TITLE

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

DEGREE
in
COURSE

by

NAME
ROLL NUMBER
EMAIL ID

CENTER NAME
International Institute of Information Technology
Hyderabad - 500 032, INDIA
MONTH YEAR

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

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

_____ _____

Date Adviser: Prof. NAME

To SOMEONE

Acknowledgments

Acknowledgements goes here ...

Abstract

Abstract goes here ...

Contents

List of Figures

List of Tables

Table Page

Chapter 1

Introduction

The popular opinion thanks to a confluence of distributional semantics and machine learning is that systems such as `word2vec` learn ”vector representations”. This seemed to me to be confusing the encoding the object (as a tuple of real numbers) with the algebraic structure (that of a vector space). Hence, the modest goal of the thesis is an attempt to disambiguate the term “vector representations”.

Towards this end, we begin by surveying the algebraic structure of a vector space, and what it would really mean if words were vectors. We then provide a short explanation of the training mechanism of `word2vec`, and discuss why the embeddings that are learnt by systems it and its ilk do not deserve to be called vectors.

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

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

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

Chapter 2

Are words vectors?

2.1 word2vec

The core training loop of `word2vec` (trained using negative sampling, in skip-gram mode) involves attempting to learn *two* representations for a given word, called as the positive representation and the negative representation.

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

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

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

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

2.2 Testing regime

2.3 Interpretations

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

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

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

2.4 Popular misconceptions

```
while(1) {
   1. vf = vector of focus word
   2. vc = vector of context word
   3. train such that (vc . vf = 1)
   4. for(0 <= i < negative samples):
            vneg = vector of word *not* in context
            train such that (vf . vneg = 0)
}
```

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

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

2.5 The C implementation

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

The setup is incredibly well done in the C code:

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

https://github.com/tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df663beff43

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

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

https://github.com/tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df663beff43

```
for (a = 0; a < vocab_size; a++) for (b = 0; b < layer1_size; b++)
  syn1neg[a * layer1_size + b] = 0;
```

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

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

  // 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
https://github.com/tmikolov/word2vec/blob/20c129af10659f7c50e86e3be406df663beff43
// Learn weights input -> hidden
for (c = 0; c < layer1_size; c++) syn0[c + l1] += neu1e[c];
```

2.6 Why random and zero initialization?

Once again, since none of this actually explained in the original papers *or on the web*, I can only hypothesize.

My hypothesis is 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.

It's quite ingenious, really, and until this, I'd never really thought of how important initialization strategies really are.

This also explains GloVe's radical choice of having a separate vector for the negative context — they were just doing what `word2vec` does, but they told people about it

Chapter 3

Fuzzy set representations

In this paper, 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.

3.1 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"* `cite: firth1957synopsis`. Popular implementations of word embeddings such as word2vec `cite: mikolov2013efficient` and GloVe `cite: pennington2014glove` aim to represent words as embeddings in a vector space. These embeddings are trained to be oriented such that vectors with higher similarities have higher dot products when normalized. Some of the most common methods of intrinsic evaluation of word embeddings include similarity, analogy and compositionality. While similarity is computed using the notion of dot product, analogy and compositionality use vector addition.

However, distributional representations of words over vector spaces have an inherent lack of interpretablity `cite: goldberg2014word2vec`. Furthermore, due to the symmetric nature of the vector space operations for similarity and analogy, which are far from human similarity judgements `cite: tversky1977features`. Other word representations tried to provide asymmetric notions

of similarity in a non-contextualized setting, including Gaussian embeddings cite: vilnis2014word and word similarity by dependency cite: gawron2014improving. However, these models could not account for the inherent compositionality of word embeddings cite: mikolov2013distributed.

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

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

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

3.2 Related Work

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

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

cite: bergmair2011monte used a fuzzy set theoretic view of features associated with word representations. While these features were not adopted from the vector space directly, it presents a

unique perspective of entailment chains for reasoning tasks. Their analysis of inference using fuzzy representations provides interpretability in reasoning tasks.

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

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

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

3.3 Background: Fuzzy Sets and Fuzzy Logic

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

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

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

$$(A \cap B)[i] \equiv A[i] \times B[i] \quad \text{(set intersection)}$$

$$(A \cup B)[i] \equiv A[i] + B[i] - A[i] \times B[i] \text{ (set union)}$$

$$(A \sqcup B)[i] \equiv \max(1, \min(0, A[i] + B[i])) \text{ (disjoint union)}$$

$$(\neg A)[i] \equiv 1 - A[i] \quad \text{(complement)}$$

$$(A \setminus B)[i] \equiv A[i] - \min(A[i], B[i]) \quad \text{(set difference)}$$

$$(A \subseteq B) \equiv \forall x \in \Omega : \mu_A(x) \leq \mu_B(x) \text{ (set inclusion)}$$

$$|A| \equiv \sum_{i \in \Omega} \mu_A(i) \quad \text{(cardinality)}$$

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

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

This formulation will be useful in section 3.4.4 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.

3.4 Representation and Operations

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

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

3.4.1 Constructing the Tuple of Feature Probabilities

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

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

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

In line with our discussion from 3.3, 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 3.4.2.

3.4.2 Operations on Feature Probabilities

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

3.4.2.0.1 Feature Union, Intersection and Difference In section 3.3, we showed the formulation of fuzzy set operations, assuming a finite universe of elements. As we saw in section 3.4.1, 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.