Geometric Queries in Word Embedding

Gal Beniamini (305207136) and Assaf Brown (203457700)

Abstract

Word embedding is a common methodology for natural language processing tasks. Embedding algorithms produce succinct representations which can be utilised to answer semantic queries such as locating a word's "nearest neighbours", and even discovering abstract relationships between words. While these models can be used to reason about single words, they do not offer a straightforward method for reasoning about *collections of words*, such as phrases or sentences.

In this work, we explore the aspect of embedding from a geometric standpoint; instead of viewing sets of words simply as points within the embedded space, we form shapes corresponding to multiple words, and answer semantic queries by computing properties of the above shapes.

To this end, we both implemented and evaluated different embedding techniques, proposed several intuitive algorithms for modeling sets of words, implemented each algorithm, and analysed their respective performance.

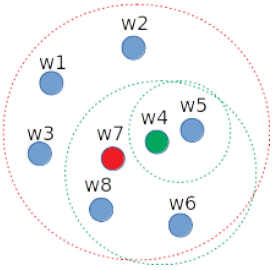

Fig 1: Bounding words with n-dimensional spheres

Word Embedding

Word embedding is a wide-spread technique, in which a compact representation of words is computed (usually as vectors in some high-dimensional space), with the goal of preserving a subset of the semantic properties of the underlying text.

Most commonly, the embedding is a two-step process; first, statistics relating to the co-occurrences of words with their neighbours are gathered. Then, a compact representation is extracted, based on the relative frequencies, using which efficient computations may be performed. Current state-of-the-art algorithms for embedding, such as GloVe[1] and Word2Vec,[2] follow the model described above (although they differ in their method for constructing the low-dimension representation).

Once an embedding is created, it can be utilised to answer semantic queries. Some queries are relatively straightforward; for example, the "nearest neighbours" (under some norm) of a given word are likely to include words from a semantically similar domain. Moreover, in some cases,

---

[1] Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[2] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

unique properties emerge from the resulting embedding — for example, under Word2Vec, vector arithmetic on word vectors can sometimes be used to deduce abstract relationships between them. Consequently, removing the 'man' vector from 'king' and adding 'woman' to it, results in a vector similar to 'queen'.

Much like the algorithms above, we decided to research our own techniques for the creation of word embeddings. To this end, we first computed the full co-occurrence matrix over the corpus. Then, we normalized each row according to the L1 norm, resulting in a probability vector for each word, representing its relative co-appearances with its surrounding neighbours.

A key hypothesis prevalent in word embeddings, is that *interrelated words should tend to appear in "similar" contexts*. Consequently, the respective probability vectors of related words should bear some similarity. Namely, they should be "close" to one another under some norm (or divergence metric).

Be that as it may, the full probability vectors for each word are too large to be used as an efficient embedding (their dimension is equal to the size of the vocabulary, which can easily exceed several hundred thousands). Therefore, some dimensionality reduction must first be performed, while still aiming to preserve the relative "closeness" of the original words' vectors.

Random Projections and the Johnson Lindenstrauss Lemma

Since most words only appear with a small fraction of the vocabulary in any given context, co-occurrence matrices tend to be *highly sparse*. Therefore, when computing the co-occurrence matrix, we opted for a sparse matrix representation,[3] allowing us to both efficiently update and compute operations on the matrix, without ever keeping the full representation in memory. In fact, for a decently-sized vocabulary (of 150,000 words), a full matrix in which each word is assigned a 32-bit value, would require approximately 80GB of memory, making it rather impractical.

Using the sparse co-occurrence matrix, our goal is to compute a matrix in a lower dimension, in which the norm of the difference between vectors in preserved. The Johnson-Lindenstrauss Lemma[4] states that, given $0 < \varepsilon < 1$, a set of *m* points in R^N, and $n > \frac{8ln(m)}{\varepsilon}$, there is a linear map $f : R^N \to R^n$ such that for every two points *u,v* in the set:

$$(1-\varepsilon)\cdot||u-v||^2 \leq |f(u)-f(v)|^2 \leq (1+\varepsilon)\cdot||u-v||^2$$

In other words, a linear mapping exists which allows us to reduce the dimension, while only warping the distances within a constant multiplicative factor of the original L2 norms — the exact goal we had in mind.

This linear mapping can be found in polynomial linear time by generating random projections, checking if they are orthogonal projections, and if so applying them and scaling the resulting matrix by a factor in order to restore the diminished distances to their original values. Sparse random projections of the form above can be computed using SciPy's *SpraseRandomProjection.*[5]

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.lil_matrix.html
[4] https://en.wikipedia.org/wiki/Johnson%E2%80%93Lindenstrauss_lemma
[5] http://scikit-learn.org/stable/auto_examples/plot_johnson_lindenstrauss_bound.html

Relying on the Johnson Lindenstrauss Lemma, we decided that given the size of our vocabulary (~150,000), a distortion factor of 0.1 (allowing up to 10% distortion) would result in a decent reduction in the dimension.

