Gradient Descent vs. Graphical Models

Linas Vepstas

3 August 2018

Abstract

This tract attempts to provide a broad sketch of the unsupervised language learning problem, as seen from various differing viewpoints.

The first part compares the operation of deep-learning/neural-net inspired language models to traditional lingistics-inspired language models. It is demonstrated that skip-grams, such as those in the canonical word2vec model, resemble parse rules associated with dependency grammars. This comparison is not obvious, as these two representations seem to be quite different, the tasks that each is intended to solve seem to be quite different, and the algorithms employed are quite different. In fact, many of these differences are superficial; there is more in common than there might seem. The unifying viewpoint is a vector representation of words-in-context. The context is an N-gram, skip-gram or adagram, in the one case, and a dependency linkage disjunct in the other.

The second part examines the idea of a word+context as a bipartite graph, with words on the left side interconnected to contexts on the right. This bipartite graph can be factored into three parts, with words sorted into buckets of word-sense-disambiguated synonyms on the left, buckets of similar grammatical contexts on the right, and a tightly integrated central factor. Different approaches to factorization are explored, including neural-net-inspired low-rank matrix factorization algorithms, information-theoretic clustering, and, most importantly coclustering.

The third examines how to stitch together the different vector spaces (the different grammatical contexts associated with each word) into a unified whole, using concepts from sheaf theory.

This is a work in progress. The third part is incomplete.

1 Introduction

The introduction here is minimal; it is assumed that the reader is generally conversant with both Link Grammar[1, 2]. A slightly more details review of N-gram, neural net[3] and SkipGram[**?**, 4] models are given, including AdaGram. It is generally assumed that the reader is generally familiar with various concepts employed in the OpenCog language-learning project.

XXX A new introduction needs to be written. Sorry. XXX

1.1 Link Grammar

Link Grammar defines a word-disjunct pair as an object of the general form

```
word: A- & B+ & C+ & ...;
```

The disjunct is the expression to the right of the colon. The notations A-, B+, C+, etc. are called connectors; they are types that indicate what other (classes of) words the given word can connect to. The minus and plus signs indicate whether the connectors link to the left or to the right. The number of such connectors is variable, depending on the disjunct. The ampersand serves as a reminder that, during parsing, all connectors must be satisfied; that is, the connectors are conjoined. A given word can have multiple disjuncts associated with it, these are disjoined from one-another, thus the name "disjunct".

For the language learning project, it is convenient to extend the above to the notion of a pseudo-disjunct, where the connector types are replaced by instances of individual words. For example

```
ran: girl- & home+;
```

is used to represent the grammatical structure of the sentence "the girl ran home", namely, that the verb "ran" can attach to the word "girl" (the subject) on the left, and the word "home" (the object) on the right. An early goal of the language learning project is to automatically discern such pseudo-disjuncts; a later goal is to automatically classify such individual-word connectors into word-classes, and so generalizing individual words into connector types, just as in traditional Link Grammar.

The primary learning mechanism is to accumulate observation counts of different disjuncts, thus leading naturally to the idea of word-vectors. For example, one might observe the vector $\vec{v}_{ran}$ represented as

```
ran: 3(girl- & home+) + 2(girl- & away+);
```

that might naturally arise if the sentence "the girl ran home" was observed three times, and "the girl ran away" was observed twice. Such vectors can be used to judge word-similarity. For example, a different vector $\vec{v}_{walked}$ represented as

```
walked: 2(girl- & home+) + 3(girl- & away+);
```

suggests that the cosine-product $\cos(\vec{v}_{ran}, \vec{v}_{walked})$ between the two might be used to judge word-similarity: "ran" and "walked" can be used in syntactically similar ways.

The disjunct representation also allows other kinds of vectors, such as those anchored on connectors. Using the examples above, one also has a vector for the word "home", which can be awkwardly written as

```
3(ran: girl- & home+) + 2(walked: girl- & home+)
```

Note that the counts, here, of 3 and 2, are identical to the counts above: all of these counts are derived from the same observational dataset. What differs is the choice of the attachment-point for which the vector is to be formed.

A less awkward notation for these two kinds of vectors would be nice; however, for current purposes, it is enough to distinguish them with superscripts D and C: *viz.* write $\vec{v}^D_{ran}$ for the disjunct-based vector, and $\vec{v}^C_{home}$ for the connector-based vector. Given any word w, there will be vectors $\vec{v}^D_w$ and also $\vec{v}^C_w$. These two vectors can be taken together as $\vec{v}^D_w \oplus \vec{v}^C_w$ and inhabit two orthogonal subspaces of $\vec{V}^D \oplus \vec{V}^C$. There are many interesting relationships between these subspaces; the most important of these, developed in a later section, is that they can be viewed as sections of a sheaf of a graph.

