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, #haskell, and #harmless

Acknowledgments

To #haskell, #harmless and #math, for teaching me so much more than I could possibly have learnt on my own; To davean for the uninvited talk. To topos for all the category theory. To ekmett for #harmless. To Z-module to showing me the beauty of groups, to drazak for teaching me orbit-stabilizer, to dalcde for correcting my definition of compactness, to antonfire for yelling at me when I didn't understand the totient, to bgamari for infinite patience with dropped projects, to PlanckWalk for the patient explanation of an elliptic PDE, to Millennial for a description of the langlands, to sure for what a sheaf is, to all of you for making a corner of the internet feel like home. To mona, for SpaG, britpick, coffee, and tech support. To adlucem, for dubious frog chasing. To Superty, nitin, and our goddess, for what is lost may be found again. ParticleMania and Itihas, thanks for the many obtuse conversations that shaped what I thought. To Kannan and Manish, and Venkatesh, for giving me the levity to pursue what I wanted. To Uday, for the math, warmth, and opportunities. To Arnaud, for the many conversations. To Tobias, for being a way better advisor than I ever deserved. To Vyn and GodotMisogi, for Math-Utopia. To Luca, Senpai, and LLLB for bake and pancakes. To codelegend, for going along with my incorrect explanations. To panda, for marx, coffee, and the baked potato recipe. To Goose, for all the physics. To Ameya, Animesh, Aman,Athreya, Auro, Shaanjeet, and all the rest of theory group for our shared love. To Timo for mate, Tal for the coffee, Greg for the cherry vodka. To all the folks at IISc, Tweag, ETH, and theory group for the great times. To Alok, for beginning this project in the first place. To mum, for the cats and tea. To dad, for teaching me Squeak.

Abstract

> It is only proper to realize that language is largely a historical accident.
>
> John Von Neumann

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 [23], 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

The sky is the limit, so lets build rockets!

Anonymous

Natural language processing has, since its inception, required techniques to extract semantic information from raw corpora with little to no supervision. To perform such unsupervised learning, many models have been built, based off of the "Distributional Hypothesis", which states that words that occur in the same contexts tend to have similar meanings. The underlying idea that "a word is characterized by the company it keeps" was popularized by Firth [15].

Harris 1954 is citation here

An early technique which exploited distributional semantics was Latent Semantic Analysis (LSA) [25]. LSA analyzes relationships by assuming that words that are close in meaning will occur in similar pieces of text (the distributional hypothesis), and thus matrix containing word counts per document (rows represent unique words and columns represent each document) is constructed from a large piece of text and 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, where angles close to `1` represent very similar documents, and angles close to `0` represent dis-similar documents.

`Word2vec` is a recent technique to create unsupervised word embeddings for NLP from unstructured data, whose informal correctness is based on the distributional hypothesis. 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 vector. The training mechanism yields vectors such that the cosine of the angle between the vectors reflects the the level of semantic similarity between the words represented by those vectors.

citations on bo
word2vec and
rectness

neural networ

Nitpick: it is e
surface form w
is a vector, bec
I would argue

n told we
' in theses, as
ally you're the
e writing it.

On the other hand, it is very unclear why the word2vec training regime is as successful as it is. In comparison to LSA, which has theoretical guarantees since it is based on linear-algebraic ideas such as SVD, the correctness of word2vec is far less clear. In this thesis, we attempt to provide a *linguistically* as well as *mathematically* motivated explanation for the correctness of word2vec. We begin with a deep dive into the word2vec implementation. Following this, we chase a *linguistic* theory of meaning, by considering *Montague semantics* [23]. Developing on this idea, we borrow the tools of *Abstract Interpretation* [**?**], a general framework enabling the construction and the soundness proof of approximated semantics of programs. We recall here its core features, which is used to approximate the (formal) meaning/semantics of a program. A core principle in the Abstract Interpretation theory is that all kinds of semantics can be expressed within the framework of fixpoints of monotonic operators in partially ordered structures. In this thesis, we use ideas from abstract interpretation to propose a scheme to extract *Montagueian semantics* from word2vec, and evaluate the performance of our proposed system. Finally, we generalize word2vec far beyond its original setting. Concretely, we tackle the following questions:

1. How word2vec *really* works, 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. This is answered in section 2.1, where we analyze the word2vec sources and extract insights from this process.

2. What is a general theory of meaning that can be adapted to statistical machine learning? Why is abstract interpretation a potential candidate? This is answered in chapter 3, where we explain Abstract Interpretation, and link this theory to Montague semantics.

3. How can Abstract Interpretation be melded into the word2vec training regime? This is answered in chapter 4, where we create new *fuzzy embeddings* and evaluate their effectiveness.

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? This is answered in chapter 5 where we generalize word2vec to arbitrary Lie groups.

Chapter 2

Background

2.1 A deep dive into `word2vec`

> Understanding an idea meant entangling it so thoroughly with all the other symbols in your mind that it changed the way you thought about everything.
>
> Greg Egan

The popular opinion in the NLP literature is that that systems such as `word2vec` learn vector representations [27], [39]. A word embedding learned as a vector, where words with a similar meaning can be given similar vectors. More specifically, the aim of providing vector representations to words is to understand that words that occur in *similar contexts* should be given "similar" vectors. While `word2vec` was not the first mechanism to think of embedding words in a vector space [4, 6]. However, it is the first which achieved an efficient regime for training and the requisite reduction in dimensionality for storing and evalution, thereby making it a benchmark for modern natural language processing. However, few in-depth analyses of the model and *why* it is accurate in downstream tasks and efficient to train made it an important development in the field.

Hence, this forms as the launching point for our study. We will first describe the popular description of the algorithm as found online, describe gaps in both the online descriptions and in the original paper. Next, we dissect the source code of `word2vec`. Armed with detailed knowledge of the real training mechanism, we lay out the list of open questions in the design of `word2vec`. To answer these questions, we explain the mathematical theory of abstract interpretation and connect this to montague semantics in chapter 3, and we then use this to build fuzzy set representations in chapter 4.

talk about eva
tion, not just t
ing, because fu
had insight int
asymmetric ev

2.1.1 word2vec in the literature

```
Maintain a word -> vector mapping.
while(1) {
  vf = vector corresponding to focus word
  vc = vector corresponding to context word
  train such that (vc . vf = 1)
  for(0 <= i < negative samples):
  vneg = vector of word *not* in context
  train such that (vf . vneg = 0)
}
```

Figure 2.1 word2vec: high-level *incorrect* pseudocode. This is the popular explanation of available on Wikipedia and the TensorFlow page.

The classic explanation of `word2vec`, in skip-gram, with negative sampling is the pseudocode shown in Figure 2.1. The paper is arguably too terse to extract out any pseudocode from, so we thus defer to the explanations available on Wikipedia and the TensorFlow reimplementation.

The original `word2vec C` implementation does not do what is explained above. serious users of word embeddings either (a) directly invoke the original C implementation of `word2vec`, or (b) invoke the `gensim` implementation [40], which is *transliterated* from the C source to the extent that the variables names are the same as that of the C implementation. Hence, we shall read the source code to better understand the training scheme of `word2vec`.

implementa-
ks

2.1.2 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 [1]. This necessitates interleaving high level explanations with direct snippets of C source code; We do not apologize for this. We view this as a feature of this thesis, not a bug, since we deal with details such as the initialization and gradient calculations. 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 **??** explains the implementation of `word2vec`.

t the master
directly?

[1] sources taken from https://github.com/tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df663beff438/word2vec.c

2.1.2.1 Initialization

(a) 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/word2vec.c#L369
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;
}
```

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

```
// tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df663beff438/word2vec.c#L365
for (a = 0; a < vocab_size; a++) for (b = 0; b < layer1_size; b++)
syn1neg[a * layer1_size + b] = 0;
```

2.1.2.2 Training

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. Line number 4 decides whether we are picking a positive sample (`d = 0`) or a negative sample (`d > 0`). If we are picking a positive sample, then we expect at 5 to find the dot-product with the `word` to be `1`. On the other hand, if we are picking a negative sample, then we expect at 10 to pick a random word, and at 13 for the dot-product to be zero. 20 computes the dot-product between the focus word and the picked context word. 23-25 computes the gradient $g \equiv \sigma(\texttt{label} - \texttt{focus} \cdot \texttt{ctx})$. 30-31 performs the backprop of the loss of the focus and context word. Note that the gradient of the focus word is stored into a buffer `neu1e`. This ensures that the gradient of the focus word is held constant throught the positive and negative samples. Finally, on 36, the buffered gradient `neu1e` is propagated into the focus word.