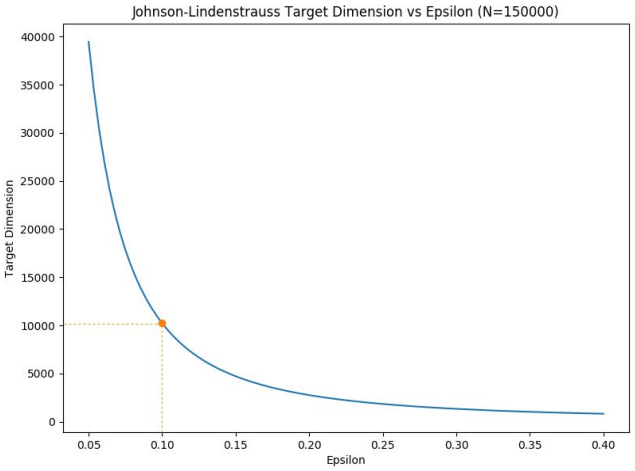

Fig 2: Johnson-Lindenstrauss Target Dimension Plot

While computing the random projection of our co-occurrence matrix was sufficiently efficient, the resulting embedding was lacking in several senses. Firstly, the embedding dimension, while much smaller than the original dimension (by a factor of 15x), was still quite large, thus imposing significant computation limitations when running geometric algorithms on the resulting vectors.

In addition, although the embedding technique indeed clustered "simple" words, such as non-polysemous nouns, in nearby regions of space, it performed more poorly on "complex" words. One reason for this shortcoming might be the fact that the JL-Lemma only guarantees minimal distortion on the L2 norms. However, when computing similarity between probability vectors, the L1 norm (or divergence metrics such as KL-Divergence)[6] is generally preferred. Using the Cauchy-Schwarz inequality,[7] we can deduce that the L1 and L2 norms satisfy that for every vector *x* of dimension *n*:

$$||x||_2 \leq ||x||_1 \leq \sqrt{n} \cdot ||x||_2$$

Therefore, for larger values of *n*, the L2 norm can be much larger than the L1 norm, which may result in suboptimal performance (as the random projection only considers L2 norms).

Principal Component Analysis (PCA)

To overcome the aforementioned shortcomings of the random projection method, we chose to examine the performance of the PCA decomposition on the co-occurrence matrix.

Given an *N x N* matrix, the PCA decomposition allows us to compute the *top-n* principal components. Consequently, we can project the data points onto these components to produce a new matrix, of dimension *N x n.* Note that since PCA cannot be efficiently computed on large sparse matrices, we rely on Scikit-Learn's TruncatedSVD[8] implementation to achieve an approximation of the principal component decomposition.

Geometrically, the PCA decomposition can be thought of as projecting the dataset onto the axes along which the vectors are the most "pronounced". Therefore, the distances between any two vectors in the dataset also decompose primarily along the same axes, implying that for a

[6] https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence
[7] https://en.wikipedia.org/wiki/Cauchy%E2%80%93Schwarz_inequality
[8] http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

sufficiently large number of components, the distances between points should remain approximately preserved.

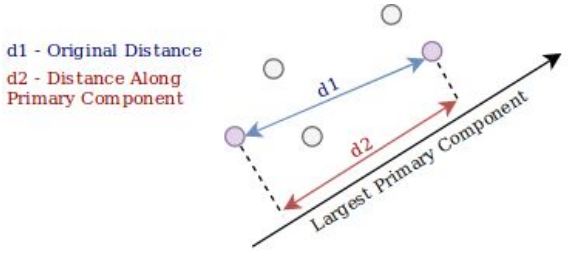

Fig 3: Approximate distance preservation in PCA

Using the above decomposition, we were able to reduce the dimension far further than was possible with the random projection method, without suffering much loss of accuracy. In fact, after experimenting with the target dimension, it appeared that reducing to 300-500 dimensions yielded more accurate results. The fact that such a relatively low dimension performed better than higher-dimension models can likely be attributed to avoiding "overfitting" the data. Moreover, due to the low dimension, we were able to compute our geometric algorithms on the embedding more efficiently, and with greater numerical accuracy.

However, it quickly became apparent that this embedding was still underperforming with regards to the queries we were running on the dataset. As a result, we next chose to use a "hybrid" approach — load a pre-existing highly-performing embedding, and analyse it using the co-occurrences found in our own corpora. Consequently, we downloaded and imported a 300-dimensional GloVe embedding.[9] The performance analyses of our algorithms in the following sections were produced using the above model. The embedding was analysed using the co-occurrences found in the Wacky corpus.[10]

