Mathematical Structures for Word Embeddings

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Computer Science and Engineering by Research

by

Siddharth Bhat
20161105
siddharth.bhat@research.iiit.ac.in

International Institute of Information Technology
(Deemed to be University)()
Hyderabad - 500 032, INDIA
April 2021

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Mathematical Structures for Word Embeddings" by Siddharth Bhat, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. Kannan Srinathan

To #math & #harmless

Acknowledgments

#harmless, #math, ness, celina, mona, dad, mum, ciel, bob, arjun, nitin, alok, projit, tushant, sahiti, kannan, manish, venkatesh, harmless, davean, topos, z-module, drazak, dalcde, antonfire, PlanckWalk, Millennial, sure, Vyn, GodotMisogi. uday, arnaud, tobias, animesh, mahathi, bharti, bharath, athreya, shaanjeet, auro, saumya, lata, ameya, aman, tal, timo, greg, matthias, simione, anastasia, theory group, luca, senpai.

Abstract

With the growing use of natural language processing tools (NLP) in wide-ranging applications, it becomes imperative to understand why NLP models are as successful as they are. In particular, it is essential to understand what mathematical structures precisely underpin these models. To investigate the mathematical models of NLP knowledge representations, we focus on the setting of unsupervised word embeddings, due to the presence of robust models, and a seemingly simple mathematical structure. We find that even in this restricted setting, there are subtle, cross-cutting concerns as to how the model learns — beginning from the high level description of learning a "vector", to the low level details of implementations of these algorithms, including initialization and gradient calculation. We build a theory of knowledge representations for word embeddings, inspired by two seemingly unrelated ideas (a) Montague semantics, a linguistic theory of meaning which ascribes set-theoretic meaning to language, and (b) abstract interpretation, a mathematical theory of creating computable approximations to uncomputable semantics of programs. We find that syntheszing these ideas provide a way to extract fuzzy-set embeddings from existing word-embeddings, which provides the full range of set-theoretic operations (union, intersection, and others), along with probabilistic operations (KL divergence, entropy) which are used to perform polysemy detection and word sense disambiguation, while retaining performance on regular word embeddings tasks such as similarity. Next, we turn our attention to generalizing the word embedding training regime by extracting the geometry which is currently held implicit. This leads us to formulate a general theory of learning word representations on Lie groups. In summary, we provide insight into the mathematical structure of word representations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

NLP has, since its inception, required techniques to extract semantic information from raw corpora with little to no supervision. Some early techniques to perform this were systems which used linear-algebraic ideas such as SVD. This reliance on linear algebra made these algorithms amenable to mathematical analysis. For example, Latent semantic analysis (LSA) is a technique in natural language processing, in particular distributional semantics, of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. LSA assumes that words that are close in meaning will occur in similar pieces of text (the distributional hypothesis). A matrix containing word counts per document (rows represent unique words and columns represent each document) is constructed from a large piece of text and a mathematical technique called singular value decomposition (SVD) is used to reduce the number of rows while preserving the similarity structure among columns. Documents are then compared by taking the cosine of the angle between the two vectors (or the dot product between the normalizations of the two vectors) formed by any two columns. Values close to 1 represent very similar documents while values close to 0 represent very dissimilar documents.1

`Word2vec` is a technique to create unsupervised word embeddings for NLP from unstructured data. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As befitting the name, `word2vec` represents each distinct word with a particular list of numbers called a vector. The vectors are chosen carefully such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors.

A common explanation given fo the success of `word2vec` is The Distributional Hypothesis, which states that words that occur in the same contexts tend to have similar meanings (Harris, 1954). The underlying idea that ”a word is characterized by the company it keeps” was popularized by Firth (1957), and it is implicit in Weaver’s (1955) discussion of word sense disambiguation (originally written as a memorandum, in 1949). The Distributional Hypothesis is the basis for Statistical Semantics. Although the Distributional Hypothesis originated in Linguistics, it is now receiving attention in Cognitive Science (McDonald and Ramscar, 2001). The origin and theoretical basis of the Distributional Hypothesis is discussed by Sahlgren (2008).

On the other hand, it is very unclear why the `word2vec` training regime is as successful as it is. We aim to provide an explanation in two ways. First, we attempt to begin with a *linguistic* theory of meaning, with that of Montague semantics. From here, we borrow the tools of *Abstract Interpretation*, a technique development in computer science for program analysis, which is used to approximate the (formal) meaning/semantics of a program. From this launching point, we ..

Furthermore, we attempt to understand why *vectors* are critical to word embeddings, and if possible, whether we can generalize the notion of a vector embedding to arbitrary spaces.

Concretely, we tackle the following questions:

1. How does `word2vec` *really* work, including an investigation of all the tricks that are used in its reference C implementation and the bearings they have on the quality of the generated word vectors.

2. What is a general theory of meaning that can be adapted to statistical machine learning? Why is abstract interpretation a potential candidate?

3. How can Abstract Interpretation be retrofitted into the `word2vec` unsupervised training regime?

4. What is the largest setting where `word2vec` can be implemented? In particular, how much of the mathematical structure of a vector is exploited by `word2vec` embeddings, and by how much can we generalize this training regime?

Chapter 2

A deep dive into `word2vec`

The popular opinion thanks to a confluence of distributional semantics and machine learning is that systems such as `word2vec` learn ”vector representations”. It is hard to quantify what a vector representation *is*. In particular, it conflates the encoding of the learnt representation (as a tuple of real numbers) with mathematical structure (a vector space). Towards this end, we begin by surveying the algebraic structure of a vector space, and what it would really mean if words were vectors. We then provide a short explanation of the training mechanism of `word2vec`, and discuss why the embeddings that are learnt by systems it and its ilk do not deserve to be called vectors.

Thereafter, we take a step back and start from scratch; what ought the axioms be? Starting from a purely linguistic approach, we are led to conjecture, following the seminal work of montague, that a set-based representation ought to be the best representation possible.

Following an empirical approach, we attempt to generalize the success of `word2vec` and the newly developed theory of contrastive losses, where we conjecture that a Lie group representation ought to be the best representation possible.

We report on experiments performed on both of these representations, always explained through the dual lenses of linguistics and abstract algebra. We hope that this blend of perspectives provides fresh insight into representation learning for NLP.

2.1 `word2vec` in the literature

```
while(1) {
        1. vf = vector of focus word
        2. vc = vector of context word
        3. train such that (vc . vf = 1)
        4. for(0 <= i < negative samples):
```

```
        vneg = vector of word *not* in context
        train such that (vf . vneg = 0)
}
```

The original `word2vec C` implementation does not do what's explained above, and is drastically different. Most serious users of word embeddings, who use embeddings generated from `word2vec` do one of the following things:

- They invoke the original C implementation directly.
- They invoke the `gensim` implementation, which is *transliterated* from the C source to the extent that the variables names are the same.

2.2 **word2vec** in the sources

We analyze the original C implementation of `word2vec`, to point out salient features, curious implementation decisions and bugs in the implementation. The core training loop of `word2vec` (trained using negative sampling, in skip-gram mode) ingests a corpus, and then attempts to learn *two* representations for a given word, called as the positive and negative representation. At a high level, the python pseudocode at Figure 2.1 explains the implementation of `word2vec`.

The `C` implementation in fact maintains *two vectors for each word*, one where it appears as a focus word, and one where it appears as a context word. (Is this sounding familiar? Indeed, it appears that GloVe actually took this idea from ‘word2vec‘, which has never mentioned this fact!)

