/complex` (ComplEx and DistMult).



(a) Additive Models



(b) Multiplicative Models

Figure 3: Alignment to Mean (ATM) vs Density plots for relation embeddings learned by various additive (top row) and multiplicative (bottom row) KG embedding methods. For each method, a plot averaged across entity frequency bins is shown. Trends in these plots are similar to those in Figure 2. Main findings from these plots are summarized in Section 6.1.

- Does negative sampling have any effect on the embedding geometry? (Section 6.2)
- Does the dimension of embedding have any effect on its geometry? (Section 6.3)
- How is task performance related to embedding geometry? (Section 6.4)

In each subsection, we summarize the main findings at the beginning, followed by evidence supporting those findings.

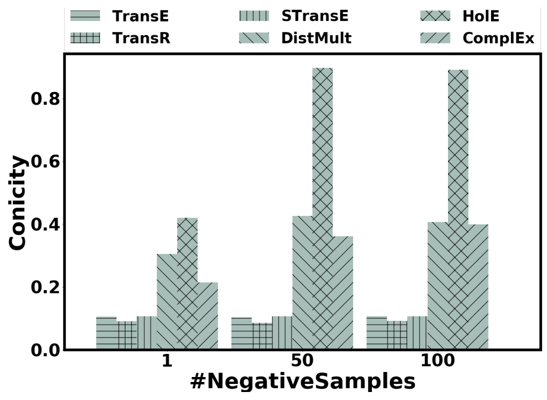
6.1 Effect of Model Type on Geometry

Summary of Findings:
Additive: Low conicity and high vector spread.
Multiplicative: High conicity and low vector spread.

In this section, we explore whether the type of the score function optimized during the training has any effect on the geometry of the resulting embedding. For this experiment, we set the number of negative samples to 1 and the vector dimension to 100 (we got similar results for 50-dimensional vectors). Figure 2 and Figure 3 show the distribution of ATMs of these sampled entity and relation vectors, respectively.[3]

Entity Embeddings: As seen in Figure 2, there is a stark difference between the geometries of entity vectors produced by additive and multiplicative models. The ATMs of all entity vectors produced by multiplicative models are positive with very low vector spread. Their high conicity suggests that they are not uniformly dispersed in the vector space, but lie in a narrow cone along the mean vector. This is in contrast to the entity vectors obtained from additive models which are both positive and negative with higher vector spread. From the lower values of conicity, we conclude that entity vectors from additive models are evenly dispersed in the vector space. This observation is also reinforced by looking at the high vector spread of additive models in comparison to that of multiplicative models. We also observed that additive models are sensitive to the frequency of entities, with high frequency bins having higher conicity than low frequency bins. However, no such pattern was observed for multiplicative models and

[3]We also tried using the *global* mean instead of mean of the sampled set for calculating cosine similarity in ATM, and got very similar results.



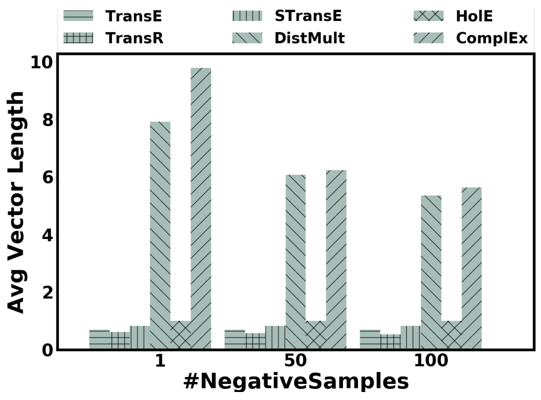


Figure 4: Conicity (left) and Average Vector Length (right) vs Number of negative samples for entity vectors learned using various KG embedding methods. In each bar group, first three models are additive, while the last three are multiplicative. Main findings from these plots are summarized in Section 6.2

conicity was consistently similar across frequency bins. For clarity, we have not shown different plots for individual frequency bins.