```
1  for (d = 0; d < negative + 1; d++) {
2   // if we are performing negative sampling, in the 1st iteration,
3   // pick a word from the context and set the dot product target to 1
```

```
  if (d == 0) {
    target = word; label = 1;
  } else {
    // for all other iterations, pick a word randomly and set the dot
    //product target to 0
    next_random = next_random * (unsigned long long)25214903917 + 11;
    target = table[(next_random >> 16) % table_size];
    if (target == 0) target = next_random % (vocab_size - 1) + 1;
    if (target == word) continue;
    label = 0;
  }
  l2 = target * layer1_size;
  f = 0;

  // find dot product of original vector with negative sample vector
  // store in f
  for (c = 0; c < layer1_size; c++) f += syn0[c + l1] * syn1neg[c + l2];

  // set g = sigmoid(f) (roughly, the actual formula is slightly more complex)
  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 / MAX_EXP / 2))]) * alpha;

  // 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 + l1];
}
// Finally, after all samples, update syn1 from neu1e
// tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df663beff438/word2vec.c#L541
```

```
35 // Learn weights input -> hidden
36 for (c = 0; c < layer1_size; c++) syn0[c + l1] += neu1e[c];
```

2.1.2.3 Pseudocode Implementation

We outline what we have learnt from delving into the C sources of `word2vec` to provide high-level pseudocode of the algorithm.

```
def learn(lix: int, rix:int, larr:np.array, rarr:np,array,
  target: float):
  """
  gradient descent on
  loss = (target - dot(l, r))^2 where
  l = larr[lix]; r = rarr[rix]
  """
  dot = np.dot(larr[lix], rarr[rix])
  gloss = 2 * (target - out)
  #dloss/dl = 2 * (target - dot(l, r)) r
  #dloss/dr = 2 * (target - dot(l, r)) l
  lgrad =  EPSILON * gloss * rarr[rix]
  rgrad =  EPSILON * gloss * rarr[rix]
  # l -= eps * dloss/dl; r -= eps * dloss/dr
  larr[lix] += EPSILON * gloss * rarr[rix]
  rarr[rix] += EPSILON * gloss * larr[lix]

def train(corpus: list, DIMSIZE: int):
  """
  train word2vec of dimension DIMSIZE
  on the given corpus, which is a list of words
  from the source corpus.
  Example:
  train(["the", "man", "was" "tall", "the", "quick", "brown", "fox"], 20)
  """
```

```
# vocabulary is set of words in corpus
vocab = set(corpus)
VOCABSIZE = len(vocab)
# map each unique word to an index
# for array indexing.
vocab2ix = dict([(word, ix) for (ix, word) in enumerate(corpus)])
# positive and negative sample vectors.
# positive vectors are random initialized
poss = np.rand((VOCABSIZE, DIMSIZE))
# negative vectors are zero initialized
negs = np.zeros((VOCABSIZE, DIMSIZE))

# for every location in the corpus
for wix in range(len(corpus)):
  # find word at location,
  w = vocab2ix[corpus[wix]]
  l = max(wix-WINDOWSIZE, 0)
  r = min(wix+WINDOWSIZE, len(corpus)-1)

  # for every positive sample, ie, word in the range [l, r]
  for w2ix in range(l, r+1):
      w2 = vocab2ix[corpus[w2ix]]
      # learn the positive sample.
      learn(lix=w, rix=w2, larr=poss, rarr=poss, target=1.0)

  # learn negative samples for the word.
  for _ in range(NNEGSAMPLES)
      # pick a random word.
      w2ix = random.randint(0, len(corpus)-1)
```

```
w2 = vocab2ix[corpus[w2ix]]
learn(lix=w, rix=w2, larr=poss, rarr=negs, out=out, target=0.0)
```

2.1.3 Bugs

In this subsection, we list bugs that were discovered by us in the `word2vec` codebase. While these do not impact training significantly, they do point out the dangers of using C as programming language, as well as provides justification for reading the original sources.

2.1.3.1 Epoch resetting

```
1 if (eof || (word_count > train_words / num_threads)) {
2   word_count_actual += word_count - last_word_count; local_iter--;
3   if (local_iter == 0) break;
4   word_count = 0; last_word_count = 0; sentence_length = 0;
5   // eof = 0; /* Missing code! */
6   fseek(fi, file_size / (long long)num_threads * (long long)id,
7      SEEK_SET);
8   continue;
9 }
```

`word2vec` trains on multiple threads in parallel. The thread that reads the final chunk of text does not reset its `eof` (end-of-file) state upon reaching the end of file. The code at line 5 ought to have been present, but is missing. Thus, the final fragment of text is trained for only a single epoch, which causes the word vectors to miss data about the last section of the document. This causes the overall accuracy to be lowered than the potential optima.

2.1.4 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.

Nothing on th
threading issu
The fact that c
ing the numbe

2.1.5 Relationship to GloVe

Global Vectors (GloVe) [39] is a word2vec inspired unsupervised learning algorithm for obtaining vector representations for words. Training is performed on global word-word co-occurrence statistics from a corpus, and the resulting representations containlinear substructures of the word vector space.

The innovation in GloVe to have the training objective of GloVe to learn vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Since the logarithm of a ratio equals the difference of logarithms, this objective associates the logarithm of ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode meaning, this information gets encoded as vector differences as well. During training, the GloVe implementation uses*two vectors for each word*, one where it appears as a focus word, and one where it appears as a context word, much as word2vec does. The source code snippet is taken from the file src/glove.c at the Github repository stanfordnlp/glove. [2]