The correct conceptual model for the observational data is that of a large network graph, with observation counts attached to each vertex. This network graph can be understood as a sheaf (see reference). The above examples show how certain sections of that graph can be made to correspond to vectors. Obviously, these vectors are not independent of one-another, as they are all different slices through the same dataset. Rather, they provide a local, linear view of the language graph, reminiscent of the tangent-space of a manifold.

1.2 Statistical Models

The task of language learning is commonly taken to be one of estimating the probability of a text, consisting of a sequence of words. One common model assumes that the probability of the text can be approximated by the product of the conditional probabilities of individual words, and specifically, of how each word conditionally depends on all of the previous ones:[3]

$$\widehat{P}\left(w_1^T\right) = \prod_{t=1}^{T} P\left(w_t \left| w_1^{t-1}\right.\right)$$

Here, the text is presumed to consist of T words w_t occurring in sequential order. The notation w_i^n is used to denote a sequence of words, that is, $w_i^n = (w_i, w_{i+1}, \cdots, w_n)$. Thus, the text as a whole is denoted by w_1^T, and so $\widehat{P}\left(w_1^T\right)$ is an approximate model for the probability $P\left(w_1^T\right)$ of observing the text (the carat over P serving to remind that approximations are being made; that the model is an approximation for the "true" probability.)

Although this statistical model is commonly taken as gospel, it is, of course, wrong: we know, a priori, that sentences are constructed whole before being written, and so the current word also depends on future words, ones that follow it in the text. This is the case not just at the sentence-level, but also at the level of the entire text, as the writer already has a theme in mind. To estimate the probability of a word at a given location, one must look at words to both the left and right of the given location.

At any rate, for T greater than a few dozen words, the above becomes computationally intractable, and so instead one approximates the conditional probabilities by limiting the word-sequence to a sliding window of length N. It is convenient, at this point, to also allow words on the left, as well as those on the right, to determine the conditional probability. Following Mikolov[?], one

may write the probability

$$p\left(w_t \left| w_{t-c}, \cdots w_{t-1}, w_{t+1}, \cdots w_{t+c}\right.\right)$$

of observing a word w_t, at location t in the text, as being conditioned on a local context (sliding window) of $N = 2c$ surrounding words, to the left and right, in the text. The probability of the text is then modeled by

$$\widehat{P}\left(w_1^T\right) = \prod_{t=1}^{T} p\left(w_t \left| w_{t-c}, \cdots w_{t-1}, w_{t+1}, \cdots w_{t+c}\right.\right)$$

The smaller window does make the computation more tractable. Here, the window is written in a manifestly symmetric fashion; in general, one might ponder a window with a different number of words to the left or right.

The above contains another key simplification: the total probability is assumed to factor into the product of single-word probabilities, and each single-word probability is translationally invariant; that is, the probability has no explicit dependence on the index t. This is commonly taken to be a reasonable simplification, but is, of course, "obviously" wrong. The words at the end of a text occur with different probabilities than those at the beginning; for example, in a dramatic story, a new character may appear mid-way, or the setting may move from indoors to outdoors, with furniture-words becoming uncommon, while nature-words becoming common. The translational invariance only becomes plausible in the limit of $T \to \infty$ where one is considering "all human language". This too, is preposterous; first, because not everything that can be said has been said; second, because different individuals speak differently, and third, because new words are invented regularly, as others become archaic. Despite all this, the assumption of translational invariance is entirely appropriate at this level.

1.3 N-Gram Model

Without any further elaboration, and taken at face value, the above defines what is more-or-less the "classic" N-gram model. The general property is that there is a sliding window of N words in width, and one is using all of those words to make a prediction. Because of the combinatorial explosion in the size of the vocabulary, N is usually kept small: $N = 3$ (trigrams) or $N = 5$. That is, for a vocabulary of W words, there are W^N probabilities p that must be computed (trained) and remembered. For $W = 10^4$ and $N = 3$, this requires $W^N = 10^{12}$ probabilities to be maintained: at 8 giga-probabilities, this is clearly near the edge of what is possible with present-day computers.

The model can be made computationally tractable with several variants. One common variant is to blend together, in varying proportions, the models for $N = 0$, $N = 1$ and $N = 2$. The details are of no particular concern to the rest of this essay.