Relation Embeddings: As in entity embeddings, we observe a similar trend when we look at the distribution of ATMs for relation vectors in Figure 3. The conicity of relation vectors generated using additive models is almost zero across frequency bands. This coupled with the high vector spread observed, suggests that these vectors are scattered throughout the vector space. Relation vectors from multiplicative models exhibit high conicity and low vector spread, suggesting that they lie in a narrow cone centered at origin, like their entity counterparts.

6.2 Effect of Number of Negative Samples on Geometry

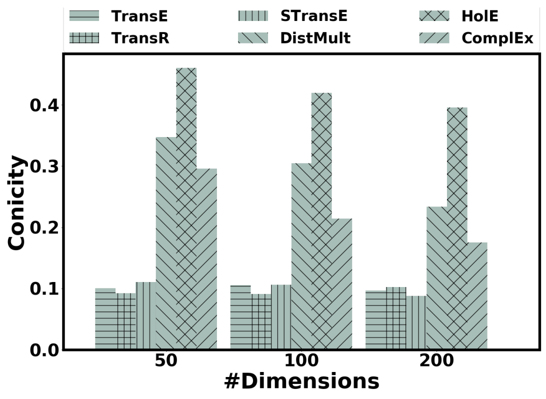
Summary of Findings:
Additive: Conicity and average length are invariant to changes in #NegativeSamples for both entities and relations.
Multiplicative: Conicity increases while average vector length decrease with increasing #NegativeSamples for entities. Conicity decreases, while average vector length remains constant (except HolE) for relations.

For experiments in this section, we keep the vector dimension constant at 100.

Entity Embeddings: As seen in Figure 4 (left), the conicity of entity vectors increases as the number of negative samples is increased for multiplicative models. In contrast, conicity of the entity vectors generated by additive models is unaffected by change in number of negative samples and they continue to be dispersed throughout the vector space. From Figure 4 (right), we observe that the average length of entity vectors produced by additive models is also invariant of any changes in number of negative samples. On the other hand, increase in negative sampling decreases the average entity vector length for all multiplicative models except HolE. The average entity vector length for HolE is nearly 1 for any number of negative samples, which is understandable considering it constrains the entity vectors to lie inside a unit ball (Nickel et al., 2016). This constraint is also enforced by the additive models: TransE, TransR, and STransE.

Relation Embeddings: Similar to entity embeddings, in case of relation vectors trained using additive models, the average length and conicity do not change while varying the number of negative samples. However, the conicity of relation vectors from multiplicative models decreases with increase in negative sampling. The average relation vector length is invariant for all multiplicative methods, except for HolE. We see a surprisingly big jump in average relation vector length for HolE going from 1 to 50 negative samples, but it does not change after that. Due to space constraints in the paper, we refer the reader to the Supplementary Section for plots discussing the effect of number of negative samples on geometry of relation vectors.

We note that the multiplicative score between two vectors may be increased by either increasing the alignment between the two vectors (i.e., increasing Conicity and reducing vector spread between them), or by increasing their lengths. It is interesting to note that we see exactly these effects in the geometry of multiplicative methods



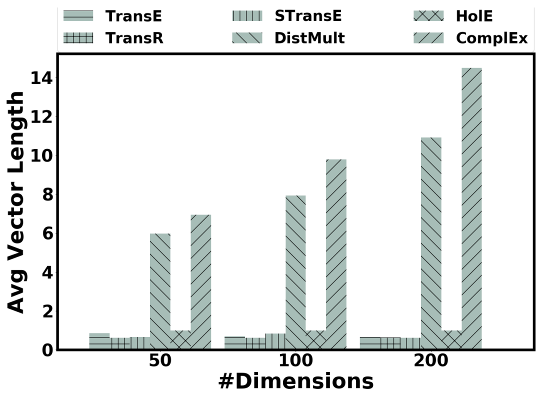


Figure 5: Conicity (left) and Average Vector Length (right) vs Number of Dimensions for entity vectors learned using various KG embedding methods. In each bar group, first three models are additive, while the last three are multiplicative. Main findings from these plots are summarized in Section 6.3.

analyzed above.