The setup is incredibly well done in the C code:

- An array called ‘syn0‘ holds the vector embedding of a word when it occurs as a *focus word*. This is random initialized.

```
// tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df663beff438/word2v
for (a = 0; a < vocab_size; a++) for (b = 0; b < layer1_size; b++) {
        next_random = next_random * (unsigned long long)25214903917 + 11;
        syn0[a * layer1_size + b] =
        (((next_random & 0xFFFF) / (real)65536) - 0.5) / layer1_size;
}
```

Another array called `syn1neg` holds the vector of a word when it occurs as a *context word*. This is zero initialized.

```
https://github.com/tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df6
for (a = 0; a < vocab_size; a++) for (b = 0; b < layer1_size; b++)
syn1neg[a * layer1_size + b] = 0;
```

During training (skip-gram, negative sampling, though other cases are also similar), we first pick a focus word. This is held constant throughout the positive and negative sample training. The gradients of the focus vector are accumulated in a buffer, and are applied to the focus word *after* it has been affected by both positive and negative samples.

```
if (negative > 0) for (d = 0; d < negative + 1; d++) {
        // if we are performing negative sampling, in the 1st iteration,
        // pick a word from the context and set the dot product target to
        if (d == 0) {
                target = word;
                label = 1;
        } else {
                // for all other iterations, pick a word randomly and set
                //product target to 0
                next_random = next_random * (unsigned long long)252149039
                target = table[(next_random >> 16) % table_size];
                if (target == 0) target = next_random % (vocab_size - 1)
                if (target == word) continue;
                label = 0;
        }
        l2 = target * layer1_size;
        f = 0;

        // find dot product of original vector with negative sample vecto
        // store in f
        for (c = 0; c < layer1_size; c++) f += syn0[c + l1] * syn1neg[c +

        // set g = sigmoid(f) (roughly, the actual formula is slightly mo
        if (f > MAX_EXP) g = (label - 1) * alpha;
```

```
            else if (f < -MAX_EXP) g = (label - 0) * alpha;
            else g = (label - expTable[(int)((f + MAX_EXP) * (EXP_TABLE_SIZE

            // 1. update the vector syn1neg,
            // 2. DO NOT UPDATE syn0
            // 3. STORE THE syn0 gradient in a temporary buffer neu1e
            for (c = 0; c < layer1_size; c++) neu1e[c] += g * syn1neg[c + l2]
            for (c = 0; c < layer1_size; c++) syn1neg[c + l2] += g * syn0[c +
    }
    // Finally, after all samples, update syn1 from neu1e
    https://github.com/tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df6
    // Learn weights input -> hidden
    for (c = 0; c < layer1_size; c++) syn0[c + l1] += neu1e[c];
```

2.2.1 Initialization choices

We conjecture that since the negative samples come from all over the text and are not really weighed by frequency, you can wind up picking *any word*, and more often than not, *a word whose vector has not been trained much at all*. If this vector actually had a value, then it could move the actually important focus word randomly. The solution is to set all negative samples to zero, so that *only vectors that have occurred somewhat frequently* will affect the representation of another vector. This also explains GloVe's radical choice of having a separate vector for the negative context — they re-used `word2vec` 's scheme, re-interpreting the negative vector and positive vector as accounting for different aspects of co-occurence.

2.2.2 Bugs

- bug in the sigmoid indexing. - bug in the document indexing. -

2.3 The real `word2vec` algorithm

In this section, we outline what we have learnt from delving into the C sources of `word2vec` to provide high-level pseudocode of the algorithm. We discuss the ramifications of the implementation, and the various choices that have been made in the process.

```
def learn(lix: int, rix:int, larr:np.array, rarr:np,array,
    out :float, target: float):
    # gradient descent on loss (target - out)^2 where
    # out = larr[lix] . rarr[rix ]
def train(corpus):
  poss = np.array((VOCABSIZE, DIMSIZE))
  negs = np.array((VOCABSIZE, DIMSIZE))

  for wix in range(len(corpus)):
    w = poss[corpus[wix]]
    l = max(wix-WINDOWSIZE, 0)
    r = min(wix+WINDOWSIZE, len(corpus)-1)

    for xix in range(l, r+1):
        x = poss[xix]
        out = np.dot(x, w)
        learn(lix=wix, rix=xix, larr=poss, rarr=poss, out=out, target=1.0)

    for _ in range(NNEGSAMPLES)
        xix = random.randint(0, len(corpus)-1)
        x = negs[xix]
        out = np.dot(x, w)
        learn(lix=wix, rix=xix, larr=poss, rarr=negs, out=out, target=0.0)
```

Figure 2.1: word2vec pseudocode, explaining the core training algorithm

.

2.4 The output of `word2vec`

The output of the training regime are popularly interpreted as vectors. However, we augment this discussion by bringing in nuance. In particular, we note that:

- Scalar multiples of the same vector represent the same concept as per the testing regime.
- Vector addition has no clear meaning in the representation.
- The identity element (the zero vector) holds no semantic meaning in the representation.
- The reality of the training regime as-implemented makes it unclear as to what mathematical object is being learnt.

The classic explanation of ‘word2vec‘, in skip-gram, with negative sampling, in the paper and countless blog posts on the internet is as follows:

2.5 Conclusion

We learn that the `word2vec` training regime creates two sets of vectors, henceforth referred to as `syn0` and `syn1neg`. The negative vectors `syn1neg` are thrown away, while the positive vectors `syn0` are the ones that are left at the end. The training regime *does not use* gradient descent. Rather, a style of perceptron based update is used.

Chapter 3

Montague Semantics: Interpretations

Montague grammar is an approach to natural language semantics, named after American logician Richard Montague. The Montague grammar is based on mathematical logic, especially higher-order predicate logic and lambda calculus, and makes use of the notions of intensional logic, via Kripke models. Montague pioneered this approach in the 1960s and early 1970s.

Montague held the view that natural language was a formal language very much in the same sense as predicate logic was a formal language. As such, in Montague's view, the study of natural language belonged to mathematics, and not to psychology (Thomason 1974, 2). Montague formulated his views:

> There is in my opinion no important theoretical difference between natural languages and the artificial languages of logicians; indeed I consider it possible to comprehend the syntax and semantics of both kinds of languages with a single natural and mathematically precise theory. (Montague 1970c, 373)

Montague semantics is not interested in a particular situation (e.g. the real world) but in semantical properties of language. When formalizing such properties, reference to a class of models has to be made, and therefore the interpretation of a language will be defined with respect to a set of (suitable) models. For example, in the introduction we mentioned that the characterization of entailment was a basic goal of semantics. That notion is defined as follows. Sentence

An example: Suppose that the meaning of walk, or sing is (for each model in the class) defined as the set of individuals who share respectively the property of (being an individual that is) walking or the property of (being an individual that is) singing. By appealing to the principle of compositionality, if there is a rule that combines these two expressions to the verb phrase walk and sing, there must be a corresponding rule that determines the meaning of that verb phrase. In this case, the resulting meaning will be the intersection of the two sets. Consequently, in all models the meaning of walk and sing is a subset of the meaning of walk. Furthermore, we have a rule that combines the noun phrase John with a verb phrase. The resulting sentence John walks and sings means that John is an element of the set denoted by the verb phrase. Note that in any model in which John is element of the intersection of walkers and singers, he is an element of the set of walkers. So John walks and sings entails John walks.

In Montague 1973, the scope ambiguity is dealt with by providing for the sentence two different derivations. (Non-determinism). On the reading that every has wide scope, the sentence is produced from every man and loves a woman. On the reading that only one woman is involved, the sentence is obtained from Every man loves him is an artifact, a placeholder, or, one might say, a syntactic variable. A special kind of rule, called a 'quantifying-in rule', will replace this him by a noun phrase or a pronoun (in case there are more occurrences of this placeholder). The placeholder corresponds with a logical variable that becomes bound by the semantic counterpart of the quantifying-in rule. For the sentence under discussion, the effect of the application of the quantifying-in rule to a woman and Every man loves him.

For Montague the principle of compositionality did not seem to be a subject of deliberation or discussion, but the only way to proceed. In effect he made compositionality the core part of his 'theory of meaning' (Montague 1970c, 378), which was later summed up in the slogan: 'Syntax is an algebra, semantics is an algebra, and meaning is a homomorphism between them' (Janssen 1983, 25). Yet although Montague used the term 'Frege's functionality principle' for the way in which extension and intension are compositionally intertwined, he did not have a special term for compositionality in general. Later authors, who identified the Principle of Compositionality as a cornerstone of Montague's work, also used the term 'Frege's Principle' (originating with Cresswell 1973, 75); Thomason 1980 is an early source for the term 'compositional'.

Chapter 4

Abstract Interpretation: Semantics

Abstract interpretation is a mathematical theory of approximation of meaning. It was created by Cousout and Cousout to unify many disparate program analysis techniques that had been developed until then. The broad idea is to consider two worlds: a concrete world, which contains "all the meaning", yet which is hopeless non-computable. In parallel, we consider an abstract world, one that is but a mere approximation to the concrete. This loss of information allows the abstract domain be *computable*, thereby providing the ability to approximate concrete meaning. These two worlds are enriched with the structure of a *lattice*, which allows the conjunction and disjunction of meaning; Disjunction to combine facts from disparate sources, such as the `then` and the `else` branch of an `if-then-else`, and conjunction to combine facts, such as the statement `assert(x == "cat" || x == "dog" )` and `assert(x != "cat")` allowing one to deduce `x = "dog"`.