1.4 Model Building

The combinatorial explosion can be avoided by proposing models that "guess", in an *a priori* fashion, that some of these probabilities are zero, or that they are (approximately) equal to one-another, or that they can be grouped or summed in some other ways. More correctly, one hypothesizes that the vast majority of the probabilities are either zero or fall into classes where they are equal: say, all but one in ten-thousand, or thereabouts, is the *de facto* order of magnitude obtained in these models. Alternately, one can hypothesize that certain linear combinations of the probabilities are identical; this is the common tactic of most deep-learning algorithms.

There is a fairly rich variety of models. Reviewed immediately below are two common foundational models: the so-called CBOW model, and the SkipGram model. The general goal of this paper is to demonstrate that Link Grammar, and thus dependency grammars in general, can be understood to also fit into this same class of probabilistic models. What differs is the mechanism by which the models are trained; the Link Grammar training algorithm, already sketched above, is not a hill-climbing/deep-learning technique. A proper comparison will be made after the initial review of the CBOW and SkipGram models.

1.5 CBOW

Mikolov, *etal[4, ?]* propose a model termed as the "continuous bag-of-words" model. It is presented as a simplification of neural net models that have been proposed earlier. As a simplification, it makes sense to present it first; neural net models are reviewed below.

In the CBOW model, each (input) word w is represented by an "input" vector $\vec{v}_w$ of relatively small dimension. One does the same in an ordinary bag-of-words model, but with much higher dimension. In an ordinary bag-of-words model, one considers a vector space of dimension W, with W being the size of the vocabulary. One then makes frequentist observations, counting how often each word is observed in some text. The result of this counting is a vector living in a W-dimensional space. Different texts correspond to different vectors. However, nothing about the grammar of individual sentences or words is learned in this process.

In the CBOW model, the dimension of the space in which the vector $\vec{v}_w$ lives is set to a much smaller value $D \ll W$. Commonly used values for D are in the range of 50–300; by contrast, typical vocabulary sizes W range from 10^4 to 10^6. The mismatch of dimensions results in the mapping sometimes being called "dimensional reduction".

In the CBOW model, the mapping from the space of words to the space of vectors $\vec{v}_w$ is linear; there are no non-linear functions, as there would be in a neural net. That is, the mapping is given by a matrix π of dimension $D \times W$. Maps from higher to lower dimensional spaces are called "projections". (The notation of the lower-case Greek letter π for projection is common-place in the mathematical literature, but uncommon in the machine-learning world. It's

convenient here, as it avoids burning yet another roman letter.) The projection matrix π is unknown at the outset; the goal of training is to determine it.

The CBOW is a model of the conditional probability

$$p\left(w_t \left| w_{t-c}, \cdots w_{t-1}, w_{t+1}, \cdots w_{t+c}\right.\right)$$

As already mentioned, it projects each word down to a lower-dimensional space. To get the output word w_t, one has to "unproject" back out, which is conventionally done with a different projection matrix π'. To establish some notation: let $\hat{e}_w$ be a W-dimensional unit vector that is all-zero, except for a single, solitary 1 in the w'th position (this is sometimes called the "one-hot" vector in machine learning). Then one has that $\vec{v}_w = \pi \hat{e}_w$ is the projection of w – equivalently, it is the w'th column of the matrix π. For the reverse projection, let $\vec{u}_w = \pi' \hat{e}_w$. (Many machine-learning texts write $\vec{v}'_w$ for $\vec{u}_w$; we use a different letter here, instead of a prime, to help maintain distinctness. Almost all machine-learning texts avoid putting the vector arrow over the letters; here, they serve to remind the reader where the vector is, so as to avoid confusion in later sections.)

Let I be the set of context (or "input") word subscript offsets; to be consistent with the above, one would have $I = \{-c, -c+1, \cdots, -1, +1, \cdots, +c\}$. By abuse of notation, one might also write, for offset t or for word w_t, that $I = \{t-c, t-c+1, \cdots t-1, t+1, \cdots, t+c\}$ or that $I = w_I = \{w_{t-c}, w_{t-c+1}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+c}\}$; exactly which of these sets is intended will hopefully be clear from context.

The CBOW model then uses the Boltzmann distribution obtained from a certain partition function, sometimes called the "softmax" model. The model is given by

$$p\left(w_t \left| w_I \right.\right) = \frac{\exp \sum_{i \in I} \vec{u}_t \cdot \vec{v}_i}{\sum_{j \in W} \exp \sum_{i \in I} \vec{u}_j \cdot \vec{v}_i}$$

The sum in the numerator runs over all words in the input set I; the sum in the denominator runs over all words in the vocabulary W. The sum in the denominator explicitly normalizes the probability to be a unit probability. That is, for fixed w_I, one has that $1 = \sum_j p\left(w_j \left| w_I \right.\right)$.