```
/* Calculate cost, save diff for gradients */
diff = 0;
for (b = 0; b < vector_size; b++) diff += W[b + l1] * W[b + l2]; // dot product of word and contex
diff += W[vector_size + l1] + W[vector_size + l2]; // add separate bias for each word
diff = diff - log(cr.val); // target: log(cr.val)
```

2.1.6 Conclusion

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.

We learn that the word2vec training regime creates two sets of vectors, henceforth referred to as syn0 and syn1neg. Training proceeds by defining a loss function which attempts to bring similar vectors close together by trying to make the vectors point along the same direction

[2] https://github.com/stanfordnlp/GloVe/blob/f921427257e6a1e601bff425ec10048b7eacd98d/src/glove.c#L190-L193

(have dot-product 1), and attempts to push dissimilar vectors far away, by trying to make the vectors orthogonal (have dot product 0). At the end of the training regime, the negative vectors `syn1neg` are thrown away, while the positive vectors `syn0` are the ones that are saved. The training regime uses a variation of gradient descent. The trained vectors are un-normalized.

During evaluation, the vectors are first normalized by length. Next, similarity is performed by performing cosine similarity, while analogy is performed by adding and subtracting vectors.

Thus, this leaves the question open as to what is really being learnt by `word2vec`. We answer this question by hypothesizing that the representation that is learnt by `word2vec` in fact encodes *sets of word senses*, which we extract from `word2vec` by using ideas of abstract interpretation. We explain the ideas of abstract interpretation in chapter 3, and then proceed to use these ideas to build fuzzy set representations in chapter 3. Finally, we propose an abstract version of the `word2vec` training algorithm on any choice of Lie group in chapter 5.

Slogan 1. *word2vec = dot product + gradient descent + vector operations.*

Chapter 3

Abstract Interpretation

> Among the thousand-and-one faces whereby form chooses to reveal itself to us, the one that fascinates me more than any other, and continues to fascinate me, is the structure hidden in mathematical things.
>
> Alexander Grothendieck

Montague grammar is an approach to natural language semantics pioneered by Richard Montague [23]. The Montague grammar is based on mathematical logic. 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. To quote:

> 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 semantics is used to describe the semantic properties of language. Broadly speaking, a sentence is given meaning by constructing the set of all possible worlds where that sentence is true. For a mathematical example, the sentence $\mathfrak{l} \equiv (\forall x \in \mathbb{Z}, x < 2)$ is given meaning by constructing the set $S_{\mathfrak{l}} \equiv \{x \in \mathbb{Z} : x < 2\} = \{1, 0, -1, \dots\}$. A false proposition, such that $\mathfrak{l}' \equiv \forall x \in \mathbb{Z}, x > x + 1$ is given meaning by the *empty set* $S_{\mathfrak{l}'} \equiv \{x \in \mathbb{Z} : x > x + 1\} = \emptyset$. As a linguistic example, the meaning of walk, or sing is defined as the set of individuals who share respectively the property of walking or the property of singining. 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. Thus, the meaning of walk and sing is a subset of the meaning of walk.

Ambiguity is dealt with by constructing sets which capture *all possible meanings*. For example, the sentence "Every man loves a woman" can be ascribed multiple meanings: (a) Every man loves the *same* woman (say, Lilith), or (b) every man loves a *different* woman. This is dealt with by considering the union of *both sets of meanings*. To resolve ambiguity given more context, we intersect the union of both sets of meaning with the correct context.

In general, we see that to associate *syntax*,our strings of text with *semantics*, we *define semantics* to be sets of object which satisfy the word. This is very reminiscent of what happens in logic. For example, in propositional calculus, we can associate the syntactic string $(x \vee y)$, where x and y are variables is associated to *all assignments* of x and y which make the expression true. Hence, we have that the semantics of $x \vee y$ is $\{(x = T, y = T), (x = F, y = T), (x = T, y = F)\}$. See that this is really carrying the semantics of $(x \vee y)$, for if we change the syntactic string $(x \vee y)$ to another semantically equivalent string such as $(x \vee y \vee y)$, the set of values which makes $(x \vee y \vee y)$ true is the same as the set of values which make $(x \vee y)$ true. Thus, it is indeed the case that this notion of meaning is semantic.

A general theory of such associations between syntax and semantics was developed by the programming languages community, who had to assign semantic meaning to the syntactic notion of strings in order to analyze and optimize computer programs. Towards this goal, they developed a powerful framework called *Abstract Interpretation* [10] to not only capture the notion of approximating meaning, but also capture the notion of creating hiearchies of approximations of meaning. This allows one to begin from the entire meaning of a program, which is uncomputable ue to the Halting Program, and create a lossy, approximate, but computable semantics of the program. It was used to unify many disparate program analysis techniques, such as dataflow analysis [24] and type checking [14]. The object that lies at the heart of Abstract Interpretation is the *Galois connection* [11], which is a way to relate two partial orders, such that one of them is an approximation of the other.

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. We exploit Montague semantics as the bridge to connect the distributional laature of `word2vec` with the set-theoretic nature of fuzzy sets. We conjecture that `word2vec` computes a sequence of abstract interpretations, first from the uncomputable semantics of natural language that of the Montagueian semantics, which we extract by exploiting fuzzy set theory. We garner evidence towards this conjecture.

3.0.1 Abstracting Size

Let us consider the concept of size in the framework of montague semantics. A word is associate to all concepts that are attached to it. In particular, if a word carries a connotation of being `large`, its semantics will contain all concepts of being long. For example, the word "sun" will contain concepts such as `{hot, fire, huge, astronomical }` where words such as `huge` and `astronomical` are to do with the concept of being large. Similarly, for a word which denotes something that is `small`, its semantics will contain all concepts related to being small. For example, the word `hair` will contain concepts such as `silky, strand, small, filament, thin`, where concepts such as `small` and `thin` refer to size. For some word that represents *both* large and small, such as `moderate`, the semantics will contain concepts of *both* large and small. Finally, for a word that represents neither, such as `Tea`, the semantics of the word will contain *neither* largeness nor smallness. Thus, amongst all possible concepts, we can focus on the small collection of four possible choices of having a notion of size: (1) being large, (2) being small, (3) being both, and being neither.

To encode this formally, we first consider the collection of all semantics. This is going to be the collection of all subsets of the set of all objects, since the semantics of a word is given by the set of all objects that denote the word. For example, the word `large` is denoted by the set of objects which are large, such as `{nile, train, rope, ... }`, while the word `small` is denoted by the set of objects which are small, such as `{hair, hobbits, dots, ... }`. [1].

The concretization function attempts to make the idea of *long* and *short* concrete, by sending the concepts $\bot, L, S, \top$ to the empty set, the set of meanings of the word `long`, the set of meanings of the word `short`, and to the union of the two sets of meaning.