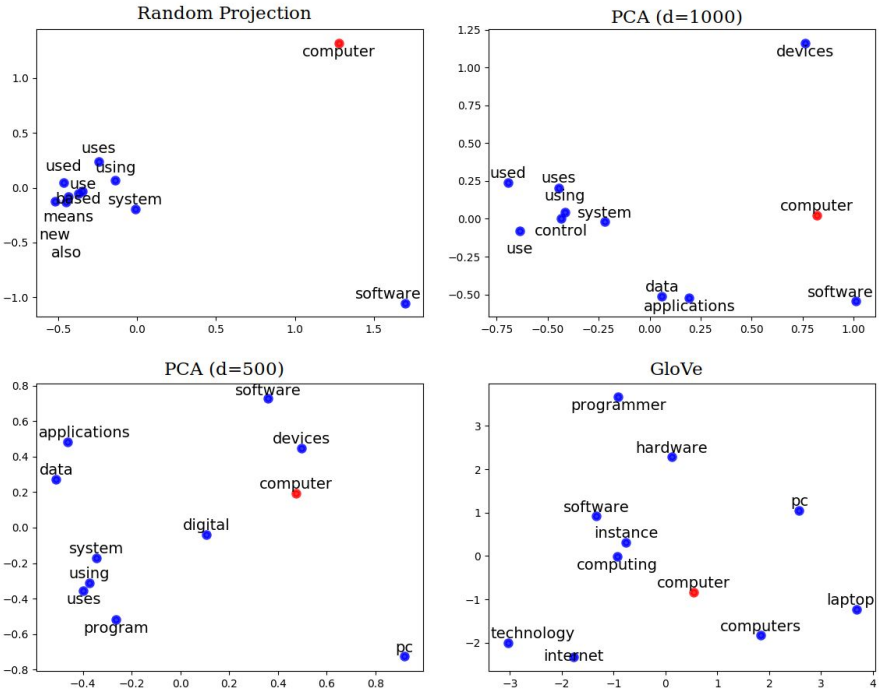

Fig 4: 2D-PCA projection of nearest neighbours of "computer" under each model

[9] Glove.6B - 300-dimension embedding produced over 2014 Wikipedia and Gigawords 5. http://nlp.stanford.edu/data/glove.6B.zip

[10] http://wacky.sslmit.unibo.it/doku.php?id=corpora. We used a randomised subset of 500,000 sentences from the "ukWaC" corpus. The corpus was split into sentences using NLTK's NIST Tokenizer.

In addition, we performed manual analysis of the embeddings by inspecting the closest neighbours for several words. To avoid replicating the entire matrix, we devised a batching algorithm which leverages a constant-size min-heap to efficiently locate the n-closest words. Running the algorithm on each of the aforementioned embedding techniques, we were able to gauge their relative efficiency and precision. A visualisation of the 2D PCA of the 10 nearest neighbours for the word "computer" under each embedding technique, can be seen in Fig 4.

Mathematical Background

Bounding spheres

After running an embedding algorithm, each word is assigned a vector in a high-dimensional space. Our chosen embedding is one that *preserves semantic similarity* of words - that is, the closer together the vectors appear in the embedded space, the more likely they are to be semantically related.

In the same vein, one might choose to bound a *set of points* within the embedding-space using a high-dimensional geometric object. Since the object contains all of the initial points, the volume it encompasses may correspond to"regions" of space which share the semantics of the initial set of points. For example, if all words corresponding to hot beverages lie within the same area in space, then we may naturally assume that a high-dimensional geometric object bounding these points should subsume the category of hot beverages.

However, manipulating and computing the minimal bounding object for a set of high dimensional points is not straightforward. Naturally, we would aim for an object with the *minimal* volume possible, while still making sure that the object and its corresponding properties can be efficiently computed.

As our algorithms (presented in the next sections) require the computation of intersection of pairs of objects, it would be more efficient to restrict ourselves to *convex shapes*. However, many convex shapes are *under specified* within the target domain. For example, computing the convex hull of a set of points requires a set of points with cardinality larger than the embedding dimension. Since we are working with high-dimensional embeddings, this rules out the use of convex hulls. On the other hand, more complex shapes may overcome the problem of under-specification, but they would impose further computational difficulties.

Instead, we chose to work with n-dimensional spheres. We believe spheres strike a balance between computability and accuracy. On the one hand, several algorithms exist for computing the bounding n-dimensional sphere for a set of points. On the other hand, the volume of the sphere will be smaller than that of a bounding n-dimensional hypercube. Furthermore, properties such as the center of mass and breadth of the object follow naturally from its representation (namely, the sphere's center and its radius).

Computing N-Dimensional Spheres

Given a set of vectors, our initial goal is to compute the *minimal n-dimensional ball*[11] which contains all the given vectors. Ritter's algorithm[12] allows for a method of efficiently computing such a ball in linear time (*O(nd)*, where *n* is the number of points, and *d* is the dimension). While this method is efficient, it yields a 5-20% approximation of the minimal bounding ball,[13] making it somewhat inaccurate.

In contrast, Welzl's algorithm is another probabilistic algorithm which runs in expected linear time, but whose approximation results are more robust.[14] The algorithm relies on Seidel's randomised linear programming, and has several efficient implementations. Welzl's algorithm was later further improved by Bernd Gärtner,[15] who implemented a modified version of the algorithm as a library, for which Python bindings exist.[16] As our project was written in Python, we were able to leverage the aforementioned bindings (after porting them from Python 3 to Python 2).[17]

Computing the Intersection Volume of N-Dimensional Balls

Given two n-dimensional balls, we might choose to compute their corresponding intersection volume. However, computing the volume of the intersection of two n-dimensional balls is not straightforward. Up to three dimensions, a closed formula can be found for computing the intersection volume.[18] However, the general problem with n-dimensional balls requires a more careful derivation.