Computation of the matrices π and π' is done by explicitly expanding them in the expression above, and then performing hill-climbing, attempting to maximize the probability. To provide a nicer landscape for hill-climbing, it is usually done on the "loss function" $E = -\log p\left(w_t \left| w_I \right.\right)$. One works with the gradient $\nabla_{\pi,\pi'} E$ and takes small steps uphill. The detailed mechanics for doing this does not concern this essay; it is widely covered in many other texts[5].

By convention, π and π' are taken to be two distinct projection matrices. I do not currently know of any theoretical reason nor experimental result why this should be done, instead of taking $\pi = \pi'$.

1.6 SkipGram

The SkipGram model is very similar to the CBOW model, and is commonly presented as it's opposite. It uses essentially the same Boltzmann distribution

as CBOW, except that it is now looking at the probability $p(w_I|w_t)$ of the context I given the target word w_t. Explicitly, the model is given by

$$p(w_I|w_t) = \frac{\exp \sum_{i\in I} \vec{u}_t \cdot \vec{v}_i}{\sum_{I\in W^N} \exp \sum_{i\in I} \vec{u}_t \cdot \vec{v}_i}$$

That is, the word w_t is held fixed, and the sum ranges over all possible N-tuples I in the (now much larger) space W^N (as always, N is the width of the sliding window.

As in the CBOW model, the projection matrices π and π' are computed by means of hill-climbing the loss-function. The important contribution of of Mikolov *et al.* is not only to describe this model, but also to propose several algorithmic variations to minimize the RAM footprint, and to improve the speed of convergence.

Both SkipGram and CBOW are sometimes called "neural net" models, but this is perhaps slightly misleading, as neither make use of the sigmoid function that is characteristic of neural nets. Given that the characteristic commonality is that the probabilities are obtained by hill-climbing, it seems more appropriate to simply call these "deep learning" models. The distinction is made more clear in the next section.

1.7 Perceptrons as Neural Nets

The neural net model proposed by Bengio[3] is worth reviewing, as it places the CBOW and SkipGram models in context. It builds on the same basic mechanics, except that it now replaces the dot-product $\sum_{i\in I} \vec{u}_t \cdot \vec{v}_i$ by a "hidden" feed-forward (perceptron) neural layer.

The perceptron consists of another projection, this time called the "weight matrix" H, and a non-linear sigmoid function $\sigma(x)$, which is commonly taken to be $\tanh x$ or $1/(1+e^{-x})$ or similar, according to taste.

The input to the weight matrix is the vector $\vec{v}_I$ which is a Cartesian product of the input vectors $\vec{v}_i$ for the $i \in I$. That is,

$$\vec{v}_I = \vec{v}_{t-c} \times \vec{v}_{t-c+1} \times \cdots \times \vec{v}_{t+c}$$

where, for illustration, we've taken the same I as given in the previous sections. This vector is ND-dimensional, where, as always, N is the cardinality of I and D is the dimension of the projected space.

The input vector $\vec{v}_I$ is then sent through a weight matrix h to a "hidden" neuron layer consisting of H neurons. That is, the matrix h has dimensions $ND \times H$. An offset vector $\vec{d}$ (of dimension H) is used to properly center the result in the sigmoid. The output of the perceptron is then the H-dimensional vector

$$\vec{s} = \sigma\left(h\vec{v} + \vec{d}\right)$$

where the sigmoid is understood to act component by component; that is, the k'th component $[\vec{s}]_k$ of the vector $\vec{s}$ is given by

$$[\vec{s}]_k = \sigma\left(\left[h\vec{v} + \vec{d}\right]_k\right)$$

This is then passed through the "anti-"projection matrix π', as before, except that here, π' must be $H \times W$-dimensional. Maintaining the notation from earlier sections, the perceptron model is then

$$p(w_t|w_I) = \frac{\exp \vec{u}_t \cdot \vec{s}}{\sum_{j \in W} \exp \vec{u}_j \cdot \vec{s}}$$

Just as in the CBOW/SkipGram model, training can be accomplished by hill-climbing, this time by taking not only π and π' as free parameters, but also h and $\vec{d}$.

Typical choices for the dimension H is in the 500–1000 range, and is thus comparable to the size of ND, making the weight matrix h approximately square. That is, the weight matrix h does a minimal amount of, if any at all, of dimensional reduction.