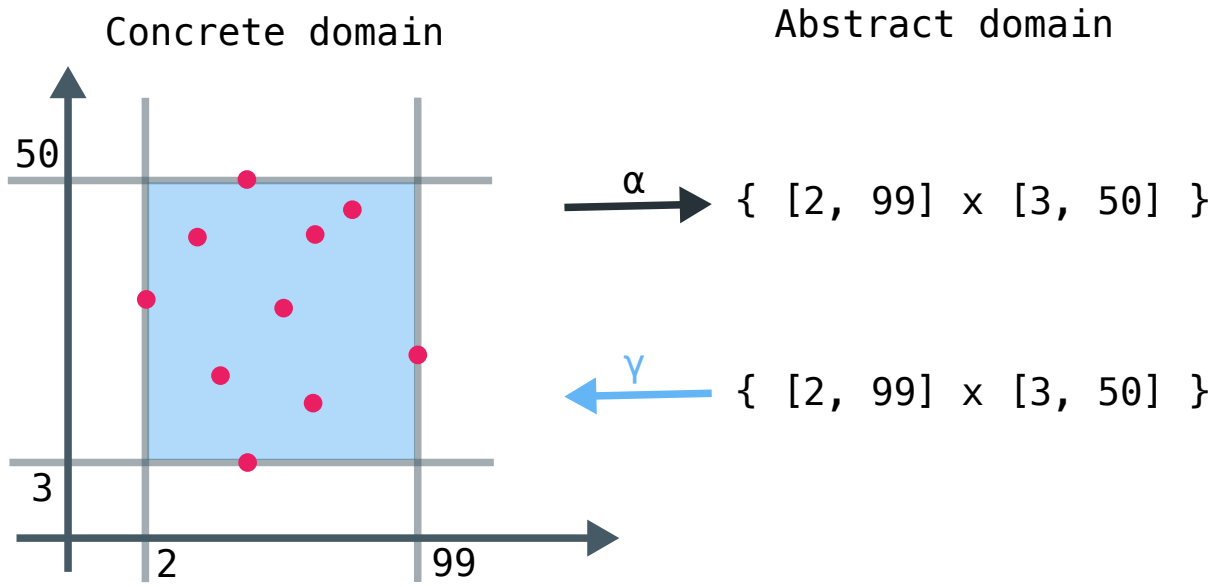
3.0.2 Interval domain

A more mathematical, second example is that of the interval domain, where we map subsets of the plane to rectangles whose axes are aligned with the x and y axes. These can be represented as a pair of intervals, and hence this is known as the *interval domain*. The interval domain allows us to show a mathematical example of abstraction, as well as allowing us to show how to perform abstract interpretations of union and intersection, which is important to develop a compositional theory of meaning.

domain $I \equiv Z \times Z$ represents closed intervals. So an element $(1,3) \in I$ is an abstract representation of the closed interval $[1,3] = \{1,2,3\}$. Similarly, the element $(1,0) \in I$ is an abstract representation of the interval $[1,0] = \emptyset$, as there is no interval whose leftmost point is 1 and rightmost point is 0. In this case, the abstraction function $\alpha : L \to L^\sharp$ goes from a set of integers to the smallest interval containing these numbers. For example, $\alpha(\{1,2,3\}) = (1,3)$,

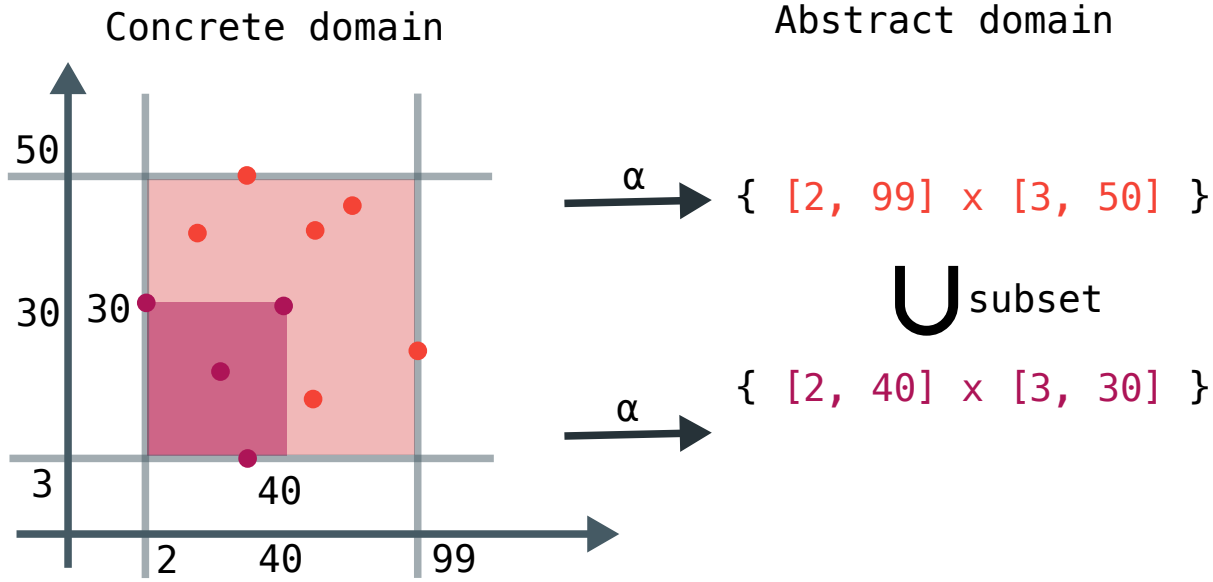
[1] We avoid discussing concerns related to Russell's paradox; These can be obviated by either arbitrarily limiting the sizes of all sets involved, or performing more sophisticated set theoretic constructions

and $\alpha(\{1,5\}) = (1,3,5)$. See that the latter is an approximation: we are representing the triple of numbers 1, 3 and 5 by the *whole interval* $(1,5)$, which we think of as $\{1,2,3,4,5\}$. To make this formal, we define the concretization function $\gamma : L^\sharp \to L$, where $\gamma((l,r)) \equiv \{x \in \mathbb{Z} : l \leqslant x \leqslant r\}$.



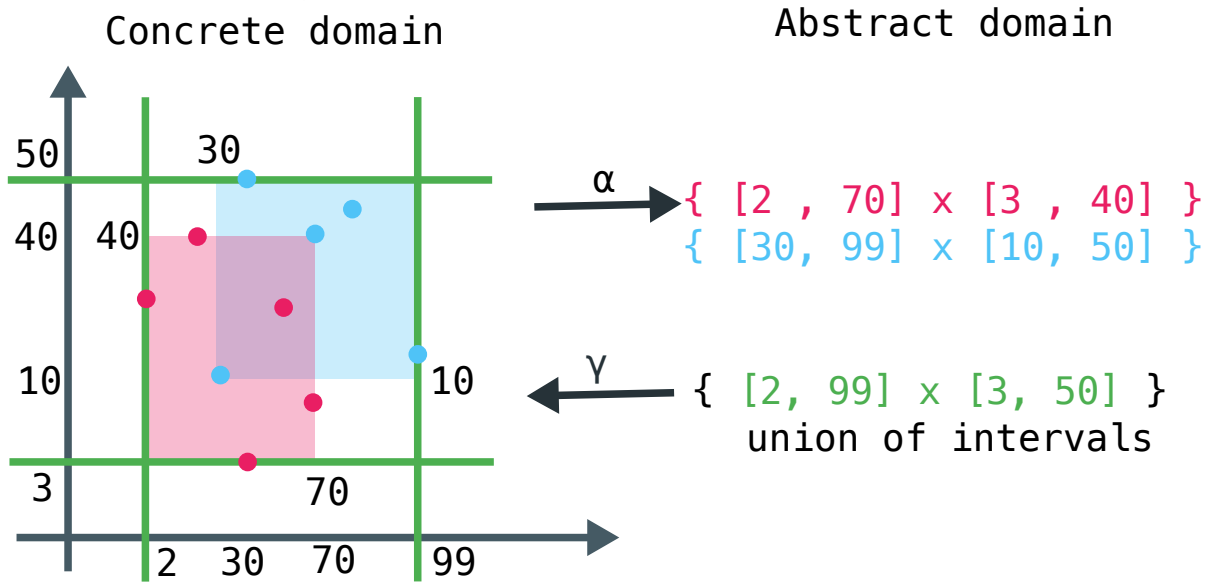
3.0.2.1 Monotonicity in the Interval domain

The process of abstraction is monotonic. If we have a larger collection of points, then we necessarily get an equal or larger interval which bounds these points. The property where more information in the input yields more information in the output is called as *monotonicity*. Formally speaking, given $x \leqslant x$, we are guaranteed that the abstraction α maintains that $\alpha(x) \leqslant \alpha(x')$.



3.0.2.2 Unions in the Interval domain

To define *union* and *intersection* of our intervals, we intuitively want toe *union* of two intervals to be the smallest interval that *contains* them. Similarly, we want the *intersection* of two intervals to be the *largest* interval *contained* in both of them.



This shows that the theory of abstract interpretation is compatible with set operations. In particular, the theory of abstract interpretation is capable of abstracting notions of union and intersection, since the union and intersection of the sets of integers is approximated by joins and meets in the abstract even-odd lattice.

3.0.3 The Galois connection condition

The abstract and the concrete world are linked by the properties:

$$(1) \quad C \subseteq (\gamma \circ \alpha)(C)$$
$$(2) \quad a = (\alpha \circ \gamma)(a)$$

These first equation tell us that abstracting a concrete idea C into $\alpha(C)$ and then concretizing the abstract idea using $\gamma(\alpha(C))$ might lead to 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.

3.0.4 Formalism

Definition 1. *j A partial order is a set* L *equipped with a binary relation* $\leqslant \subseteq L \times L$ *which is (a) reflexive:* $(a \leqslant a)$*, (b) anti-symmetric:* $a \leqslant b \wedge b \leqslant a \implies a = b$*, and (c) transitive:* $a \leqslant b \wedge b \leqslant c \implies a \leqslant c$*.*

Definition 2. *A monotone map is a morphism of partial orders. Formally, it is a function* $f : (L, \leqslant) \to (L^\sharp, \leqslant^\sharp)$ *such that* $a \leqslant b \implies f(a) \leqslant^\sharp f(b)$*.*

Definition 3. *A galois connectio between two lattice* $(L, \leqslant)$ *and* $(L^\sharp, \leqslant^\sharp)$ *is a pair of monotone maps* $\alpha : (L, \leqslant) \to (L^\sharp, \leqslant^\sharp)$*, and* $\gamma : (L^\sharp, \leqslant^\sharp) \to (L, \leqslant)$ *which satisfy the galois connection property:*

$$(1) \quad C \subseteq (\gamma \circ \alpha)(C)$$

$$(2) \quad a = (\alpha \circ \gamma)(a)$$

3.0.5 Conclusion

We learnt that meaning can be (and often is) represented by Galois connections. In particular, the notion of Montague semantics is a Galois connection, which have been exploited by the computer science community to analyze and optimize programs.

Slogan 2. *All meaning is abstract interpretation. This, linguistic meaning better be.*

Chapter 4

Fuzzy set representations

> And if we tamper with our inheritance, so what? What is more ours to tamper with?
>
> Ian M Banks

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.

4.1 Introduction

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"* [15]. Popular implementations of word embeddings such as word2vec [31] and GloVe [39] 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 nor-

malized. 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 [17]. Furthermore, due to the symmetric nature of the vector space operations for similarity and analogy, which are far from human similarity judgements [42]. Other word representations tried to provide asymmetric notions of similarity in a non-contextualized setting, including Gaussian embeddings [44] and word similarity by dependency [16]. However, these models could not account for the inherent compositionality of word embeddings [32].

not needed, al
used in introd

Moreover, while work has been done on providing entailment for vector space models by entirely reinterpreting word2vec as an entailment based semantic model [20], 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 [7]. 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 [28].

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.

4.2 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.

[26] 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.

[5] 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.

[18] 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.

[21] 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.

[12, 13] 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.

4.3 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 $0 \leqslant \mu_A \leqslant 1$, 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.

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

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 $\mathtt{true}$ with probability (p_i^S) and $\mathtt{false}$ 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:

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

This formulation will be useful in section 4.4.4 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.

4.4 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

4.4.1 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.

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

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 4.3, 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 4.4.2.

4.4.2 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 $w_1, w_2 \in \text{VOCAB}$, $\hat{w}_1$ 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.

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

$\hat{R}$	$\vec{R}$	$\hat{V}$	$\vec{V}$	$\hat{R} \cup \hat{V}$
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 4.1 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.

4.4.2.0.1 Feature Union, Intersection and Difference In section 4.3, we showed the formulation of fuzzy set operations, assuming a finite universe of elements. As we saw in section 4.4.1, 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.

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 4.1. 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 4.2 and 4.3 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.

4.4.2.0.2 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 4.6.3.

4.4.3 Interpreting Entropy