In the most general setting, an abstract interpretation is a manifestation of an *adjunction*, a mathematical structure which captures a notion of "approximate inverses". It will be too much of a detour to explore the vast land of categories and adjunctions. Suffice to say that many of the deep ideas of mathematics arise out of adjunctions. The relationship between algebra and geometry arises out of adjunctions such as Hilbert's Nullstellensatz (.

Here, we shall focus on the special case of an adjunction for lattices, known as a *Galois connection*. We will show how the notion of a Galois connection provides a good notion of "approximation of meaning", and how this directly relates to Montague's ideas of representing semantics as subsets.

4.1 Abstracting away even-ness

Consider a notion of "odd" and even", and how we can generalize this notion to arbitrary subsets of the integers. Let us consider two lattices, $L \equiv 2^{\mathbb{Z}}$, the set of subsets of the integers, and $L^\sharp \equiv \{\bot, O, E, \top\}$, the odd-even lattice. We define two functions, the abstract function $\alpha : L \to L^\sharp$ which abstracts away information about subsets of the integers into `even` and `odd`, and the concretization function

$\gamma : L^\sharp \to L$ which concretizes the idea of even-ness and odd-ness into subsets of the integers. The abstraction function maps the empty subset to $\bot$, since we know nothing about an empty set. It maps sets of integers that are all even to E, since such a set is "even". Similarly, it maps set of integers that are all odd to O, since such a set is "odd". For sets that contain a mixture of both even and odd, it maps these top $\top$ to indicate "failure".

$$\alpha : 2^{\mathbb{Z}} \to L^\sharp; \quad \alpha(S) \equiv \begin{cases} \bot & S = \emptyset \\ E & S \text{ contains only even numbers} \\ O & S \text{ contains only odd numbers} \\ \top & \text{otherwise} \end{cases}$$

The concretization function attempts to make the idea of *even* and *odd* concrete, by sending the concepts $\bot, E, O, \top$ to the sets that best represent them. This is given by:

$$\begin{aligned} &\gamma : L^\sharp \to 2^{\mathbb{Z}} \\ &\gamma(\bot) \equiv \emptyset; \quad \gamma(\top) \equiv \mathbb{Z}; \\ &\gamma(E) \equiv \{2k : k \in\in \mathbb{Z}\}; \quad \gamma(O) \equiv \{2k+1 : k \in \mathbb{Z}\} \end{aligned}$$

That is, γ maps E to the even numbers, O to the set of all odd numbers, $\bot$ to the empty set, and $\top$ to the set of all numbers, since we know nothing. Let's explore the relationship between α and γ. What happens when we consider the compositions $\alpha \circ \gamma$ (read as "α after γ") and $\gamma \circ alpha$?

This leads us to the following relations:

$$\begin{aligned} &C \subseteq (\gamma \circ \alpha)(C) \\ &a = (\alpha \circ \gamma)(a) \end{aligned}$$

These first equation tell us that abstracting a concrete idea C into $\alpha(C)$ and then concretizing the abstract idea using $\gamma(\alpha(C))$ creates a loss of information. This captures the intuition that the abstraction is lossy. The second equation tells us that if we start with an abstract object a, then concretize it with $\gamma(a)$, this loses no information; Bringing it back up to the abstract world with $alpha(\gamma(a)))$ recovers the a.

4.2 Interval domain

A second example is that of the interval domain, where we map subsets of the integers $\not\models^Z$ to *intervals*, The interval domain is a less trivial domain that showcases conjunction and disjunction. For this purpose, we pursue this example as well.

4.3 The formal viewpoint

A partial order L is .. A monotone map is a ... A meet semi-lattice is.. A semi lattice morphism is ... A galois connection between two lattices is ...

4.4 The bridge

Montague semantics wishes to abstract natural language with sets. We believe that what is really going on is that we have a sequence of abstract interpretations, first from the uncomputable semantics of natural language that of the uncomputable montagueian semantics. We conjecture that systems such as `word2vec` themselves compute another layer of abstract interpretation. We attempt to extract evidence for this, and come up non-empty-handed.

Chapter 5

Confluence: Fuzzy set representations

We provide an alternate perspective on word representations, by reinterpreting the dimensions of the vector space of a word embedding as a collection of features. In this reinterpretation, every component of the word vector is normalized against all the word vectors in the vocabulary. This idea now allows us to view each vector as an n-tuple (akin to a fuzzy set), where n is the dimensionality of the word representation and each element represents the probability of the word possessing a feature. Indeed, this representation enables the use fuzzy set theoretic operations, such as union, intersection and difference. Unlike previous attempts, we show that this representation of words provides a notion of similarity which is inherently asymmetric and hence closer to human similarity judgements. We compare the performance of this representation with various benchmarks, and explore some of the unique properties including function word detection, detection of polysemous words, and some insight into the interpretability provided by set theoretic operations.

5.1 Introduction

(TOTO [1] WRONG CITATION) Word embedding is one of the most crucial facets of Natural Language Processing (NLP) research. Most non-contextualized word representations aim to provide a distributional view of lexical semantics, known popularly by the adage *"a word is known by the company it keeps"* `cite: firth1957synopsis`. Popular implementations of word embeddings such as word2vec `cite: mikolov2013efficient` and GloVe `cite: pennington2014glove` aim to represent words as embeddings in a vector space. These embeddings are trained to be oriented such that vectors with higher similarities have higher dot products when normalized. Some of the most common methods of intrinsic evaluation of word embeddings include similarity, analogy and compositionality. While similarity is computed using the notion of dot product, analogy and compositionality use vector addition.

However, distributional representations of words over vector spaces have an inherent lack of interpretablity `cite: goldberg2014word2vec`. Furthermore, due to the symmetric nature of the vector space operations for similarity and analogy, which are far from human similarity judgements `cite: tversky1977features`. Other word representations tried to provide asymmetric notions of similarity in a non-contextualized setting, including Gaussian embeddings `cite: vilnis2014word` and word similarity by dependency `cite: gawron2014improving`. However, these models could not account for the inherent compositionality of word embeddings `cite: mikolov2013distributed`.

Moreover, while work has been done on providing entailment for vector space models by entirely reinterpreting word2vec as an entailment based semantic model `cite: henderson2016vector`, it requires an external notion of compositionality. Finally, word2vec and GloVe, as such, are meaning conflation deficient, meaning that a single word with all its possible meanings is represented by a single vector `cite: camacho2018word`. Sense representation models in non-contextualized representations such as multi-sense skip gram, by performing joint clustering for local word neighbourhood. However, these sense representations are conditioned on non-disambiguated senses in the context and require additional conditioning on the intended senses `cite: li2015multi`.

In this paper, we aim to answer the question: *Can a single word representation mechanism account for lexical similarity and analogy, compositionality, lexical entailment* ***and*** *be used to detect and resolve polysemy?* We find that by performing column-wise normalization of word vectors trained using the word2vec skip-gram negative sampling regime, we can indeed represent all the above characteristics in a single representation. We interpret a column wise normalized word representation. We now treat these representations as fuzzy sets and can therefore use fuzzy set theoretic operations such as union, intersection, difference, etc. while also being able to succinctly use asymmetric notions of similarity such as K-L divergence and cross entropy. Finally, we show that this representation can highlight syntactic features such as function words, use their properties to detect polysemy, and resolve it qualitatively using the inherent compositionality of this representation.

In order to make these experiments and their results observable in general, we have provided the code which can be used to run these operations. The code can be found at `https://github.com/AlokDebnath/fuzzy_embeddings`. The code also has a working command line interface where users can perform qualitative assessments on the set theoretic operations, similarity, analogy and compositionality which are discussed in the paper.

5.2 Related Work

The representation of words using logical paradigms such as fuzzy logic, tensorial representations and other probabilistic approaches have been attempted before. In this section, we uncover some of these representations in detail.

`cite: lee1999measures` introduced measures of distributional similarity to improve the probability estimation for unseen occurrences. The measure of similarity of distributional word clusters was based on multiple measures including Euclidian distance, cosine distance, Jaccard's Coefficient, and asymmetric measures like α-skew divergence.

`cite: bergmair2011monte` used a fuzzy set theoretic view of features associated with word representations. While these features were not adopted from the vector space directly, it presents a unique perspective of entailment chains for reasoning tasks. Their analysis of inference using fuzzy representations provides interpretability in reasoning tasks.