2 Similarities and Differences

On the face of it, the description given for Link Grammar seems to bear no resemblance to that of the description of probabilistic neural net models, other than to invoke vectors in some way. The descriptions of language appear to be completely different. There are similarities; they are obscured both by the notation, and the viewpoint.

2.1 Disjuncts as Context

Consider the probability

$$p(w_t|w_{t-c}, \cdots w_{t-1}, w_{t+1}, \cdots w_{t+c})$$

This is meant to indicate the probability of observing the word w_t, given c words that occur before it, and c words that occur after it. Let $c = 1$ and let $w_t = ran$, $w_{t-1} = girl$ and $w_{t+1} = home$. This clearly resembles the Link Grammar disjunct

```
ran:  girl- & home+;
```

One difference is the Link Grammar disjunct notation does not provide any location at which to attach a probability.[1] This can be remedied in a straightforward manner: write d for the disjunct, $girl- \& home+$ in this example. One can then define the probability

$$p(w|d) = \frac{p(w,d)}{p(*,d)}$$

[1] The Link Grammar software does provide a device, called the "cost", which is an additive floating point number that represents the penalty of using a particular disjunct. It can be thought of as being the same thing as $-\log p(w|d)$. The hand-crafted dictionaries provide hand-crafted estimates for this cost/log-likelihood.

where $p(w,d)$ is the probability of observing the pair (w,d), while

$$p(*,d) = \sum_{w=1}^{W} p(w,d)$$

is simply the sum over all words in the vocabulary.

The resemblance, at this point, should be obvious: the disjunct d plays the role of the N-gram context. Abusing the existing notation, one should should understand that

$$d \approx w_{t-c}, \cdots w_{t-1}, w_{t+1}, \cdots w_{t+c}$$

The abuse of notation was partly cured by writing w_I for the "input" words of the CBOW/SkipGram models, so that I was a set of relative indexes into the text. The disjunct notation does everything that the index notation can do: it specifies a fixed order of words, to the left, and to the right of the target ("output") word w_t.

In fact, the disjunct notation can do more: it can also encode parse information. That is, one could take the disjunct as being a sequence of words, with no gaps allowed between the words. If this is done, then the disjunct d becomes fully compatible with the index set I and one can legitimately write that $d = w_I$ are just two notations for saying the same thing. But the disjunct can also do more: it effectively suggests that the index set can be used as parse information.

2.2 Skip-Grams and Grammar

The above explicit identification of $d = w_I$ suggests that CBOW and SkipGram models already encode grammatical information, and that finding it is as simple as re-interpreting w_I as a disjunct. That is, given either form $p(w_t|w_I)$ or $p(w_I|w_t)$, simply re-interpret w_I as specifying left-going and right-going connectors. The Link Grammar cost is nothing other than the "loss function" $E = -\log p(w_t|w_I)$; they are one and the same thing. One could do this immediately, today: given a SkipGram dataset, one can just write an export function, and dump the contents into a Link Grammar dictionary. All that remains would be to evaluate the quality of the results.

Disjuncts are intended to capture the dependency grammar description of a language. A dependency grammar naturally "skips" over words, and "adaptively" sizes the context to be appropriate. Consider the dependency parse of "The girl, upset by the taunting, ran home in tears." There are four words, and two punctuation symbols separating the word "girl" from the word "ran". Dependency grammars do not have any difficulty in arranging for the attachment of the words "girl–ran", skipping over the post-nominal modifier phrase "upset by the taunting", which attaches to the noun, and not the verb: it's the girl who is upset, not the running.

Such long-distance attachments are problematic for CBOW or Skip-Grams, in several ways. One is that the window N must be quite large to skip over the post-nominal modifier. Counting punctuation, one must look at least seven

words to the right, in the above example. If the window is symmetric about the target word, this calls for $N \geq 14$, which is a bit larger than currently reported results; for example, Mikolov[**?**] reports results for $N = 5$. The point here is that

$$p\left(w_t = \text{girl} \,|w_{t-1} = \text{the} \,, w_{t+1} = \text{ran}\right)$$

can be trivially re-interpreted as the dictionary entry

```
girl: the- & ran+;
```

However, that is not what is needed to parse "The girl, upset by the taunting, ran home in tears." What is needed, instead, is the dictionary entry

```
girl: the- & upset+ & ran+;
```

which is invisible with an $N = 5$ window. The punctuation is also important for the post-nominal modifier; somewhere one must also find

```
upset: girl- & ,- & by+ & ,+;
```

which also does not fit in an $N = 5$ window; it requires at least $N = 9$. Long-distance attachments present a problem for the simpler, less sophisticated deep-learning models.