For a word represented using a tuple of feature probabilities, the notion of entropy is strongly tied to the notion of certainty [45], i.e. with what certainty does this word possess

$\hat{C}$	$\hat{P}$	$\hat{C} \cap \hat{P}$
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 4.2 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.

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 [30].

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 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

$\hat{F}$	$\hat{B}$	$\hat{F} \setminus \hat{B}$
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 4.3 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.

word occurs in. Table 4.4 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.

4.4.4 Similarity Measures

One of the most important notions in presenting a distributional word representation is its ability to capture similarity [43]. 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:

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.

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 4.4 On the left: Top 15 words with highest entropy with frequency $\geqslant 100$ (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.

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 [36] 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 [3] 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.

4.4.4.0.1 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:

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

We see in table 4.5 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 [36]'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").

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

Table 4.5 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

4.4.4.0.2 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 "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 [29], which is:

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

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 4.6.

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].

4.4.5 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 $(a : b)$,

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

$\hat{N}$	$\hat{M}$	$\hat{G}$	$\hat{N} \cap \hat{M}$	$\hat{N} \cap \hat{G}$
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 4.6 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.

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 rearrangement. 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

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 4.7 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.

reasoning is based on the common features between the word representations, and conflates multiple types of relations such as synonymy, hypernymy and causal relations [8]. 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 4.7.

4.5 Interesting Qualitative Observations

4.6 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.

4.6.1 Similarity and Analogy

Similarity and analogy are the most popular intrinsic evaluation mechanisms for word representations [31]. Therefore, to evaluate our representations, the first tasks we show are similarity and analogy. For similarity computations, we use the SimLex corpus [22] for training and testing at different dimensions For word analogy, we use the MSR Word Relatedness Test [33]. We compare it to the vector representation of words for different dimensions.

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 4.8 Similarity scores on the SimLex-999 dataset [22], for various dimension sizes (Dims.). The scores are provided according to the Spearman Correlation to incorporate higher precision.

4.6.1.1 Similarity

Our scores our compared to the word2vec scores of similarity using the Spearman rank correlation coefficient [41], which is a ratio of the covariances and standard deviations of the inputs being compared.

As shown in table 4.8, 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).

4.6.1.2 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.

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 4.9 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 4.10 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 [35]. The metric is *number of words*, i.e. the number of words chosen by frequency for word2vec and entropy for our representation

4.6.2 Function Word Detection

As mentioned in section 4.4.3, 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 $\geqslant 100$, 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 4.10. The list of function words is derived from [35].

4.6.3 Compositionality

Finally, we evaluate the compositionality of word embeddings. [32] 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 [**?**]. An initial set of experiments on nominal compounds using word2vec have been done before [**?**], 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 [41] and Pearson correlation [37].

The results are shown in table 4.11. 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 frequency, and its correlation with similarity of words with a

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

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 4.11 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.

nominal compound. Despite this, we show a higher Spearman correlation coefficient at 200 dimensions as well, as we capture non-linear relations.

4.6.4 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.

4.7 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 prob-

abilities. 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 5

Geometry of Word Representations

> In the deeper dreams everything was likewise more distinct, and Gillman felt that the twilight abysses around him were those of the fourth dimension.
>
> H. P. Lovecraft

Embedding words in vector spaces has become the norm for word representation. 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. While esoteric literature has attempted to change the underlying embedding space of distributional lexical semantic representations, the resultant word embeddings are specific to a given task or linguistic property.

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 are based around resolving polysemy and the treatment of function words, to which the classical answer has been extracting and using contextual information.

We first survey why we use vectors as the object that is embedded. We focus on the `word2vec` training and testing mechanism. We will see that this process uses many fundamentally different structures that is posessed by real n-dimensional space $\mathbb{R}^n$. We then disentangle these different structures for mathematical cleaniness. Once this is performed, we isolate the mathematical structures that are present, allow us to introduce a generalization based on embedding words into an arbitrary Lie group. We show how specializing this to the case of $\mathbb{R}^n$ yields us back fuzzy set representations, and we conjecture that this applied to richer spaces would yield embeddings with hithertho unexplored properties.

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

In this section, we recall the training regime of `word2vec`, and then disentangle the various structures of $\mathbb{R}^n$ that are used during training and evaluation. The fundamental operations performed in `word2vec` during training is to first pick a focus word, and then performing gradient descent with all the words around the focus word to have dot product $+1$ (positive samples), and performing gtadient descent with random words in the corpus away from the focus word to have dot product 0 (negative samples). We see that to perform this step, we need to know (1a) How to perform dot-products (a measure of similarity), and (1b) How to perform gradient descent (a method of function optimisation).