First, there are two cases for which the problem can be "easily" solved. If the two spheres are completely disjoint (i.e., farther apart than the sum of their radii), they do not intersect. Similarly, if one sphere subsumes the other, the intersection volume is that of the smaller ball.

However, when the two spheres intersect without subsumption, the intersection between the two spheres encloses a shape which can be described as two "caps", whose "base" is the hyperplane of the running through the intersection of the spheres.

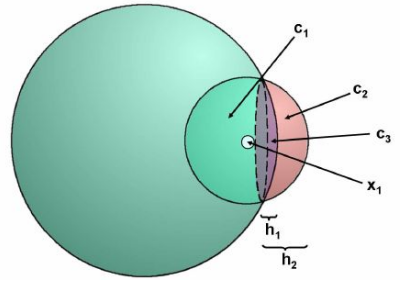

Fig 5: Two intersecting spheres and their separating plane

[11] We use the term "ball" to refer to the space inside a topological sphere, and "sphere" to refer to the surface.
[12] Ritter, J. (1990). An efficient bounding sphere. *Graphics gems*, *1*, 301-303.
[13] Claim attributed to Ritter's paper, see previous footnote.
[14] Welzl, E. (1991). Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science* (pp. 359-370). Springer, Berlin, Heidelberg.
[15] Gärtner, B. (1999, July). Fast and robust smallest enclosing balls. In *European Symposium on Algorithms* (pp. 325-338). Springer, Berlin, Heidelberg.
[16] https://github.com/weddige/miniball
[17] Our modified version can be found in the submitted project zip.
[18] http://mathworld.wolfram.com/Sphere-SphereIntersection.html

The volume of the intersection can be found by summing the volumes of both caps. A clean derivation of this problem was presented in the Asian Journal of Mathematics and Statistics.[19] While the volume of the aforementioned shapes cannot be described with elementary functions, it can be computed using the regularized incomplete beta function, for which programmatic implementations exist.[20]

Using the derivation above, we implemented an algorithm in Python which can efficiently compute the intersection volume of two n-dimensional balls.

Geometric Algorithms for Semantic Analysis

Measuring Context Similarity

Given two words, several similarity measures exist for gauging their relative relatedness. While these measures are useful, they are not immediately extensible into measuring the similarities of *sets* of words. To this end, we propose two of our own algorithms for "context similarity" - measuring how related two sets of words ("contexts"[21]) are to one another.

Both algorithms share a common theme — first, given a context, we compute its minimal bounding ball. Then, with the balls at-hand, we suggest two different metrics for estimating similarity:

1. Computing the inverse of the distance between the two balls:

$$dist - similarity(a,b) = \begin{cases} \frac{1}{\|c_a - c_b\|_2} & c_a \neq c_b \\ \infty & c_a = c_b \end{cases}, \quad [\mathrm{c}_x \text{ is the center of sphere 'x'}]$$

2. Computing the relative intersection volume of the two balls:

$$vol - similarity(a,b) = \frac{V_{a \cap b}}{max(V_a, V_b)}, \quad [\mathrm{V}_x \text{ is the volume of shape 'x'}]$$

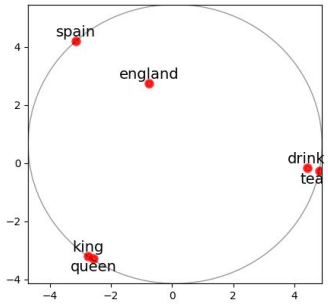

Fig 6: Bounding words within a sphere ("The King and Queen of England drink tea in Spain")

Graphically, the algorithms can be visualised in the following manner:

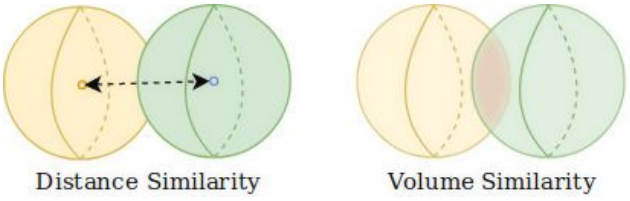

Fig 7: Two geometric similarity measures for contexts

[19] http://docsdrive.com/pdfs/ansinet/ajms/2011/66-70.pdf
[20] https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.special.betainc.html
[21] The context length used in our analysis is similar to that used while constructing the co-occurrence matrix.

Intuitively, if a bounding ball represents the semantic space spanned by a given context, then its center represents the point in space which most aptly describes the same space. Therefore, measuring the distance between those centers should be correlative to semantic similarity. More interestingly, instead of reducing the ball to a single point (and thus "losing" the information about the spanned volume), we can measure the *intersection volume* of the two balls to deduce the amount of "overlap" in the semantic space to which they correspond.