Another difficulty is that dependency grammars are naturally "adaptive" by design: verbs tend to have more attachments that nouns, which have more attachments than determiners or adjectives. That is, dependency grammars already "know" that the correct size of the context for determiners and adjectives is one: a determiner can typically modify only one noun. One expects the entry

```
the: girl+;
```

The size of the context for the word "the" is just $N = 1$; more is not needed. If the deep-learning model fails to explicitly contain an entry of the form

$$p\left(w_t = \text{the} \,|w_{t+1} = \text{girl}\right)$$

with no other context words present, then one will have trouble building a suitable dictionary.

Comment: I assume that Parsey McParseFace overcomes all of the above mentioned problems, but I have not studied it.

2.3 Graph Algorithms vs. Gradient Descent

The biggest difference between the two approaches is the nature of the training algorithm. The disjunct counting can be said to be a form of a "graph algorithm", whereas the deep-learning algorithms, proceeding by relaxation or hill-climbing, can be said to be a form of "gradient descent". One is local, the other is global in it's view of relationships. One is greedy, and exposes graphical structure immediately; the other exposes a graph only if some sort of thresholding is applied to eliminate weak links.

The disjunct probabilities are obtained by direct counting. That is, after obtaining an observational count $N(w,d)$ one computes a normalized frequency of observations

$$p(w,d) = \frac{N(w,d)}{N(*,*)}$$

so that the frequency is properly normalized: $p(*,*) = 1$. The $N(w,d)$ is just the observational count of observing the pair (w,d) in text; the probability is just the frequentist probability.

By contrast, the CBOW/SkipGram models obtain a probability similar to $p(w,d)$, but using a different technique and a different notation; for example, $p(w_j|w_I)$. The sections above point out that w_I is a lot like d; for some purposes (many purposes?) they can be treated as being the same thing. The big difference is that w_I is lacking the graph structure of d; there is some buried graph-like structure in w_I but it is not overt, and one must work to make it overt. The other big difference is that $p(w_j|w_I)$ is treated as an unknown, and arrived at by gradient descent algorithms applied to an objective function (the "loss function"), rather than by direct observational counting.

Consider again the mechanism used to obtain the counts $N(w,d)$. The first half of this mechanism is commonly called MST parsing (Maximum Spanning Tree parsing), and is described in [6]. It is a form of a greedy graph algorithm. One begins by considering the graph clique, wherein every word in the sentence is related (joined by an edge) to every other word in a sentence. Each edge is associated with a metric, that defines the length or size of the edge. In traditional MST, this metric is the mutual information of the word-pair. One can proceed in three ways from here:

- Apply thresholding, and discard all edges that have a weak, low quality connection. The result may be a disconnected graph, or a multiply-connected graph. A fixed threshold might reject too much, or it might reject too little.
- Apply a greedy algorithm, and keep only the edges of the highest quality, until a spanning tree is found, spanning all words in the sentence.
- A combination of the two.

The last may be the best. In traditional linguistics, one is interested in a parse tree that connects all of the words in a sentence, and thus indicates which words are related to which. More connections than what is in the spanning tree just confuse the issue, at least for traditional linguistics.

For disjunct counting, insisting on a spanning tree might be too strong a condition. For disjunct counting, one is only interested in how words connect to other words: one is interested in extracting and counting the "jigsaw puzzle pieces" that represent how words can connect to other words. To obtain these jigsaw-puzzle pieces, one does not need a tree that spans all words in the sentence; it is acceptable that some words might be omitted. Nor is it strictly

necessary that the parse graph be a tree: it is OK if it has cycles, since these cycles often indicate important grammatical relations.

A spanning-tree algorithm can be thought of as a kind-of thresholding algorithm, but with a local, dynamically-adjustable threshold. When a word has very strong connections to all other words, one might consider dynamically adjusting a threshold to a high value; when a word has only weak connections to all other words, one might dynamically adjust the threshold to a low value. This type of dynamically-adjustable thresholding can be confusing to define and difficult to optimize; by contrast, the spanning tree is simple and direct. At an abstract level, though, the differences can be imagined to be less than they first appear.

The net result is that by observing disjuncts, one is observing an explicit graphical structure. The disjunct is overt, in the foreground, explicitly demonstrated. It is obtained by explicitly searching for a graphical structure.

Because every graph has a corresponding adjacency matrix, one can always approach the problem from the other direction: given a matrix, declare it to be a graph, with the matrix entries being "weights" on the graph edges. This is the approach taken by the neural net, deep-learning models. They clearly have taken the world by storm, and are quite succesful in what they do. The disadvantage is that they obscure the explict graphical structure of natural langauge.