6.2.1 Correlation with Geometry of Word Embeddings

Our conclusions from the geometrical analysis of entity vectors produced by multiplicative models are similar to the results in (Mimno and Thompson, 2017), where increase in negative sampling leads to increased conicity of word vectors trained using the skip-gram with negative sampling (SGNS) method. On the other hand, additive models remain unaffected by these changes.

SGNS tries to maximize a score function of the form $\mathbf{w}^T \cdot \mathbf{c}$ for positive word context pairs, where $\mathbf{w}$ is the word vector and $\mathbf{c}$ is the context vector (Mikolov et al., 2013). This is very similar to the score function of multiplicative models as seen in Table 1. Hence, SGNS can be considered as a multiplicative model in the word domain.

Hence, we argue that our result on the increase in negative samples increasing the conicity of vectors trained using a multiplicative score function can be considered as a generalization of the one in (Mimno and Thompson, 2017).

6.3 Effect of Vector Dimension on Geometry

Summary of Findings:
Additive: Conicity and average length are invariant to changes in dimension for both entities and relations.
Multiplicative: Conicity decreases for both entities and relations with increasing dimension. Average vector length increases for both entities and relations, except for HolE entities.

Entity Embeddings: To study the effect of vector dimension on conicity and length, we set the number of negative samples to 1, while varying the vector dimension. From Figure 5 (left), we observe that the conicity for entity vectors generated by any additive model is almost invariant of increase in dimension, though STransE exhibits a slight decrease. In contrast, entity vector from multiplicative models show a clear decreasing pattern with increasing dimension.

As seen in Figure 5 (right), the average lengths of entity vectors from multiplicative models increase sharply with increasing vector dimension, except for HolE. In case of HolE, the average vector length remains constant at one. Deviation involving HolE is expected as it enforces entity vectors to fall within a unit ball. Similar constraints are enforced on entity vectors for additive models as well. Thus, the average entity vector lengths are not affected by increasing vector dimension for all additive models.

Relation Embeddings: We reach similar conclusion when analyzing against increasing dimension the change in geometry of relation vectors produced using these KG embedding methods. In this setting, the average length of relation vectors learned by HolE also increases as dimension is increased. This is consistent with the other methods in the multiplicative family. This is because, unlike entity vectors, the lengths of relation vectors of HolE are not constrained to be less than unit length. Due to lack of space, we are unable to show plots for relation vectors here, but the same can be found in the Supplementary Section.

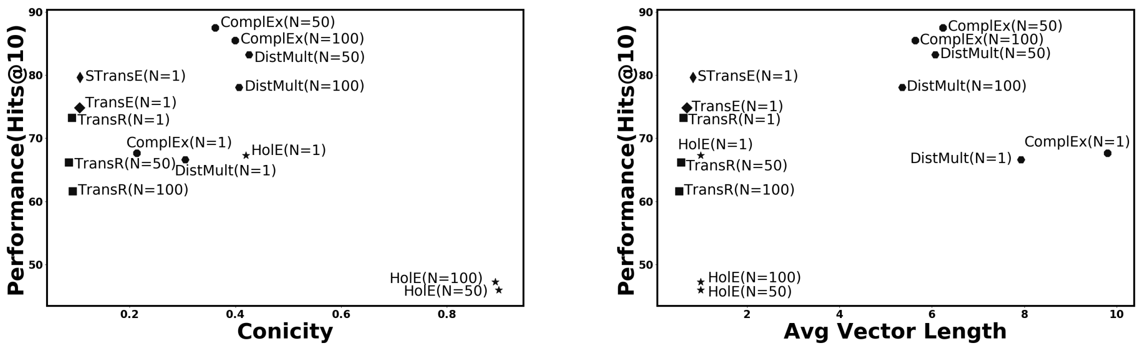


Figure 6: Relationship between Performance (HITS@10) on a link prediction task vs Conicity (left) and Avg. Vector Length (right). For each point, N represents the number of negative samples used. Main findings are summarized in Section 6.4.