To analyse the performance of both algorithms, we chose a set of sentences in which similar pairs exist, along with dissimilar pairs. We computed the similarity of all pairs in the set (both our metrics are symmetric, so the pairs are order independent). What follows are the results for both similarity algorithms on all pairs in the set:

(1) The king drinks coffee
(2) The queen eats chocolate
(3) The boy walks to the sea
(4) The girl goes to the lake
(5) Wherefore art thou Romeo

	Distance Similarity	**Volume Similarity**
(1) - (2)	0.250208	0.264818
(1) - (3)	0.186819	0.131583
(1) - (4)	0.177125	0.079758
(1) - (5)	0.15986	0.049145
(2) - (3)	0.198117	0.150516
(2) - (4)	0.198803	0.115608
(2) - (5)	0.175752	0.09174
(3) - (4)	0.246778	0.214826
(3) - (5)	0.168377	0.060055
(4) - (5)	0.163874	0.03683

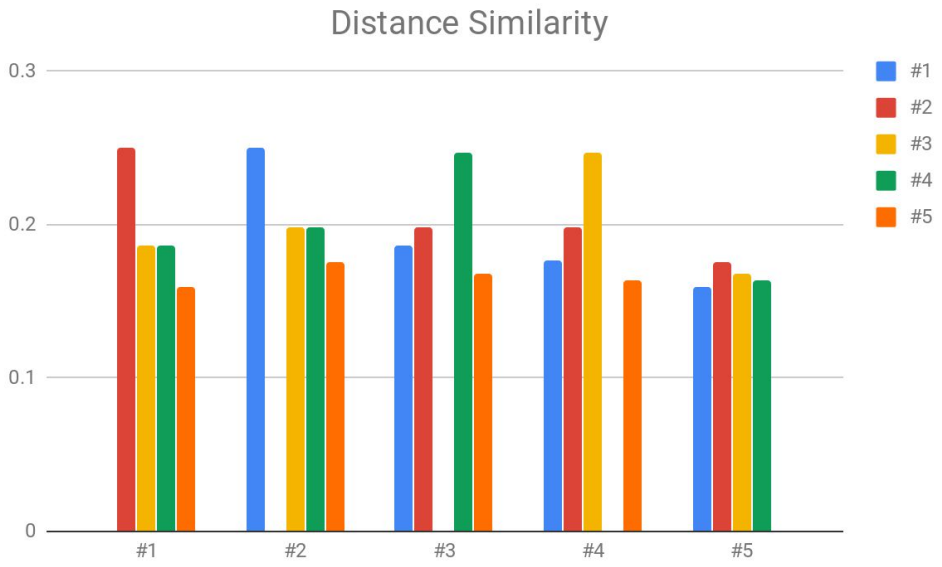

Fig 8: Distance Similarity scores for sentences #1-#5

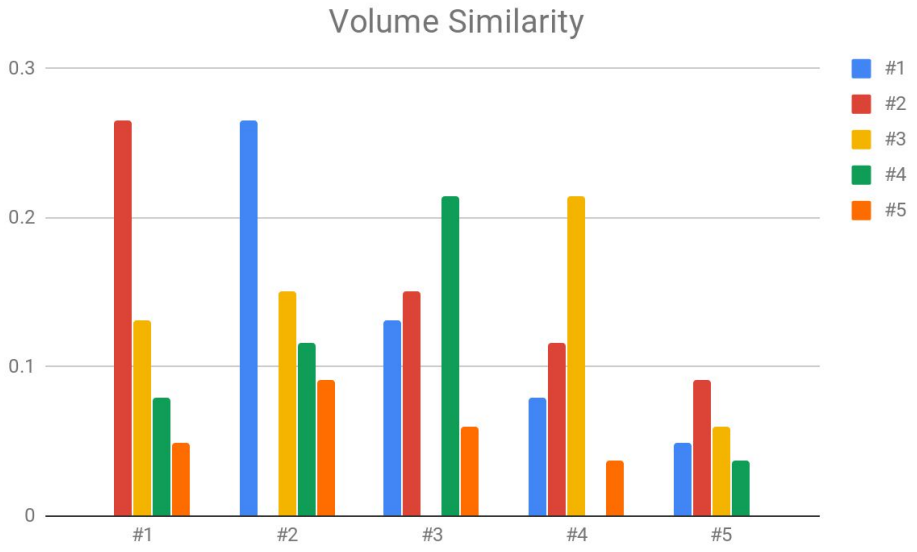

Fig 9: Volume Similarity scores for sentences #1-#5

As can be seen above, both similarity measures successfully identify the corresponding paired sentence for each instance, assigning a higher similarity to the match than to any other sentence. However, the volume similarity seems to perform far better than the distance similarity, in distinguishing between similar and dissimilar instances. Furthermore, the volume similarity correctly assigns a very small similarity score for the last sentence with any of the other sentences. Thus, our new geometric representation seems to provide some added value to the embedding system.

We hypothesise that the qualitative difference between the algorithms can be explained by the abundance of additional information encompassed in the ball volumes — instead of merely using a single point to denote the ball, the volume intersection method allows us to fully leverage the information embodied by the sphere. Concretely, for any region in space that is semantically shared by both contexts, the volume intersection score will significantly increase.