`cite: grefenstette2013towards` presents a tenosrial calculus for word embeddings, which is based on compositional operators *which uses* vector representation of words to create a compositional distributional model of meaning. By providing a category-theoretic framework, the model creates an inherently compositional structure based on distributional word representations. However, they showed that in this framework, quantifiers could not be expressed.

`cite: herbelot2015building` refers to a notion of general formal semantics inferred from a distributional representation by creating relevant ontology based on the existing distribution. This mapping is therefore from a standard distributional model to a set-theoretic model, where dimensions are predicates and weights are generalised quantifiers.

`cite: emerson2016functional, emerson2017semantic` developed functional distributional semantics, which is a probabilistic framework based on model theory. The framework relies on differentiating and learning entities and predicates and their relations, on which Bayesian inference is performed. This representation is inherently compositional, context dependent representation.

5.3 Background: Fuzzy Sets and Fuzzy Logic

In this section, we provide a basic background of fuzzy sets including some fuzzy set operations, reinterpreting sets as tuples in a universe of finite elements and showing some set operations. We also cover the computation of fuzzy entropy as a Bernoulli random variable.

A fuzzy set is defined as a set with probabilistic set membership. Therefore, a fuzzy set is denoted as $A = \{(x, \mu_A(x)), x \in \Omega\}$, where x is an element of set A with a probability $\mu_A(x)$ such that $0 \leq \mu_A \leq 1$, and Ω is the universal set.

If our universe Ω is finite and of cardinality n, our notion of probabilistic set membership is constrained to a maximum n values. Therefore, each fuzzy set A can be represented as an n-tuple, with each member of the tuple $A[i]$ being the probability of the ith member of Ω. We can rewrite a fuzzy set as an n-tuple $A' = (\mu_{A'}(x), \forall x \in \Omega)$, such that $|A'| = |\Omega|$. In this representation, $A[i]$ is the probability of the ith member of the tuple A. We define some common set operations in terms of this representation as follows.

$$(A \cap B)[i] \equiv A[i] \times B[i] \quad \text{(set intersection)}$$
$$(A \cup B)[i] \equiv A[i] + B[i] - A[i] \times B[i] \, \text{(set union)}$$
$$(A \sqcup B)[i] \equiv \max(1, \min(0, A[i] + B[i])) \, \text{(disjoint union)}$$
$$(\neg A)[i] \equiv 1 - A[i] \quad \text{(complement)}$$
$$(A \setminus B)[i] \equiv A[i] - \min(A[i], B[i]) \quad \text{(set difference)}$$
$$(A \subseteq B) \equiv \forall x \in \Omega : \mu_A(x) \leq \mu_B(x) \, \text{(set inclusion)}$$
$$|A| \equiv \sum_{i \in \Omega} \mu_A(i) \quad \text{(cardinality)}$$

The notion of entropy in fuzzy sets is an extrapolation of Shannon entropy from a single variable on the entire set. Formally, the fuzzy entropy of a set S is a measure of the uncertainty of the elements belonging to the set. The possibility of a member x belonging to the set S is a random variable X_i^S which is $true$ with probability (p_i^S) and $false$ with probability $(1 - p_i^S)$. Therefore, X_i^S is a Bernoulli random variable. In order to compute the entropy of a fuzzy set, we sum the entropy values of each X_i^S:

$$\begin{aligned} H(A) &\equiv \sum_i H(X_i^A) \\ &\equiv \sum_i -p_i^A \ln p_i^A - (1 - p_i^A) \ln\Big(1 - p_i^A\Big) \\ &\equiv \sum_i -A[i] \ln A[i] - (1 - A[i]) \ln(1 - A[i]) \end{aligned}$$

This formulation will be useful in section 5.4.4 where we discuss two asymmetric measures of similarity, cross-entropy and K-L divergence, which can be seen as a natural extension of this formulation of fuzzy entropy.

5.4 Representation and Operations

In this section, we use the mathematical formulation above to reinterpret word embeddings. We first show how these word representations are created, then detail the interpretation of each of the set operations with some examples. We also look into some measures of similarity and their formulation in this framework. All examples in this section have been taken using the Google News Negative 300 vectors[1]. We used these gold standard vectors

[1] `https://code.google.com/archive/p/word2vec/`

5.4.1 Constructing the Tuple of Feature Probabilities

We start by converting the skip-gram negative sample word vectors into a tuple of feature probabilities. In order to construct a tuple of features representation in $\mathbb{R}^n$, we consider that the projection of a vector $\vec{v}$ onto a dimension i is a function of its probability of possessing the feature associated with that dimension. We compute the conversion from a word vector to a tuple of features by first exponentiating the projection of each vector along each direction, then averaging it over that feature for the entire vocabulary size, i.e. column-wise.

$$v_{exp}[i] \equiv \exp \vec{v}[i]$$
$$\hat{v}[i] \equiv \frac{v_{\text{exp}}[i]}{\sum_{w \in \text{VOCAB}} \exp w_{\text{exp}}[i]}$$

This normalization then produces a tuple of probabilities associated with each feature (corresponding to the dimensions of $\mathbb{R}^n$).

In line with our discussion from 5.3, this tuple of probabilities is akin to our representation of a fuzzy set. Let us consider the word v, and its corresponding n-dimensional word vector $\vec{v}$. The projection of $\vec{v}$ on a dimension i normalized (as shown above) to be interpreted as *if this dimension i were a property, what is probability that v would possess that property?*

In word2vec, words are distributed in a vector space of a particular dimensionality. Our representation attempts to provide some insight into how the arrangement of vectors provides insight into the properties they share. We do so by considering a function of the projection of a word vector onto a dimension and interpreting as a probability. This allows us an avenue to explore the relation between words in relation to the properties they share. It also allows us access to the entire arsenal of set operations, which are described below in section 5.4.2.

5.4.2 Operations on Feature Probabilities

Now that word vectors can be represented as tuples of feature probabilities, we can apply fuzzy set theoretic operations in order to ascertain the veracity of the implementation. We show qualitative examples of the set operations in this subsection, and the information they capture. Throughout this subsection, we follow the following notation: For any two words $w_1, w_2 \in$ VOCAB, $\hat{w}_1$ and $\hat{w}_2$ represents those words using our representation, while $\vec{w}_1$ and $\vec{w}_2$ are the word2vec vectors of those words.

5.4.2.0.1 Feature Union, Intersection and Difference In section 5.3, we showed the formulation of fuzzy set operations, assuming a finite universe of elements. As we saw in section 5.4.1, considering each dimension as a feature allows us to reinterpret word vectors as tuples of feature probabilities. Therefore, we can use the fuzzy set theoretic operations on this reinterpretation of fuzzy sets. For convenience, these operations have been called feature union, intersection and difference.

$\hat{R}$	$\vec{R}$	$\hat{V}$	$\vec{V}$	$\hat{R} \cup \hat{V}$
risen	cashew	wavelengths	yellowish	flower
capita	risen	ultraviolet	whitish	red
peaked	soared	purple	aquamarine	stripes
declined	acuff	infrared	roans	flowers
increased	rafters	yellowish	bluish	green
rises	equalled	pigment	greenish	garlands

Table 5.1: An example of feature union. `Rose` is represented by R and `Violet` by V. We see here that while the word rose and violet have different meanings and senses, the union $R \cup V$ captures the sense of the flower as well as of colours, which are the senses common to these two words. We list words closest to the given word in the table. Closeness measured by cosine similarity for word2vec and cross-entropy-similarity for our vectors.

Intuitively, the feature intersection of words $\hat{w}_1$ and $\hat{w}_2$ should give us that word $\hat{w}_{1\cap 2}$ which has the features common between the two words; an example of which is given in table 5.1. Similarly, the feature union $\hat{w}_{1\cup 2} \simeq \hat{w}_1 \cup \hat{w}_2$ which has the properties of both the words, normalized for those properties which are common between the two, and feature difference $\hat{w}_{1\setminus 2} \simeq \hat{w}_1 \setminus \hat{w}_2$ is that word which is similar to w_1 without the features of w_2. Examples of feature intersection and feature difference are shown in table 5.2 and 5.3 respectively.

While feature union does not seem to have a word2vec analogue, we consider that feature intersection is analogous to vector addition, and feature difference as analogous to vector difference.

5.4.2.0.2 Feature Inclusion Feature inclusion is based on the subset relation of fuzzy sets. We aim to capture feature inclusion by determining if there exist two words w_1 and w_2 such that *all* the feature probabilities of $\hat{w}_1$ are less than that of $\hat{w}_2$, then $\hat{w}_2 \subseteq \hat{w}_1$. We find that feature inclusion is closely linked to hyponymy, which we will show in 5.6.3.