6.4 Relating Geometry to Performance

Summary of Findings:
Additive: Neither entites nor relations exhibit correlation between geometry and performance.
Multiplicative: Keeping negative samples fixed, lower conicity or higher average vector length for entities leads to improved performance. No relationship for relations.

In this section, we analyze the relationship between geometry and performance on the Link prediction task, using the same setting as in (Bordes et al., 2013). Figure 6 (left) presents the effects of conicity of entity vectors on performance, while Figure 6 (right) shows the effects of average entity vector length.[4]

As we see from Figure 6 (left), for fixed number of negative samples, the multiplicative model with lower conicity of entity vectors achieves better performance. This performance gain is larger for higher numbers of negative samples (N). Additive models don't exhibit any relationship between performance and conicity, as they are all clustered around zero conicity, which is in-line with our observations in previous sections. In Figure 6 (right), for all multiplicative models except HolE, a higher average entity vector length translates to better performance, while the number of negative samples is kept fixed. Additive models and HolE don't exhibit any such patterns, as they are all clustered just below unit average entity vector length.

The above two observations for multiplicative models make intuitive sense, as lower conicity and higher average vector length would both translate to vectors being more dispersed in the space.

We see another interesting observation regarding the high sensitivity of HolE to the number of negative samples used during training. Using a large number of negative examples (e.g., $N = 50$ or 100) leads to very high conicity in case of HolE. Figure 6 (right) shows that average entity vector length of HolE is always one. These two observations point towards HolE's entity vectors lying in a tiny part of the space. This translates to HolE performing poorer than all other models in case of high numbers of negative sampling.

We also did a similar study for relation vectors, but did not see any discernible patterns.

7 Conclusion

In this paper, we have initiated a systematic study into the important but unexplored problem of analyzing geometry of various Knowledge Graph (KG) embedding methods. To the best of our knowledge, this is the first study of its kind. Through extensive experiments on multiple real-world datasets, we are able to identify several insights into the geometry of KG embeddings. We have also explored the relationship between KG embedding geometry and its task performance. We have shared all our source code to foster further research in this area.

Acknowledgements

We thank the anonymous reviewers for their constructive comments. This work is supported in part by the Ministry of Human Resources Development (Government of India), Intel, Intuit, and by gifts from Google and Accenture.

[4]A more focused analysis for multiplicative models is presented in Section 3 of Supplementary material.

References

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. AcM, pages 1247–1250.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*. pages 2787–2795.

T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. 2017. Convolutional 2D Knowledge Graph Embeddings. *ArXiv e-prints* .

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pages 601–610.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*. pages 2181–2187.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. pages 3111–3119.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM* 38(11):39–41.

David Mimno and Laure Thompson. 2017. The strange geometry of skip-gram with negative sampling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. pages 2863–2868.

T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. 2015. Never-ending learning. In *Proceedings of AAAI*.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016. Stranse: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of NAACL-HLT*. pages 460–466.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. 2016. Holographic embeddings of knowledge graphs. In *AAAI*.

Srinivas Ravishankar, Chandrahas, and Partha Pratim Talukdar. 2017. Revisiting simple neural networks for learning representations of knowledge graphs. *6th Workshop on Automated Knowledge Base Construction (AKBC) at NIPS 2017* .

M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. 2017. Modeling Relational Data with Graph Convolutional Networks. *ArXiv e-prints* .

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*. pages 926–934.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing Text for Joint Embedding of Text and Knowledge Bases. In *Empirical Methods in Natural Language Processing (EMNLP)*. ACL Association for Computational Linguistics.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *ICML*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*. Citeseer, pages 1112–1119.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* .