Measuring Context Breadth

In addition to creating a similarity measure, our representation of sets of words provides us with a direct avenue through which we can ascertain the "breadth" of the semantic space spanned by given words. This measure might correlate to the semantic consistency and validity of sentences, as most valid sentences are naturally restricted to a certain "frame of reference". Thus, an overly wide bounding ball for a given context may imply that this sentence is not semantically consistent.

Several classic examples exist for sentences which are grammatically correct, but semantically nonsensical. A prominent example is the following phrase, by Noam Chomsky:

"Colourless green ideas sleep furiously"

To evaluate our hypothesis, we measured the context breadth the first 1000 5-word sentences in our corpus, and compared them to the breadth of the above sentence.

Indeed, the sentence receives an abnormally large context breadth, as compared to other sentences of the same length. Fitting a normal distribution to the resulting histogram (*N(5.07,0.508)*), we get that the probability of finding context breadths larger than or equal to that of Chomsky's phrase is 0.09. To the best of our knowledge, this method is dissimilar to other methods for detecting nonsensical sentences.[22]

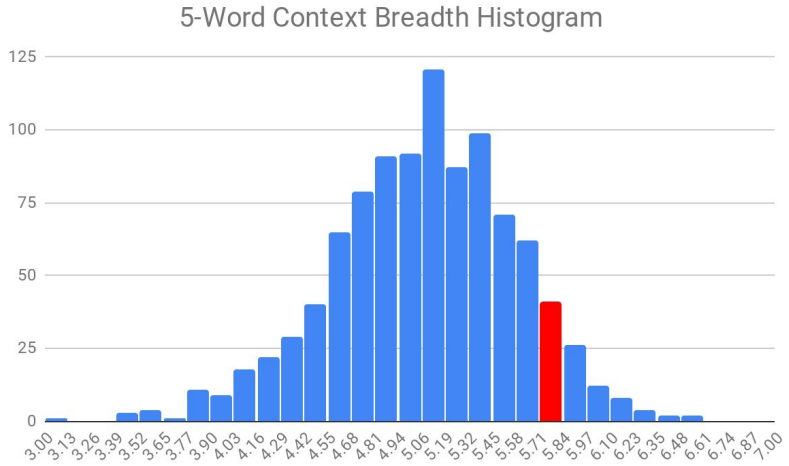

Fig 10: Context breadth histogram on 1000 5-word natural sentences

Furthermore, we manually inspected all sentences in the above set which yielded a breadth larger than or equal to that of Chomsky's phrase, and discovered that the vast majority of them were indeed nonsensical, and not natural sentences. This affirms our belief that the breadth measurement is indicative of semantic consistency.

To measure the statistical validity of the width of the context of natural sentences, we first computed the distribution of the radii of real sentences from the corpus. Then, we took the same

[22] A different statistical approach for detecting nonsensicality sentence can be found at:
Pereira, F. (2000). Formal grammar and information theory: together again?. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, *358*(1769), 1239-1253.

sentences and shuffled the words between them (keeping the length of sentences constant), and measured the distribution for the shuffled sentences.

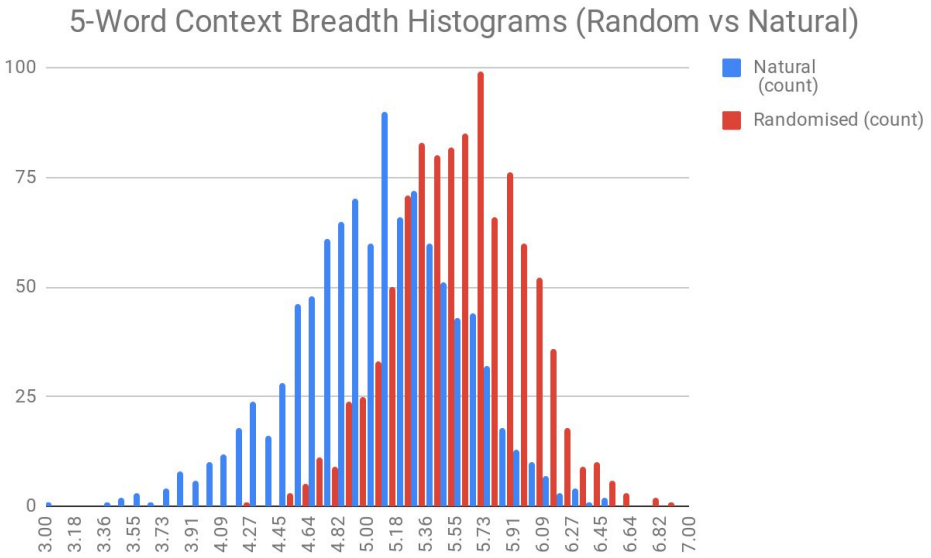

Fig 11: Comparison of context breadth histograms for randomised vs natural sentences

A t-test showed the difference between distributions to be significant, with a p-value $6.2 \cdot 10^{-113}$. Additionally, Chomsky's phrase lies precisely in the largest bucket for the randomised distribution (close to its mean), and therefore ROC analysis[23] on the phrase would have classified the sentence as "random".