5.4.3 Interpreting Entropy

For a word represented using a tuple of feature probabilities, the notion of entropy is strongly tied to the notion of certainty `cite: xuecheng1992entropy`, i.e. with what certainty does this word possess or not possess this set of features? Formally, the fuzzy entropy of a set S is a measure of the uncertainty of elements belonging to the set. The possibility a member x_i belonging to S is a random variable X_i^S, which is `true` with probability p_i^S, `false` with probability $(1 - p_i^S)$. Thus, X_i^S is a Bernoulli random variable. So, to measure the fuzzy entropy of a set, we add up the entropy values of each of the X_i^S `cite: mackay2003information`.

Intuitively, words with the highest entropy are those which have features which are equally likely to belong to them and to their complement, i.e. $\forall i \in \Omega, A[i] \simeq 1 - A[i]$. So words with high fuzzy entropy

$\hat{C}$	$\hat{P}$	$\hat{C} \cap \hat{P}$
hardware	vested	cpu
graphics	purchasing	hardware
multitasking	capita	powerpc
console	exercise	machine
firewire	parity	multitasking
mainframe	veto	microcode
$\vec{C}$	$\vec{P}$	$\vec{C} + \vec{P}$
bioses	centralize	expandability
scummvm	veto	writable
hardware	decembrist	cpcs
imovie	exercised	reconfigure
writable	redistribution	backplane
console	devolving	oem

Table 5.2: An example of feature intersection with the possible word2vec analogue (vector addition). The word `computer` is represented by C and `power` by P. Note that power is also a decent example of polysemy, and we see that in the context of computers, the connotations of hardware and the CPU are the most accessible. We list words closest to the given word in the table. Closeness measured by cosine similarity for word2vec and cross-entropy-similarity for our vectors.

can occur only in two scenarios: (1) The words occur with very low frequency so their random initialization remained, or (2) The words occur around so many different word groups that their corresponding fuzzy sets have some probability of possessing most of the features.

Therefore, our representation of words as tuples of features can be used to isolate function words better than the more commonly considered notion of simply using frequency, as it identifies the information theoretic distribution of features based on the context the function word occurs in. Table 5.4 provides the top 15 function words by entropy, and the correspodingly ranked words by frequency. We see that frequency is clearly not a good enough measure to identify function words.

5.4.4 Similarity Measures

One of the most important notions in presenting a distributional word representation is its ability to capture similarity `cite: van2006finding`. Since we use and modify vector based word representations, we aim to preserve the ”distribution” of the vector embeddings, while providing a more robust interpretation of similarity measures. With respect to similarity, we make two strong claims:

$\hat{F}$	$\hat{B}$	$\hat{F} \setminus \hat{B}$
french	isles	communaut
english	colonial	aise
france	subcontinent	langue
german	cinema	monet
spanish	boer	dictionnaire
british	canadians	gascon
$\vec{F}$	$\vec{B}$	$\vec{F} - \vec{B}$
french	scottish	ranjit
english	american	privatised
france	thatcherism	tardis
german	netherlands	molloy
spanish	hillier	isaacs
british	cukcs	raj

Table 5.3: An example of feature difference, along with a possible word2vec analogue (vector difference). `French` is represented by F and `British` by B. We see here that set difference capture french words from the dataset, while there does not seem to be any such correlation in the vector difference. We list words closest to the given word in the table. Closeness measured by cosine similarity for word2vec and cross-entropy-similarity for our vectors.

1. Representing words as a tuple of feature probabilities lends us an inherent notion of similarity. Feature difference provides this notion, as it estimates the difference between two words along each feature probability.

2. Our representation allows for an easy adoption of known similarity measures such as K-L divergence and cross-entropy.

Note that feature difference (based on fuzzy set difference), K-L divergence and cross-entropy are all asymmetric measures of similarity. As `cite: nematzadeh2017evaluating` points out, human similarity judgements are inherently asymmetric in nature. We would like to point out that while most methods of introducing asymmetric similarity measures in word2vec account for both the focus and context vector `cite: asr2018querying` and provide the asymmetry by querying on this combination of focus and context representations of each word. Our representation, on the other hand, uses only the focus representations (which are a part of the word representations used for downstream task as well as any other intrinsic evaluation), and still provides an innately asymmetric notion of similarity.

and	the	in	one	which	to	however	*two*	for	*eight*
this	of	of	in	the	*zero*	to	is	a	for
as	and	only	a	also	*nine*	it	as	but	*s*

Table 5.4: On the left: Top 15 words with highest entropy with frequency ≥ 100 (note that all of them are function words). On the right: Top 15 words with the highest frequency. The non-function words have been emphasized for comparison.

Example 1	$D(ganges \mid\mid delta)$	6.3105
	$D(delta \mid\mid ganges)$	6.3040
Example 2	$D(north \cap korea \mid\mid china)$	1.02923
	$D(china \mid\mid north \cap korea)$	10.60665

Table 5.5: Examples of KL-divergence as an asymmetric measure of similarity. Lower is closer. We see here that the evaluation of North Korea as a concept being closer to China than vice versa can be observed by the use of K-L Divergence on column-wise normalization.1

5.4.4.0.1 K-L Divergence From a fuzzy set perspective, we measure similarity as an overlap of features. For this purpose, we exploit the notion of fuzzy information theory by comparing how close the probability distributions of the similar words are using a standard measure, Kullback-Leibler (K-L) divergence. K-L divergence is an asymmetric measure of similarity.

The K-L divergence of a distribution P from another distribution Q is defined in terms of loss of compression. Given data d which follows distribution P, the extra bits need to store it under the false assumption that the data d follows distribution Q is the K-L divergence between the distributions P and Q. In the fuzzy case, we can compute the KL divergence as:

$$D(S \mid\mid T) \equiv D\left(X_i^S \,\middle|\middle|\, X_i^T\right) = \sum_i p_i^S \log\left(p_i^S / p_i^T\right)$$

We see in table 5.5 some qualitative examples of how K-L divergence shows the relation between two words (or phrases when composed using feature intersection as in the case of `north korea`). We exemplify `cite: nematzadeh2017evaluating`'s human annotator judgement of the distance between China and North Korea, where human annotators considered “North Korea” to be very similar to “China,” while the reverse relationship was rated as significantly less strong (“China” is not very similar to ”North Korea”).

5.4.4.0.2 Cross Entropy We also calculate the cross entropy between two words, as it can be used to determine the entropy associated with the similarity between two words. Ideally, by determining the

$\hat{N}$	$\hat{M}$	$\hat{G}$	$\hat{N} \cap \hat{M}$	$\hat{N} \cap \hat{G}$
nobility	metal	bad	fusible	good
isotope	fusible	manners	unreactive	dharma
fujwara	ductility	happiness	metalloids	morals
feudal	with	evil	ductility	virtue
clan	alnico	excellent	heavy	righteous
$\vec{N}$	$\vec{M}$	$\vec{G}$	$\vec{N} + \vec{M}$	$\vec{N} + \vec{G}$
noblest	trivalent	bad	fusible	gracious
auctoritas	carbides	natured	metals	virtuous
abies	metallic	humoured	sulfides	believeth
eightfold	corrodes	selfless	finntroll	savages
vojt	alloying	gracious	rhodium	hedonist

Table 5.6: Polysemy of the word `noble`, in the context of the words `good` and `metal`. `noble` is represented by N, `metal` by M and `good` by G. We also provide the word2vec analogues of the same.

"spread" of the similarity of features between two words, we can determine the features that allow two words to be similar, allowing a more interpretable notion of feature-wise relation.

The cross-entropy of two distributions P and Q is a sum of the entropy of P and the K-L divergence between P and Q. In this sense, in captures both the *uncertainty in* P, as well as the distance from P to Q, to give us a general sense of the information theoretic difference between the concepts of P and Q. We use a generalized version of cross-entropy to fuzzy sets `cite: li2015fuzzy`, which is:

$$H(S,T) \equiv \sum_i H(X_i^S) + D(X_i^S \mid\mid X_i^T)$$

Feature representations which on comparison provide high cross entropy imply a more distributed feature space. Therefore, provided the right words to compute cross entropy, it could be possible to extract various features common (or associated) with a large group of words, lending some insight into how a single surface form (and its representation) can capture the distribution associated with different senses. Here, we use cross-entropy as a measure of polysemy, and isolate polysemous words based on context. We provide an example of capturing polysemy using composition by feature intersection in table 5.6.