Next, during evaluation, to check for *similarity* between the two words, we once again use the dot-product of the vectors representing the two words. *Analogy* is more complex; Given two words representing a sense of analogy (`king : man`), and a third word representing the first half on an unknown analogy (`woman:?`), we are to find the unknown word `?` which solves the analogical task `king:man::woman:?`. Recall that this is computed as $? \equiv \texttt{man} - \texttt{king} + \texttt{woman}$. The above equation relies on being able to perform (2) addition ($+$) and subtraction ($-$).

These are all different facets of $\mathbb{R}^n$. (1a) The *inner product* structure is used to compute the dot-product. (1b) The *Riemannian* structure is used to perform gradient descent, and (2) The *group* structure of vectors under vector addition is used to compute the addition and subtraction operations for analogy. With this disentanglement, we elaborate on what these mathematical structures are, after which we write down the general version of the algorithm which can still honestly be called the `word2vec` algorithm. We describe the structures in `word2vec` and their generalization is subsection 5.2.1.

5.2 Spaces for `word2vec`

In light of the discussion above, we reformulate the informally discussed notions into the language of *Lie groups*, which are mathematical structures that are equipped with addition, subtraction, and gradient descent operations. The astute reader will notice that a dot product structure is missing. Fear not, for every Lie group is associated with a corresponding *Lie algebra*, which possesses a dot-product structure, and a method to transport data back and forth from the Lie group to the Lie algebra. The definition of a Lie Group is first presented, followed by the definition of the Lie algebra and the exponential and logarithm maps. Later, the formal definitions of a Riemannian manifold, along with the connection to Lie Groups is presented.

	`word2vec`	Ours
Space	$\mathbb{R}^n$	Lie group $(G, \cdot)$
Similarity `sim(a, b)`	Dot product on $\mathbb{R}^n$	Dot product in the Lie algebra $\mathfrak{g}$
Analogy `a:b::c:?`	Vector space structure on $\mathbb{R}^n$: $(b - a + c)$	Group operation $b \cdot a^1 \cdot c$
Optimisation	Gradient descent	Riemannian gradient descent

Figure 5.1 Operations in `word2vec` and their generalization

5.2.1 Matrix Lie groups

We first provide an accessible definition of Lie groups, sweeping technicalities into the footnotes. We direct the interested reader to [19].

Slogan 3. *A Matrix Lie group is a space of matrcies closed under multiplication and matrix inverse.*

Intuitively, the idea is that we wish to generalize from vectors, where adding words is commutative (since $a + b = b + a$ for vectors), to the non-commutative regime of matrcies where $AB \neq BA$. Hence, we will replace the notion of analogy, $b - a + c$ with matrix multiplication of the form $BA^{-1}C$.

Example 4. $\mathbb{C}^\times \equiv \{z \in \mathbb{C} : |z| = 1\}$, *the collection of complex numbers with magnitude* 1 *forms a Lie Group.*

Example 5. $GL(n, \mathbb{R})$ *is the collection of all* real, *invertible* $n \times n$ *matrices.*

Example 6. $SO(n)$ *is the collection of* $n \times n$ *real matrices with determinant* $+1$

5.2.2 Dot products on the Lie Group?

We do not yet have a way to take dot-products of matrices. Naively, one might assume that given matrices M, N, we could define $\langle M|N\rangle$, the dot product of M and N as $\langle M|N\rangle \equiv \sum_{i,j} M[i,j]N[i,j]$. However, recall that a dot-product must be defined on a vector space, since the dot-product must be *bilinear*. It must obey the axiom that $\langle \lambda M|N\rangle = \langle M|\lambda N\rangle = \lambda\langle M|N\rangle$ for any scalar $\lambda \in \mathbb{R}$. However, a Lie Group does *not* come with a notion of scaling. For example, the number $1 \in \mathbb{C}^\times$, but $2 \cdot 1 = 2 \notin \mathbb{C}^\times$! More precisely, we are saying that the Lie Group G is not a vector space, and thus it does not make sense to impose a dot-product (inner product) structure, since being a vector space pre-supposes the existence of a vector space structure on the Lie Group. This is where the *Lie Algebra* enters into the picture. It gives us a way to get *vectors* from Lie Group elements, such that we can take dot-products of these vectors.

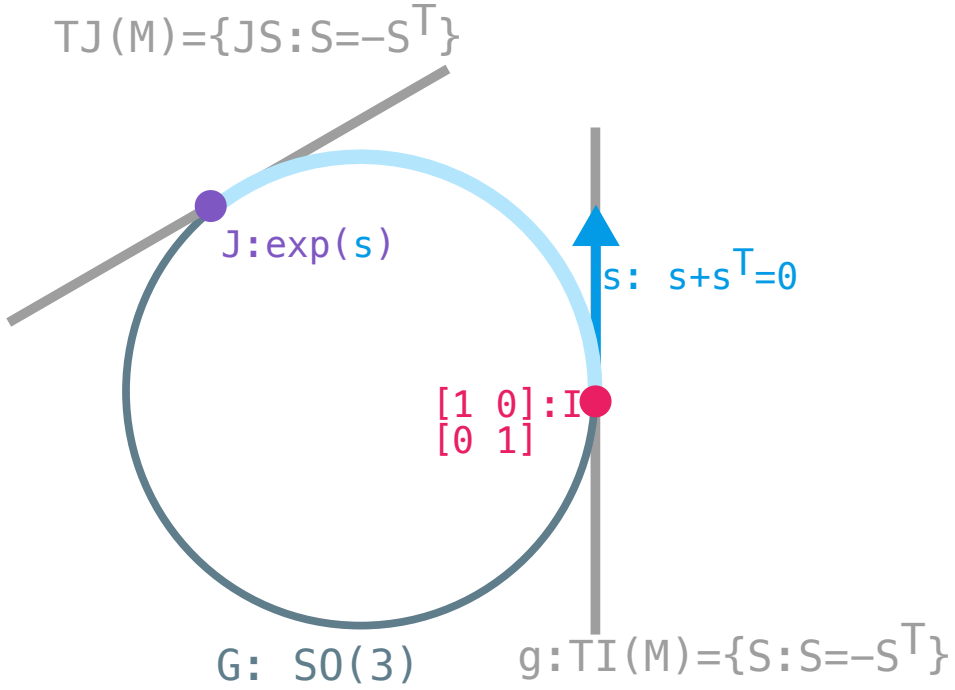


Figure 5.2 A depiction of how to imagine the Lie group $SO(3)$, the group of rotations of 3 dimensional space. The group's elements are abstractly the gray circle. The identity rotation I is denoted in pink. The tangent space at the identity I is denoted by blue; It consists of all skew symmetric matrices S such that $S + S^T = 0$. s is one such skew-symmetric matrix, which lives in the tangent space of the identity. Finally, the exponential map, which map from the lie algebra (the tangent space at the identity I) to the lie group G is shown as the light blue arc. The Lie group element $J \in SO(3)$ (in purple) is the exponential of the Lie algebra element $s \in T_I(SO(3))$.

5.2.3 Matrix Lie Algebras

A Matrix Lie Algebra is a way to convert group elements into vectors. Formally, we perform this by taking a *logarithm* of the group elements, where the logarithm operation is defined by the *matrix logarithm*. It turns out to be the case that the logarithm of a Lie Group is always a Matrix vector space. For example, going back to the example of $\mathbb{C}^\times \equiv \{z : |z| = 1\}$, recall that we can write all such number as $e^{i\theta}$. Thus, when we take the logarithm, we get the set $\{i\theta : \theta \in \mathbb{R}\}$ which is indeed a vector space, because it is closed under addition and scalar multiplication.

5.3 The generalized word2vec training

We will now be imagining our words as *elements of a lie group*. Recall that this means that they are points in some *subspace* of $\mathbb{R}^n$, where this subspace can be curved in shape. Also, the elements are equipped with the notion of a logarithm; That is, they come equipped with a function $\log : G \to \mathfrak{g}$, and $\exp : \mathfrak{g} \to G$, where log maps from the lie group (a group) to its lie algebra (a vector space), and vice versa for the exp map.

5.3.1 Overview

Following the discussion in the preceding section, we provide a compact discussion of the word2vec training algorithm, indicating how to substitute the original word2vec algorithm with its Lie Group counterpart, as well as discussing potential linguistic ramifications of this replacement.

5.3.2 Training

For the purposes of training, we need to describe how to perform the dot-product to calculate the loss, as well as how to perform gradient descent to improve the loss.

5.3.2.1 Dot Product

To compute the dot product between two elements $g, h \in G$, we first logarithmize them, and then take the dot product of their logarithms, which lives in the Lie algebra. Hence, we define the dot product on the Lie group $\langle \cdot, \cdot \rangle$ as $\langle g, h \rangle \equiv \log(g) \cdot \log(h)$.

5.3.2.2 Gradients

Next, to perform gradient descent, we need a notion of a *gradient* to decide in which direction to move, and a way to perform a gradient update. How one performs the gradient update is explained in detail in the . Intuitively, since all our manifolds are subspaces of $\mathbb{R}^n$, we first compute the gradient vector, then travel along the gradient vector (which will take us out of the manifold), and finally "retract back" into the manifold, which gives us a gradient step, starting from a point in the manifold, ending at a point in the manifold.

5.3.3 Evaluation

To perform evaluation, given the trained mapping from words to Lie Group elements as inputs, we must be able to perform similarity and analogy to reach the baseline of word2vec's

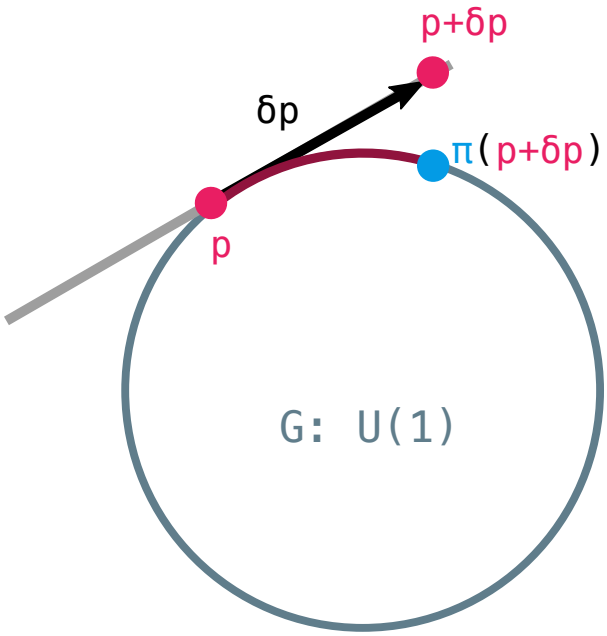


Figure 5.3 gradient descent on a manifold.

usefulness. We indicate other potential uses of the richer structure posessed by Lie Groups after describing the bare minimum structure.

5.3.3.1 Similarity