Sense Clustering and Polysemy

As we've already seen, in addition to the embedding vector corresponding to a word, one might deduce further information relating to the word by considering the contexts in which the word appears within the corpus. When words are either homographs, polysemous or vague, they have more than a single sense. Intuitively, different meanings of words will appear in different contexts; the further apart the words' meanings are (semantically), the more pronounced the differences between the corresponding contexts.

Previous research suggested a spectrum that consist of homography[24] on the one end, vagueness on the other, with polysemy between the two.[25] In these models, a homograph has two unrelated meanings, whereas a polysemous item has a single meaning with multiple senses, and a vague item has a single core meaning, with ambiguous variations. The notion of a continuum implies that there are no rigid boundaries between classes and therefore we will always find mixed cases. It is clear, therefore, that the closer the word is to being a homograph, the easier it becomes to distinguish between its senses.

Naturally, we may leverage the relation between (prevalent) contexts and the semantic sense, in order to distinguish between appearances of words according to their senses. To this end, we devised the following algorithm:

---

[23] https://en.wikipedia.org/wiki/Receiver_operating_characteristic
[24] https://en.wikipedia.org/wiki/Homonym
[25] Brisard, F., Van Rillaer, G., & Sandra, D. (2001). Processing polysemous, homonymous, and vague adjectives. *AMSTERDAM STUDIES IN THE THEORY AND HISTORY OF LINGUISTIC SCIENCE SERIES 4*, 261-284.

1. For each context in which the word appears, compute the mean vector[26] of the context
2. Cluster the resulting vectors using k-means

In order to gauge the effectiveness of the algorithm above, we ran the algorithm against a set of polysemous words, taken mostly from a pre-existing list.[27] We performed the clustering into two sets[28], and then selected the top three closest contexts to the cluster's center, for each cluster. We then evaluated the number of errors our clustering algorithm performed, on the same set.[29] The following figure presents the error histogram for ten polysemous words.

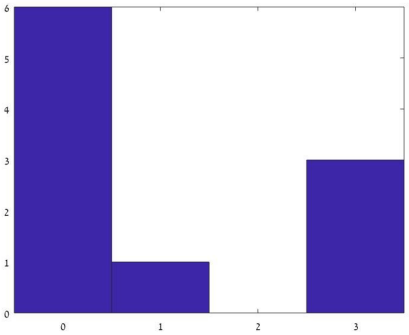

Fig 12: Loss histogram of context clustering on 10 polysemous words

Our algorithm suffered an average loss of 1 on the set above. To analyse the algorithm's performance, we compared it against a completely random algorithm, which simply assigns words to clusters uniformly. After calculating the probabilities for each loss a random algorithm in a single classification task, we programmed a Monte-Carlo simulation for 10 words (similarly to our own experiment), and ran the simulation for 10000 rounds. The proportion of rounds in which the simulation received a loss of less than or equal to our own (empirical p-value) was 0.068.

A rough sense of the clustering can be visualised by projecting the clustered data points onto a 2-D plane, using PCA. One such example, for the word "chair", can be seen in the following figure:

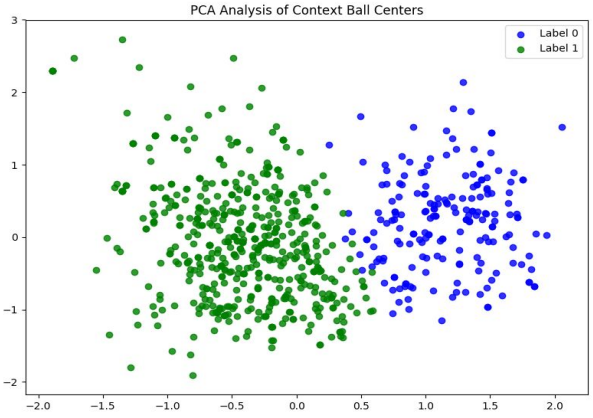

Fig 13: Context clustering visualisation using 2D PCA for the word "chair"

[26] The mean vector for each context is computed without including the target word's vector
[27]We chose words based on the lists in https://goo.gl/UdgqMq. The words were selected manually to avoid evaluating words that rarely appear in more than a single sense.
[28] It should be noted that if we were to a posteriori adjust the numbers of clusters, the performance of the algorithm could be improved.
[29] We assigned a classification error of zero if all chosen sentences were classified correctly. Similarly, error values of one and two represent the presence of more than a single sense within a one, or two clusters, respectively. Lastly, an error value of three represents a case in which all contexts referred to the same sense.

Semantic Containment

Word embedding algorithms provide us with a way to represent a word using a single point in space. While this representation is useful, it fails embody the wealth of information present in the original words' co-occurrences. To address this shortcoming, we suggest an alternate view of words, in which we assign each word with its own *volume* within the embedding space.