3 Model Building and Vector Representations

The key driver behind the deep-learning models is the replacement of intractable probabilistic models by those that are computationally efficient. This is accomplished in several stages. First, the full-text probability function $P(\text{sentence} \mid \text{fulltext})$ is replaced by the much simpler probability function $P(\text{word} \mid \text{fulltext})$. The former probability function is extremely high-dimensional, whereas the later is less so. Its still computationally infeasible, so there are two directions one can go in. The traditional bag-of-words model replaces "fulltext" by "set of words in the fulltext" AKA the "bag", and so one computes $P(\text{word} \mid \text{bag})$ which is computationally feasible. Algorithms such as TF-IDF, and many others accomplish this. The characteristic idea here is to ignore the (syntactic) structure of the full-text, completely erasing all indication of word-order.

The bag, however, loses syntactic and semantic structure, and so goes to far. An alternate route is to start with $P(\text{word} \mid \text{fulltext})$ and simplify it by using instead a sliding-window probability function $P(\text{word} \mid \text{window})$, thus giving the N-gram model. The characteristic idea here is to explicitly set $P(\text{word} \mid \text{other-words}) = 0$ whenever the other-words are not in the window.

The N-gram model is still computationally intractable for $N \geq 3$ and so the deep-learning models propose that yet more entries in $P(\text{word} \mid \text{window})$ can be ignored or conflated. Conceptually, the models propose computing $P(\text{word} \mid \text{context})$ with the context being a projection to a low-dimensional space. These ideas can be illustrated more precisely. Let

$$\vec{v}_I = \vec{v}_{t-c} \times \vec{v}_{t-c+1} \times \cdots \times \vec{v}_{t+c}$$

be the context, with $\vec{v}_w = \pi \hat{e}_w$ the projection of the unit vector of the word down to the low-dimensional "hidden layer" vector space. This projection can be written as $\vec{v}_I = [\pi \oplus \cdots \oplus \pi] (\hat{e}_{t-c} \times \hat{e}_{t-c+1} \times \cdots \times \hat{e}_{t-c})$ where $\pi \oplus \cdots \oplus \pi$ is the block-diagonal matrix

$$\pi \oplus \cdots \oplus \pi = \begin{bmatrix} \pi & 0 & & & \\ 0 & \pi & & & \\ & & \ddots & & \\ & & & \pi & 0 \\ & & & 0 & \pi \end{bmatrix}$$

and so the off-block-diagonal entries are explicitly assumed to be zero, as an *a priori* built-in assumption. Note that the zero entries in this matrix greatly outnumber the non-zero entries. Almost all entries are zero.

Its useful to keep tabs on these sizes. The matrix π was $D \times W$-dimensional, with W the number of vocabulary words (as always) and D the "hidden" dimension. For a window of size N, the matrix $\pi \oplus \cdots \oplus \pi$ has dimensions $ND \times NW$. Of these, only NDW are non-zero, the remaining $N(N-1)DW$ are zero. That's a lot of zeros.

One can do one of several things with the vector $\vec{v}_I$. In the SkipGram and CBOW models, one sums over words; that is, one creates the vector $\sum_{i \in I} \vec{v}_i$. Its worth writing this out, matrix style. One has that

$$\sum_{i \in I} \vec{v}_i = S \vec{v_I}$$

where the matrix S is a concatenation of identity matrices.

$$S = \left[\begin{array}{cc|cc|c|cc} 1 & 0 & 1 & 0 & & 1 & 0 \\ 0 & 1 & 0 & 1 & \cdots & 0 & 1 \\ & & & & & & \\ & \ddots & & \ddots & & & \ddots \\ & 1 & & 1 & & & 1 \end{array}\right]$$

The reason for writing it out in this way to understand that there is another dimensional reduction: again, almost all entries in this matrix are zero. Each identity matrix was $D \times D$ dimensional, and there are N of them, so that S has dimensions $D \times ND$. Of these, there are only ND non-zero entries; the remaining $ND(D-1)$ are all zero. The reduction is huge.

For the perceptron model of Bengio, the matrix S is replaced by a weight matrix h projecting to the perceptron layer. All of the entries in the matrix h are, by assumption, non-zero. This perhaps helps make it clear just how much more complex the perceptron model is. Since h is an approximately square matrix, this implies a large-number of non-zero entries.