We compute similarity between two words by computing the dot product between the logarithm of the lie group elements which represent the words. Given two word-matrices h, h', we first map into their Lie Algebra via $\log(h), \log(h')$. This is now the Lie Algebra, which is an inner product space so we then take the dot product $\log(h) \cdot \log(h')$. Linguistically, we are reducing a word to its infinitesimal components, before comparing the words along these infinitesimal senses of meaning.

il xD

5.3.3.2 Analogy

To compute the analogy $a : b :: c :?$, we generalize the vectorial $b - a + c$ into $b \cdot a^{-1} \cdot c$. Linguistically, this provides the same notion of analogy as `word2vec` does, since we are performing a similar operation of cancelling the meaning of a and adding the meaning of c to a base word b.

5.3.4 Sense difference

Recall that for an arbitrary Lie group, the group operation *need not be commutative*. This is in start contrast to the `word2vec` case, where $a + b = b + a$ holds for all vectors. This non-commutativity allows one to define the *commutator* of two elements, $[a, b] \equiv aba^{-1}b^{-1}$, which measures how much a and b fail to commute. If the commutator is the identity element of the group, that is, if $[a, b] = 0$, then the two elements commute, since $[a, b] = aba^{-1}b^{-1} = baa^{-1}b^{-1} = bb^{-1} = e$. On the other hand, if a and b do not commute, then the commutator $[a, b]$ is a non-trivial element of the group. Linguistically speaking, the commutator of two words $[v, w]$ gives us a new word which measures by how much the senses of v and w fail to commute. elaboration

5.4 Formalisms

5.4.1 Riemannian Manifold

A Riemannian manifold M of dimension k *embedded* in a space of dimension n is, intuitively, a surface with k-degrees of freedom, which has been embedded in $\mathbb{R}^n$. For example, a circle has one degree of freedom, the angle θ. This can be embedded in 2D space $\mathbb{R}^2$ via the map $\theta \mapsto (\cos\theta, \sin\theta)$. Similarly, the sphere has two degres of freedom, the azimuthal angle θ and the angle at the azimuth ϕ, and can be mapped into 3D via the map $(\theta, \phi) \mapsto (\cos\theta\cos\phi, \cos\theta\sin\phi, \sin\theta)$.

Here, we assume that the reader is familiar with basic point-set topology [34]. Formally, the Riemannian manifold of dimension k embedded in a space of dimension N is a subset $M \subseteq \mathbb{R}^N$ such that for each $p \in M$, there exists an open set $C \subseteq R^k$ (of parameters), and an an open neighbourhood $O \subseteq \mathbb{R}^n$ of p (a local parmetrization of M around p), and an onto map $x : C \to O \cap M$ (of coordinates), such that (1) x is differentiable, (2) x has a differentiable inverse, and (3) the jacobian of x is one-to-one. [1]

Intuitively, a Riemannian manifold allows us to assign co-ordinates at each local region of the manifold which looks like $\mathbb{R}^N$. For example, the Earth in total is curved, since it is a sphere. However, around each of us, it is approximated by a plane, since the Earth appears flat around a small region. Thus, we can continue to use the differential calculus that we know and love to compute gradients on a Riemannian manifold, which we will use to perform gradient descent on curved spaces.

[1] We provide the extrinsic definition of a Riemannian manifold as a subspace of $\mathbb{R}^n$. A definition without reference to the embedding space is an intrinsic definition, which is provided in standard texts, such as Lee's introduction to smooth manifolds

5.4.2 Tangent space

On every manifold, at each point, we have a *tangent space*, which represents the space of possible directions we can move on the space at that point. For example, at each point on a circle, we can define a velocity that is tangential to the circle. See that this space that is tangential is *one dimensional*, even though the circle is itself two-dimensional. The gradient of a function at a point is a vector which lives in the tangent space a point.

5.4.3 Lie groups

A lie group is a Riemannian manifold M which is also a group, whose group structure is compatible with the manifold structure. This means that we have an associative *group operation* $* : M \times M \to M$, which comes with an inverse $(\cdot)^{-1} : M \to M$, and an identity $e \in M$ such that the group operation $*$ and the inverse $(\cdot)^{-1}$ are continuous with respect to the inherited subspace topology on M.

Intuitively, this gives us the operations (1) $v \cdot w \simeq v + w$ to add vectors, (2) $(v)^{-1} \simeq -v$ to invert a vector, and (3) $v \cdot w^{-1} \simeq v - w$ to subtract vectors. This allows us to generalize analogy to curved spaces, by using lie group operations to perform analogy.

5.4.4 Lie Algberas

The Lie algebra of a Lie group is the *tangent space* of the Lie group *at* the identity element. Informally, the idea is that while the Lie group is a "non-linear" object, the tangent space represents tangents / "linearlizations" of the group. Moreover, in a group, given a tangent vector at the identity, we can *exponentiate* this tangent vector to get a Lie group element.

For example, we consider the Lie group of complex numbers of unit norm, $G \equiv \mathbb{C}^\times = \{z \in \mathbb{C} : |z| = 1\}$. The Lie algebra corresponding to G, denoted by $\mathfrak{g}$ are the real numbers $\mathfrak{g} = \mathbb{R}$. We can *exponentiate* a real number via the map $\exp : \mathfrak{g} \to G$, $\exp(r) \equiv e^{ir}$. Similarly, we have a logarithm map $\log : G \to \mathfrak{g}$, which takes any element $z \in \mathbb{C}^\times$, and produces a real number $\log(z) \equiv \arg(z)$. Notice that the space $\mathfrak{g} = \mathbb{R}$ is a real vector space, since it contains an additive identity 0, while the space $G = C^\times$ is not a real vector space, since it is not closed under scaling by reals.

For a more non-trivial example, we consider the space of all rotations in 3D. These are given by the orthogonal matrices $SO(3) \equiv \{O : OO^T = I\}$. We will not derive the Lie algebra space formally, which can be found in **[?]** [2]. Informally, the idea is that the Lie algebra represents a logarithmization of the tangent space. Thus, we must compute $\log(SO(3)) = \{\log(O) : \log(OO^T) = \log(I)\}$. This leads to the calculation:

[2] A derivation of the tangent space of $SO(n)$ performed by the author can be found at https://math.stackexchange.com/questions/3389983/explicit-description-of-tangent-spaces-of-on

$$\log(SO(3)) = \{\log(O) : \log(OO^T) = \log(I)\}$$
$$\log(SO(3)) = \{\log(O) : \log(O) + \log(O^T) = \log(I)\}$$
$$\log(SO(3)) = \{\log(O) : \log(O) + \log(O)^T = \log(I)\}$$
$$\log(SO(3)) = \{\log(O) : \log(O) + \log(O)^T = 0\}$$
$$\text{Skew} = \{Z : Z + Z^T = 0\}$$

Thus, the tangent space of $SO(3)$ is the space Skew of skew-symmetric matrices. Notice that skew symmetric matrices form a vector space, since the sum of two skew symmetric matrices is skew-symmetric, and skew-symmetric matrices are closed under scaling by the real numbers. On the other hand, $SO(3)$ is not a vector space; For example, the element $I \in SO(3)$ as $II^T = I$. On the other hand, consider $J \equiv 2I$. We have $JJ^T = 4I \neq I$. Thus $SO(3)$ is not closed under scaling by reals, and is thus not a vector space. Hence, we see that even in this case, the Lie group is not a vector space, while the Lie algebra is a vector space. Philosophically, skew-symmetric matrcies correspond to angular momenta, which are indeed the "velocities" of a rotation.

For each secti
also add lingu
corollaries. Th
make it a bit e
to digest the t
sition from de
math topic to
math topic

5.4.5 Optimisation on Riemannian Manifolds

.

The eventual goal of the `word2vec` training regime is to minimise the loss function which attempts to align word vectors of words occuring in similar contexts, as per the distribution hypothesis. Classical `word2vec` uses gradient descent to mimise this loss function, as it is computationally cheap, and has good theoretical guarantees of being able to find local minima. Hence, we propose to use *Riemannian gradient descent*, an analogue of gradient descent which enables the minimization of a loss function for arbitrary Riemannian manifolds, and in particular, for loss functions over 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 $\overline{\nabla}c : (p : \overline{M}) \to \overline{T_pM}; \overline{\nabla}c \equiv dE\bar{d}c$