To this end, we suggested the following method for associating a volume with a given word:

1. For each context in which the word appears, compute the mean vector of the context
2. Find the minimal enclosing n-dimensional ball for the resulting set of vectors

Using the mechanism above, we can now answer queries regarding the semantics of single words. We decided to explore the aspect of "*containment*" — attempting to measure the extent to which the semantic space related to a given word is contained within the semantic space of another. In order to measure the degree of containment, we simply computed the ratio between the intersection of the volumes corresponding to each word, divided by the volume of the first word, namely:

$$semantic - containment(a,b) = \frac{V_{a \cap b}}{V_a}$$

(where V_x is the volume of the shapes of word 'x')

We hypothesise that this measure will be able to capture the notion of "semantic containment" (for example, relationships such as hypernym-hyponym), since the more "general" a word is (that is, the closer it is to the root of the WordNet tree), the more likely it is to appear in a wider variety of contexts, thus expanding the volume of it's enclosing ball. Thus, more general words should tend to subsume less general words.

To measure the performance the algorithm outlined above, we generated a list of 100 word pairs, in which one word serves as a hypernym, and the other as its corresponding hyponym. We measured the classification error on each pair in the above set, yielding a success rate of 67%. While the results are far from ideal, they are still statistically significant when compared to a random classifier, as the probability of observing a success rate greater than or equal to 67% is:

$$\underset{S \sim Bin(100, \frac{1}{2})}{Pr}[S \geq 67] \leq 0.000205$$

A notable advantage of the measure above lies in its asymmetry, which ties in with the fact that cognitive research discovered that the similarity relation (as measured by human subjects) is not symmetric.

In his seminal paper on similarity, Amos Tversky confirms (and shows examples of) the asymmetric nature of cognitive space.[30] The paper goes on to suggest a class of measurement techniques, based on set theory, using which the similarity between two sets can be estimated. Given two sets *A,B*, he suggests the following calculation:

$$S(a,b) = \frac{f(A \cap B)}{f(A \cap B) + \alpha \cdot f(A \setminus B) + \beta \cdot f(B \setminus A)},\ \alpha, \beta \geq 0$$

[30] Tversky, A. (1977). Features of similarity. *Psychological review*, *84*(4), 327.

Therefore, our containment measure can be viewed as a special case, in which $\alpha = 1, \beta = 0$. Furthermore, the above formulation allow us to compare our model with a few existing cognitive models that Tversky mentions in his article. Most notably, in his "focusing hypothesis" Tversky claims that non-salient objects (words, in our case), are perceived as closer to salient objects ("prototypes") than the inverse. In terms of the model above, this relation would imply that $\alpha, \beta > 0$ and $a > \beta$ (where B is the more salient object).

To check if our model aligned with Tversky's "focusing hypothesis", we generated a list of 15 word pairs, each containing one salient and one non-salient word. We arbitrary chose the parameters $\alpha = 1, \beta = \frac{1}{2}$ (w.l.o.g., as the results are independent of actual parameter values). Our test showed that out of the 15 pairs, only 2 were misclassified. When compared to a random classifier, the probability of observing a success rate greater than or equal to our own is:

$$\underset{S \sim Bin(15, \frac{1}{2})}{Pr}[S \geq 13] \leq 0.0037$$

Conclusions and Further Research

In this work, we explored the use of geometric queries applied to word embeddings. We saw how intuitive reasoning about volumes within the embedding-space can serve as a medium through which deductions can be made regarding the semantics of words. By leveraging geometric queries, we were able to measure similarity between sets of words, compute the "semantic span" of a sentence, cluster polysemous words according to their senses, and even discern whether two words share a hypernym-hyponym relationship (yielding asymmetric relations between words).

Be that as it may, we believe many additional avenues of research regarding geometric queries still remain unexplored. Furthermore, several aspects used in our own algorithms could be expanded upon, perhaps leading to improved results. What follows is a brief analysis of key aspects we've identified as candidates for future research.

Firstly, while the use of n-dimensional balls allowed us to strike a balance between computability and precision, we believe this concept can be further refined by exploring the following ideas:

1. Reduce outlier sensitivity - Bounding spheres are very sensitive to outliers. In this work we relied on the quality of the corpus analysed, but outlier analysis can be performed to remove points prior to computing enclosing balls,[31] thus reducing the algorithms' sensitivity.
2. Refine shape volume - Several other techniques can be used to compute a minimal enclosing volume, including tessellation[32] or using hyper-rectangles.[33] We believe that smaller shapes may more accurately represent the semantic region encompassed by a set of points.
3. Apply gradient to shapes - Our current algorithms assign the same "weight" to all regions of the enclosing shape. However, regions that are closer-by to the bounded words, may theoretically bear more significance. In the same vein, "rare" words might be deemed more informative, and thus increase the "weight" of the regions surrounding them. This additional information can be leveraged by considering the weight of each region when integrating over volume of the intersection of two shapes.

[31] For example, by removing words whose embedding is larger than *m* times the standard deviation.
[32] https://en.wikipedia.org/wiki/Tessellation_(computer_graphics)
[33] https://en.wikipedia.org/wiki/Hyperrectangle

Furthermore, our context clustering and polysemy analysis may be dramatically improved by considering stopwords. We opted to remove stopwords from the embedding since their co-occurrences were very “broad”, thus causing the bounding shapes containing them to encompass nearly the entire embedding space. However, stopwords can carry information pertaining to the part-of-speech of a given word. This information would have allowed the algorithm to more easily distinguish between different senses of a word (as homographs, for example, are likely to be used as different parts of speech).