We can see that the words which are most similar to `noble` are a combination of words from many senses, which provides some perspective into its distribution, . Indeed, it has an entropy value of 6.2765[2].

[2]For reference, the word `the` has an entropy of 6.2934.

Word 1	Word 2	Word 3	word2vec	Our representation
bacteria	tuberculosis	virus	polio	hiv
cold	freezing	hot	evaporates	boiling
ds	nintendo	dreamcast	playstation	sega
pool	billiards	karate	taekwondo	judo

Table 5.7: Examples of analogy compared to the analogy in word2vec. We see here that the comparisons constructed by feature representations are similar to those given by the standard word vectors.

5.4.5 Constructing Analogy

Finally, we construct the notion of analogy in our representation of a word as a tuple of features. Word analogy is usually represented as a problem where given a pairing $(a : b)$, and a prior x, we are asked to compute an unknown word $y_?$ such that $a : b :: x : y_?$. In the vector space model, analogy is computed based on vector distances. We find that this training mechanism does not have a consistent interpretation beyond evaluation. This is because normalization of vectors *performed only during inference, not during training*. Thus, computing analogy in terms of vector distances provides little insight into the distribution of vectors or to the notion of the length of the word vectors, which seems to be essential to analogy computation using vector operations

In using a fuzzy set theoretic representation, vector projections are inherently normalized, making them feature dense. This allows us to compute analogies much better in lower dimension spaces. We consider analogy to be an operation involving union and set difference. Word analogy is computed as follows:

$$
\begin{aligned}
& a : b :: x : y_? \\
& y_? = b - a + x \implies y_? = (b + x) - a \\
& y = (b \sqcup x) \setminus a \quad \text{(Set-theoretic interpretation)}
\end{aligned}
$$

Notice that this form of word analogy can be ”derived” from the vector formula by re-arrangement. We use non-disjoint set union so that the common features are not eliminated, but the values are clipped at $(0, 1]$ so that the fuzzy representation is consistent. Analogical reasoning is based on the common features between the word representations, and conflates multiple types of relations such as synonymy, hypernymy and causal relations `cite: chen2017evaluating`. Using fuzzy set theoretic representations, we can also provide a context for the analogy, effectively reconstructing analogous reasoning to account for the type of relation from a lexical semantic perspective.

Some examples of word analogy based are presented in table 5.7.

Dims.	word2vec	Our Representation	
		K-L Divergence	Cross-Entropy
20	0.2478	0.2690	**0.2744**
50	0.2916	0.2966	**0.2981**
100	0.2960	0.3124	**0.3206**
200	0.3259	0.3253	**0.3298**

Table 5.8: Similarity scores on the SimLex-999 dataset `cite: hill2015simlex`, for various dimension sizes (Dims.). The scores are provided according to the Spearman Correlation to incorporate higher precision.

5.5 Interesting Qualitative Observations

5.6 Experiments and Results

In this section, we present our experiments and their results in various domains including similarity, analogy, function word detection, polysemy detection, lexical entailment and compositionality. All the experiments have been conducted on established datasets.

5.6.1 Similarity and Analogy

Similarity and analogy are the most popular intrinsic evaluation mechanisms for word representations `cite: mikolov2013efficient`. Therefore, to evaluate our representations, the first tasks we show are similarity and analogy. For similarity computations, we use the SimLex corpus `cite: hill2015simlex` for training and testing at different dimensions For word analogy, we use the MSR Word Relatedness Test `cite: mikolov2013linguistic`. We compare it to the vector representation of words for different dimensions.

5.6.1.1 Similarity

Our scores our compared to the word2vec scores of similarity using the Spearman rank correlation coefficient `cite: spearman1987proof`, which is a ratio of the covariances and standard deviations of the inputs being compared.

As shown in table 5.8, using our representation, similarity is *slightly* better represented according to the SimLex corpus. We show similarity on both the asymmetric measures of similarity for our representation, K-L divergence as well as cross-entropy. We see that cross-entropy performs better than K-L Divergence. While the similarity scores are generally higher, we see a reduction in the degree of similarity beyond 100 dimension vectors (features).

5.6.1.2 Analogy

For analogy, we see that our model outperforms word2vec at both 50 and 100 dimensions. We see that at lower dimension sizes, our normalized feature representation captures significantly more syntactic and semantic information than its vector counterpart. We conjecture that this can primarily be attributed to the fact that constructing feature probabilities provides more information about the common (and distinct) ”concepts” which are shared between two words.

Since feature representations are inherently fuzzy sets, lower dimension sizes provide a more reliable probability distribution, which becomes more and more sparse as the dimensionality of the vectors increases (i.e. number of features rise). Therefore, we notice that the increase in feature probabilities is a lot more for 50 dimensions than it is for 100.

5.6.2 Function Word Detection

As mentioned in section 5.4.3, we use entropy as a measure of detecting function words for the standard GoogleNews-300 negative sampling dataset[3]. In order to quantitatively evaluate the detection of function words, we choose the top n words in our representation ordered by entropy with a frequency ≥ 100, and compare it to the top n words ordered by frequency from word2vec; n being 15, 30 and 50. We compare the number of function words in both in table 5.10. The list of function words is derived from `cite: nation2016making`.

5.6.3 Compositionality

Finally, we evaluate the compositionality of word embeddings. `cite: mikolov2013distributed` claims that word embeddings in vector spaces possess additive compositionality, i.e. by vector addition, semantic phrases such as compounds can be well represented. We claim that our representation in fact captures the semantics of phrases by performing a literal combination of the features of the head and modifier word, therefore providing a more robust representation of phrases.

We use the English nominal compound phrases from `cite: ramisch-etal-2016-naked`. An initial set of experiments on nominal compounds using word2vec have been done before `cite: cordeiro-etal-2016-predicting`, where it was shown to be a fairly difficult task for modern non-contextual word embeddings. In order to analyse nominal compounds, we adjust our similarity metric to account for asymmetry in the similarity between the head-word and the modifier, and vice versa. We report performance on two metrics, the Spearman correlation `cite: spearman1987proof` and Pearson correlation `cite: pearson1920notes`.

The results are shown in table 5.11. The difference in scores for the Pearson and Spearman rank correlation show that word2vec at higher dimensions better represents the rank of words (by frequency), but at lower dimensions, the feature probability representation has a better analysis of both rank by

[3]https://code.google.com/archive/p/word2vec/

Category		word2vec		Our representation	
		50	100	50	100
Capital Common Countries		21.94	37.55	**39.13**	**47.23**
Capital World		13.02	20.10	**27.30**	**26.54**
Currency		12.24	18.60	**25.27**	**24.90**
City-State		10.38	16.70	**23.24**	**23.51**
Family		10.61	17.34	**23.67**	**23.88**
Adjective-Adverb	Syntactic	4.74	3.23	**7.26**	**3.83**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	9.92	15.68	**21.73**	**21.52**
Opposite	Syntactic	4.06	3.66	**7.61**	**4.92**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	9.36	14.73	**20.60**	**20.26**
Comparative	Syntactic	8.86	12.63	**16.88**	**15.39**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	10.10	15.96	**21.67**	**21.39**
Superlative	Syntactic	7.59	11.30	**14.32**	**13.36**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	9.54	15.20	**20.35**	**20.15**
Present-Participle	Syntactic	7.51	10.96	**14.31**	**13.14**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	9.34	14.73	**19.84**	**19.49**
Nationality	Syntactic	12.51	19.07	**21.64**	**21.96**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	11.51	18.16	**22.71**	**22.97**
Past Tense	Syntactic	11.65	17.09	**20.43**	**19.76**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	11.16	17.21	**21.96**	**27.72**
Plural	Syntactic	11.76	17.23	**20.53**	**19.89**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	11.26	17.28	**21.90**	**21.64**
Plural Verbs	Syntactic	11.36	16.60	**19.88**	**19.46**
	Semantic	10.61	17.34	**23.67**	**23.88**
	Overall	11.05	16.91	**21.46**	**21.30**

Table 5.9: Comparison of Analogies between word2vec and our representation for 50 and 100 dimensions (Dims.). For the first five, only overall accuracy is shown as overall accuracy is the same as semantic accuracy (as there is no syntactic accuracy measure). For all the others, we present, syntactic, semantic and overall accuracy as well. We see here that we outperform word2vec on every single metric.

top n words	**word2vec**	**Our Representation**
15	10	**15**
30	21	**30**
50	39	**47**

Table 5.10: Function word detection using entropy (in our representation) and by frequency in word2vec. We see that we consistently detect more function words than word2vec, based on the 176 function word list released by `cite: nation2016making`. The metric is *number of words*, i.e. the number of words chosen by frequency for word2vec and entropy for our representation

Dims.	Metric	word2vec	Our Representation
50	Spearman	0.3946	**0.4117**
	Pearson	0.4058	**0.4081**
100	Spearman	0.4646	**0.4912**
	Pearson	0.4457	**0.4803**
200	Spearman	0.4479	**0.4549**
	Pearson	**0.4163**	0.4091

Table 5.11: Results for compositionality of word embeddings for nominal compounds for various dimensions (Dims.). We see that almost across the board, we perform better, however, for the Pearson correlation metric, at 200 dimensions, we find that word2vec has a better representation of rank by frequency for nominal compounds.

frequency, and its correlation with similarity of words with a nominal compound. Despite this, we show a higher Spearman correlation coefficient at 200 dimensions as well, as we capture non-linear relations.