The normal space, $\overline{N_{x_0}M}$ is the orthogonal complement of the tangent space, defined as $\overline{N_{x_0}M} \equiv \{v \in \mathbb{R}^n \mid \langle v|\overline{T_{x_0}M}\rangle = 0\}$. 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 $R(p \in \overline{M}) = p$.

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

We point out that this algorithm, when specialized to $\mathbb{R}^n$ as a Lie group recovers the fuzzy set representation, where we perform similarity and analogy on the *logarithmized inputs*, which we interpreted as treating `word2vec` vectors as fuzzy sets. We leave the experimentation on more exotic manifolds as open for future work.

Chapter 6

Conclusion and Future Work

> A good proof is one that makes us wiser.
>
> Yuri Manin

In this thesis, we focus on *empirical study* to provide *mathematical* formalisms for word embeddings. Towards this goal, we provide two directions of research: One based on the linguistic theory of Montague semantics, which we link to the formal theory of abstract interpretation, from which we produce fuzzy set representations. These fuzzy embeddings are extracted from standard word embeddings, and provide access to set theoretic and probabilistic operations on word embeddings. This exhibits our conjecture that word embeddings are in fact capturing montagueian based semantics. The second prong of our research is purely mathematical, where we analyze the conflation of various ideas in classical word embeddings, and dis-entangle these ideas to provide a framework for generating word embeddings over Lie groups. We see that our generalization provides a cleaner framework to *think* about word embeddings, where we provide separate mathematical objects that captures notions of similarity, analogy, meaning distance, and meaning interpolation. For the future, we envision lines of research which exploits other geometric structures posessed by the manifold, including notions of *parallel transport*, which provide a notion of interpolation of word meaning. Furthermore, we wish to adapt results from PAC learning — in particular the contrastive losses [2] to our generalized setting of Lie groups. In closing, I hope this thesis has given you, the reader, a glimpse into pretty mathematical abstractions, reified by the humble `word2vec`.

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.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] S. Arora, H. Khandeparkar, M. Khodak, O. Plevrakis, and N. Saunshi. A theoretical analysis of contrastive unsupervised representation learning. *arXiv preprint arXiv:1902.09229*, 2019.

[3] F. T. Asr, R. Zinkov, and M. Jones. Querying word embeddings for similarity and relatedness. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 675–684, 2018.

[4] M. Baroni and A. Lenci. Distributional memory: A general framework for corpus-based semantics. *Computational Linguistics*, 36(4):673–721, 2010.

[5] R. Bergmair. *Monte Carlo Semantics: Robust inference and logical pattern processing with Natural Language text*. PhD thesis, University of Cambridge, 2011.

[6] E. Bruni, G. Boleda, M. Baroni, and N.-K. Tran. Distributional semantics in technicolor. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 136–145, 2012.

[7] J. Camacho-Collados and M. T. Pilehvar. From word to sense embeddings: A survey on vector representations of meaning. *Journal of Artificial Intelligence Research*, 63:743–788, 2018.

[8] D. Chen, J. C. Peterson, and T. L. Griffiths. Evaluating vector-space models of analogy. *arXiv preprint arXiv:1705.04416*, 2017.

[9] P. Cousot. Abstract interpretation. *ACM Computing Surveys (CSUR)*, 28(2):324–328, 1996.

[10] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252, 1977.

[11] P. Cousot and R. Cousot. Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In *International Symposium on Programming Language Implementation and Logic Programming*, pages 269–295. Springer, 1992.

[12] G. Emerson and A. Copestake. Functional distributional semantics. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 40–52, 2016.

[13] G. Emerson and A. Copestake. Semantic composition via probabilistic model theory. In *IWCS 2017-12th International Conference on Computational Semantics-Long papers*, 2017.

[14] G. File and P. Sottero. Abstract interpretation for type checking. In *International Symposium on Programming Language Implementation and Logic Programming*, pages 311–322. Springer, 1991.

[15] J. R. Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[16] J. M. Gawron. Improving sparse word similarity models with asymmetric measures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 296–301, 2014.

[17] Y. Goldberg and O. Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[18] E. Grefenstette. Towards a formal distributional semantics: Simulating logical calculi with tensors. *arXiv preprint arXiv:1304.5823*, 2013.

[19] B. Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2015.

[20] J. Henderson and D. N. Popa. A vector space for distributional semantics for entailment. *arXiv preprint arXiv:1607.03780*, 2016.

[21] A. Herbelot and E. M. Vecchi. Building a shared world: Mapping distributional to model-theoretic semantic spaces. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 22–32, 2015.

[22] F. Hill, R. Reichart, and A. Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2015.

[23] T. M. V. Janssen and T. E. Zimmermann. Montague Semantics. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2021 edition, 2021.

[24] U. Khedker, A. Sanyal, and B. Sathe. *Data flow analysis: theory and practice*. CRC Press, 2017.

[25] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[26] L. Lee. Measures of distributional similarity. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 25–32, 1999.

[27] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems*, 27:2177–2185, 2014.

[28] J. Li and D. Jurafsky. Do multi-sense embeddings improve natural language understanding? *arXiv preprint arXiv:1506.01070*, 2015.

[29] X. Li. Fuzzy cross-entropy. *Journal of Uncertainty Analysis and Applications*, 3(1):2, 2015.

[30] D. J. MacKay and D. J. Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

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

[32] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[33] T. Mikolov, W.-t. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013.

[34] J. Munkres. Topology, 2014.

[35] I. S. P. Nation. *Making and using word lists for language learning and testing*. John Benjamins Publishing Company, 2016.

[36] A. Nematzadeh, S. C. Meylan, and T. L. Griffiths. Evaluating vector-space models of word representation, or, the unreasonable effectiveness of counting words near other words. In *CogSci*, 2017.

[37] K. Pearson. Notes on the history of correlation. *Biometrika*, 13(1):25–45, 1920.

[38] J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[39] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[40] R. Rehurek, P. Sojka, et al. Gensim—statistical semantics in python. *Retrieved from genism. org*, 2011.

[41] C. Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 100(3/4):441–471, 1987.

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

[43] L. Van der Plas and J. Tiedemann. Finding synonyms using automatic word alignment and measures of distributional similarity. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 866–873. Association for Computational Linguistics, 2006.

[44] L. Vilnis and A. McCallum. Word representations via gaussian embedding. *arXiv preprint arXiv:1412.6623*, 2014.

[45] L. Xuecheng. Entropy, distance measure and similarity measure of fuzzy sets and their relations. *Fuzzy sets and systems*, 52(3):305–318, 1992.