5.6.4 Dimensionality Analysis and Feature Representations

In this subsection, we provide some interpretation of the results above, and examine the effect of scaling dimensions to the feature representation. As seen here, the evaluation has been done on smaller dimension sizes of 50 and 100, and we see that our representation can be used for a slightly larger range of tasks from the perspective of intrinsic evaluations. However, the results of quantitative analogy for higher dimensions have been observed to be lower for fuzzy representations rather than the word2vec negative-sampling word vectors.

We see that the representation we propose does not scale well as dimensions increase. This is because our representation relies on the distribution of probability mass per feature (dimension) across all the words. Therefore, increasing the dimensionality of the word vectors used makes the representation that much more sparse.

5.7 Conclusion

In this paper, we presented a reinterpretation of distributional semantics. We performed a column-wise normalization on word vectors, such that each value in this normalized representation represented the probability of the word possessing a feature that corresponded to each dimension. This provides us a representation of each word as a tuple of feature probabilities. We find that this representation can be seen as a fuzzy set, with each probability being the function of the projection of the original word vector on a dimension.

Considering word vectors as fuzzy sets allows us access to set operations such as union, intersection and difference. In our modification, these operations provide the product, disjoint sum and difference of the word representations, feature wise. Using qualitative examples, we show that our representation naturally captures an asymmetric notion of similarity using feature difference, from which known asymmetric measures can be easily constructed, such as Cross Entropy and K-L Divergence.

We qualitatively show how our model accounts for polysemy, while showing quantitative proofs of our representation's performance at lower dimensions in similarity, analogy, compositionality and function word detection. We hypothesize that lower dimensions are more suited for our representation as sparsity increases with higher dimensions, so the significance of feature probabilities reduces. This sparsity causes a diffusion of the probabilities across multiple features.

Through this work, we aim to provide some insights into interpreting word representations by showing one possible perspective and explanation of the lengths and projections of word embeddings in the vector space. These feature representations can be adapted for basic neural models, allowing the use of feature based representations at lower dimensions for downstream tasks.

Chapter 6

The geometry of word represenatations

```
def analogy(a, b, c):
  len_ab = length_of_shortest_geodesic(a, b)
  va = velocity_of_shortest_geodesic(a, b)
  va_at_c = parallel_transport(vec=va, from=a, to=c)
  d = follow_geodesic_along(from=c, vec=va_at_c, len=len_ab)
  return d
```

Embedding words in vector spaces has become the norm for distributional lexical semantic representations. While esoteric literature has attempted to change the underlying embedding space, the resultant word embeddings are specific to a given task or linguistic property. In this paper, we hypothesize that word representations need not change the embedding space, rather the object they are embedded as. We address the notion of a context vector introduced in the skip-gram, CBOW and GloVe models, and test our hypothesis by analyzing word embeddings in the symplectic manifold, where each word is represented by a 2-dimensional pair of (position, momentum), as proposed by Hamiltonian mechanics. We further generalize our representation to embedding words in a Grassmanian space $\mathbf{Gr}(p, V)$, where each word is a p-dimensional subspace of a larger n dimensional vector space V, and similarity and analogy are computed exploiting the Lie theoretic structure of the Grassmanian; its exponential map, retraction, and parallel transport of geodesics. Finally, we provide rigorous empirical evidence and theoretical insight into our embeddings' ability to capture sense information and latent syntactic characteristics, improving the expressivity of the overall representation.

Word representation is a central challenge in natural language processing. The terms *word vectors* and *word embeddings* synonymous to one another, due to the success of vector based models such as word2vec, GloVe and FastText. Even with the introduction of contextualized word representations, the underlying structure of word representations has remained the same, mapping a word to a single vector in a continuous vector space. Some of the known challenges with this assumption of word to vector have been based around resolving polysemy and the treatment of function words, to which the classical answer has been extracting and using contextual information.

6.0.1 The blessing and curse of $\mathbb{R}^n$

Let us consider the fundamental operations performed with `word2vec` :

- Similarity: Given two words, find how similar they are in meaning. This is expressed as a dot product between the two word vectors.

- Analogy: Given three words representing a sense of analogy `(king : man :: woman)`, find the unknown word `x` which represents the analogy `king : man :: woman : x`. This is computed as $x \equiv \texttt{man} - \texttt{king} + \texttt{woman}$.

Let us see how we use the vector space structure for these operations. For similarity, we use the dot product for a notion of similarity. For analogy, we use the vectorial addition and subtraction operations.

However, linguistically speaking, there is a difference between *words* such as `man`, `woman`, and *analogies*, such as `king : man`, `woman : queen`. How ought we interpret this with respect to the mathematics?

Towards capturing this difference between words and anaalogies mathematically, we rewrite the analogy as $(\texttt{man} - \texttt{king}) + \texttt{woman}$. This is suggestive: it suggests that an analogy is a subtraction of vectors of the form $(\texttt{man} - \texttt{king})$, while a word is just a vector such as `woman`. In $\mathbb{R}^n$, the subtraction of vectors continues to be a vector.

Hence, the distinction between points such as c and directions such as $(b - a)$ is a distinction without a (pun intended) difference, since both of these continue to be vectors. This no longer holds in curved spaces. Directions or paths are curved, and words are points of the space.

This distinction *does* make a difference for more complicated spaces. In particular, we choose to examine the example of a sphere versus the plane $\mathbb{R}^2$:

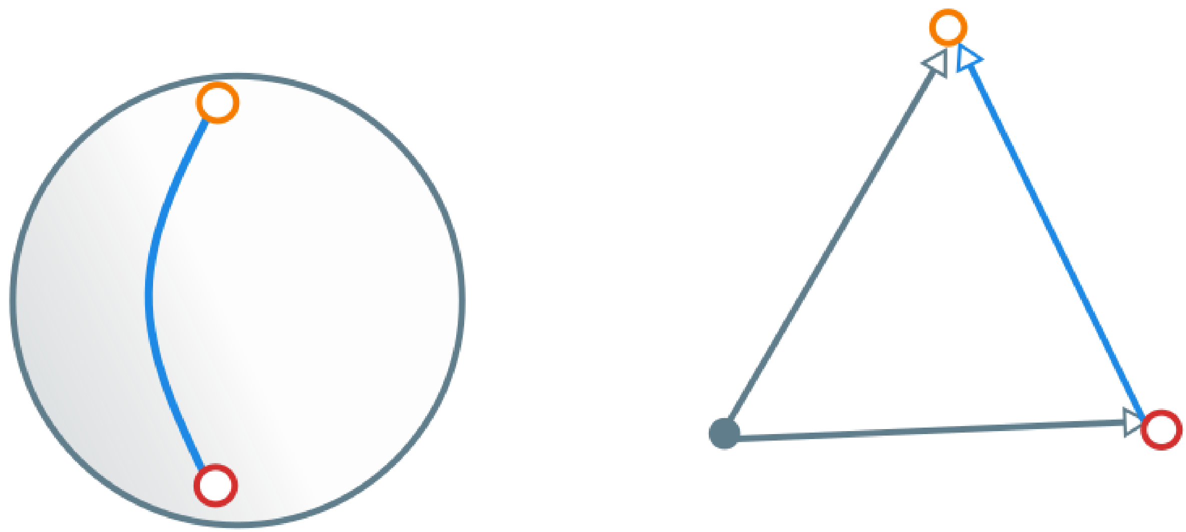

Figure 6.1: $(b - a)$ on a sphere versus the plane ($\mathbb{R}^2$). Red and orange are the words b, a. The path in blue for the sphere and the line in blue for the plane corresponds to the difference $(b - a)$. See how in the case of the plane, both the points as well as the difference can be represented by vectors. However, in the case of a sphere, this is not possible since there is no clean correspondence between a "path" and a point on the sphere.

6.0.2 Generalizing `word2vec` : Distinction with differences

Let us reformulate the fundamental operations performed with `word2vec` , in light of our discussion above.

- Similarity: Given two words, find how similar they are in meaning. This is expressed as a dot product between the two vectors.
- Analogy: Given two words, representing a sense of analogy `(king : man)`, then a third word, representing the base of another analogous pair `(woman : ?)`, find the unknown word `?`

In our description of analogy, we deliberately describe the analogy operation in two steps: first to provide an analogy `(king : man)`, and second to provide the base of a new analogy `(woman : ? )`. As we shall see, the mathematical generalization to Lie groups naturally contains this distinction; Words such as `man, king, woman` become *points* on the manifold, while analogies such as `(king : man)` become *paths* on the manifold. This is not so strange, as this is exactly what happens in `word2vec` . We have words such as `king, man, woman` which are vectors (elements of $\mathbb{R}^n$). On performing analogy `(a : b :: c : ?)`, we perform the vector space operation $b - a + c$, which can be rearranged as $c + (b - a)$, which is the base point c to which we add the direction $(b - a)$.

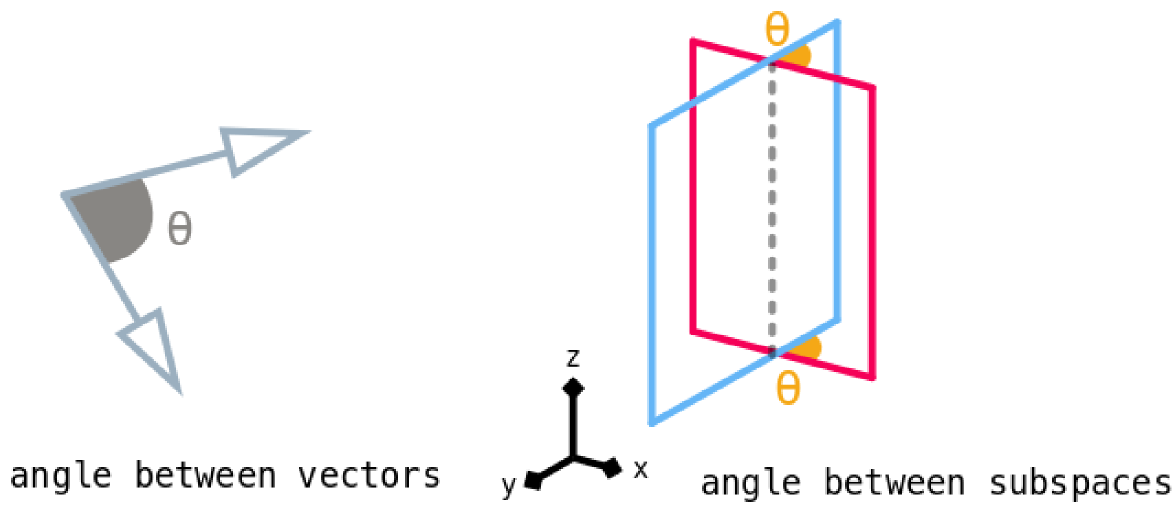

Figure 6.2: Angle between subspaces. Here, we see that when we consider angle between words, it is only possible to consider a single sense direction. When we consider angles between subspaces, the subspace can capture multiple senses since it can extend into multiple directions. The angle between the subspaces provides us our notion of similarity.

We first explain a modest generalisation of `word2vec` , the symplectic manifold, after which we move on to the full generalisation. In the symplectic manifold, we train pairs of vectors as our embedding representation, and use the angle between their subspaces as the similarity measure.

6.0.3 The Grassmanian: Generalizing `word2vec` to n-Dimensions

The grassmanian represents all k-dimensional subspaces of an n-dimensional vector space. More importantly, the grassmanian is a *lie group*; This means that we can build analogues of the `word2vec` operations for the Grassmanian. Here, we lay out a general theory of similarity and analogy, and use the theory to develop our implementation of the Grassmanian.

6.1 Lie groups, Similarly, and Analogy

A Lie group is a generalization of the situation of the Grassmanian. We have the Lie group itself, which consists of ”locations”. Then, there is a Lie algebra, which consists of ”directions at a location”. This allows us to decompose our notion of analogy as explained before, into two steps: one is to analogize, ()and the other is to transport the analogy to a new word ().

Chapter 7

Optimisation on Lie groups

We now consider manifold optimisation techniques on embedded riemannian manfiolds M, equipped with the metric $g : (p : M) \to T_pM \times T_pM \to \mathbb{R}$. The metric at a point $g(p)$ provides an inner product structure on the point T_pM for a $p \in M$.

where we are optimising a cost function $c : M \to \mathbb{R}$. We presume that we have a diffeomorphism $E : M \to \mathbb{R}^n$ (Embedding) which preserves the metric structure. We will elucidate this notion of preserving the metric structure once we formally define the mapping between tangent spaces. This allows us to treat M as a subspace of $\mathbb{R}^n$.

For any object X defined with respect to the manifold, we define a new object $\overline{X}$, which is the embedded version of X in $\mathbb{R}^n$.

We define $\overline{M} \subset \mathbb{R}^n; \overline{M} \equiv image(E)$. We define $\overline{c} : \overline{M} \subseteq \mathbb{R}^n \to \mathbb{R}; \overline{c} \equiv c \circ E^{-1}$

We then needh two operators, that allows us to project onto the tangent space and the normal space. The tangent space at a point $x_0 \in M$, $\overline{T_{x_0}M} \equiv span(\partial_i E|_{E(x_0)})$. We get an induced mapping of tangent spaces $dE : T_{x_0}M$ and $T_{x_0}\overline{M}$.

we consider the gradient $\overline{\boldsymbol{\nabla}}c : (p : \overline{M}) \to \overline{T_pM}; \overline{\boldsymbol{\nabla}}c \equiv dE\bar{d}c$

The normal space, $\overline{N_{x_0}M}$ is the orthogonal complement of the tangent space, defined as $\overline{N_{x_0}M} \equiv \left\{v \in \mathbb{R}^n \mid \langle v|\overline{T_{x_0}M}\rangle = 0\right\}$. It is often very easy to derive the projection onto the normal space, from whose orthogonal complement we derive the projection of the tangent space.

The final piece that we require is a retraction $R : \mathbb{R}^n \to \overline{M} \subseteq \mathbb{R}^n$. This allows us to project elements of the ambient space that are not on the manifold. The retraction must obey the property $R(p \in \overline{M}) = p$. (TODO: is this correct? Do we need $R(\overline{M}) = \overline{M}$ or is this pointwise?) (what are the other conditions on the retraction? smoothness?)

Given all of this machinery, the algorithm is indeed quite simple.

- $x \in \overline{M} \subseteq \mathbb{R}^n$ is the current point on the manifold as an element of $\mathbb{R}^n$
- Compute $g = \boldsymbol{\nabla}c(x) \in T_x\mathbb{R}^n$ is the gradient with respect to $\mathbb{R}^n$.

- $\overline{g} = P_{T_x} g \in T_x M$ is the projection of the gradient with respect to $\mathbb{R}^n$ onto the tangent space of the manifold.
- $x_{mid} \in \mathbb{R}^n \equiv x + \eta\overline{g}$, a motion along the tangent vector, giving a point in $\mathbb{R}^n$.
- $\overline{x}_{next} : \overline{M} \equiv R(x_{mid})$, the retraction of the motion along the tangent vector, giving a point on the manifold $\overline{M}$.

7.1 Pseudocode

7.2 Expected Outcomes

We would like to understand what happens with our generalized training mechanism. In particular, there are many Lie groups that are amenable to perform training on — the orthogonal and special orthogonal group, the steifel manifold, the grassmanian, and others. It would be interesting to how which of these training mechanisms reproduce the performance of `word2vec`, and to what degree.

Related Publications

Word Embeddings as Tuples of Feature Probabilities: Siddharth Bhat, Alok Debnath, Souvik Banerjee, Manish Shrivastava - Proceedings of the 5th Workshop on Representation Learning for NLP, 2020

Bibliography

[1] P. B. Levy. *Call-by-push-value: A Functional/imperative Synthesis*, volume 2. Springer Science & Business Media, 2012.