Its worth getting an intuitive feeling for the size of these numbers: following Mikolov, assume that $W = 10^4$ although this sharply underestimates the size of the vocabulary of English. Assume $N = 5$ and $D = 300$. The size of the input

vector space is thus $W^N = 10^{20}$, this is being modeled by a vector space of size 300. The sparsity is thus

$$\log_2 \frac{10^{20}}{300} = 31.3 \text{ bits}$$

A truly vast amount of potential information is being discarded by this language model. Of course, the claim is that the English language never carried this much information in the first place: almost all five-word sequences are meaningless non-sense; only a very small number of these are syntactically valid, and somewhat fewer are semantically meaningful.

This exposes the real question: just how meaningful are the CBOW/Skip-Gram models, and can one find better models that also have "lots of zero entries", but distribute them in a more accurate way?

3.1 Sparsity

The last question can be answered by noting that the Link Grammar disjunct representation is also a very highly sparse matrix; however, it is sparse in a very different way, and does NOT have the block-diagonal structure of the deep-learning systems. The can be explicitly illustrated and numerically quantified.

At the end of one stage of training, one obtains a matrix of observation counts $N(w,d)$, which are easily normalized to probabilities $p(w,d)$. This is, in fact, a very sparse matrix. Four datasets can be quoted: for English, the so-called "en_mtwo" dataset, and the "en_cfive" dataset; for Mandarin, the "zen" and "zen_three" datasets. Please refer to the diary for a detailed description of these datasets. The dimensions and sparsity are summarized in the table below.

name	W	$\|d\|$	sparsity
en_mtwo	137K	6.24M	16.60 bits
en_cfive	445K	23.4M	18.32 bits
zen	60K	602K	15.46 bits
zen_three	85K	4.88M	15.85 bits

Here, as always, $W = |w|$ is the number of observed vocabulary words, $|d|$ is the number of observed disjuncts, and the sparsity is the log of the number of number of non-zero pairs, measured in bits:

$$\text{sparsity} = \log_2 \frac{|w|\,|d|}{|(w,d)|}$$

Notable in the above report is that the measured sparsity seems to be approximately language-independent, and dataset-size independent.

Some of the observed sparsity is due to a lack of a sufficient number of observations of language use. Some of the sparsity is due to the fact that certain combinations really are forbidden: one really cannot string words in arbitrary order. What fraction of the sparsity is due to which effect is unclear. Curiously,

increasing the number of observations (en_cfive vs. en_mtwo) increased the sparsity; but this could also be due to the much larger vocabulary, which is now even more rarely observed. A significant part of the expanded vocabulary includes Latin and other foreign-language words, which, of necessity, will be very infrequent, and when they occur, they will be in set phrases that readers are expected to recognize. The point here is that one cannot induce a foreign-language grammar from a small number of set phrases embedded in English text. A major portion of the expanded vocabulary are geographical place names, product names and the like, which are also inherently sparse. Unlike the foreign phrases, this does not mean that they are inflexible in grammatical usage: one can use the name of a small town in a vast number of sentences, even if the observed corpus uses it in only a few.

Compared to the back-of-the-envelope estimate of sparsity for SkipGrams, the numbers reported above are much lower. There are several ways to interpret this: the simple disjunct model, as presented above, fails to compress sufficiently well, or the SkipGram model compresses too much. Its likely that both situations are the case.

3.1.1 No Large Data Limit

Natural language does not have a large-data limit. More generally, Zipf distributions cannot have a large-data limit.

Ignoring "natural" sparsity due to forbidden grammatical constructions, it is also the case that the input dataset $p(w, d)$ is both noisy and incomplete. It is noisy because the sample size is not sufficiently large to adequately approach a large-sample-size limit. Reaching this limit is fundamentally impossible, from first principles, if one assumes the Zipf distribution (as is the case here). For a Zipf distribution, half the dataset necessarily consists of *hapax legomena*. Another 15% to 20% are *dis* and *tris legomena*. Increasing the number of observations do not change these ratios: the more one observes, the more singular phenomena one will find. Noise in the dataset is unavoidable. Furthermore, implicit in this is that the dataset is necessarily incomplete: if half the dataset consists of events that were observed just once; there are "even more" events, that were never observed.

Consider, for example, the set of short sentences. One might think that, if one was able to observe every sentence ever spoken or written, one might eventually observe ever grammatically valid noun-verb combination. This is not so. The sentence "green ideas sleep furiously" is quite common, as it is a stock example sentence in linguistics. However, the similar sentence "blue concepts wilt skillfully" probably has never been written down before, until just now. The law of large numbers does not apply to the Zipfian distribution. The matrix $p(w, d)$ is necessarily noisy and incomplete, no mater how large the sample size. What is not clear is what fraction of the sparsity is due to the Zipf distribution, and what fraction is the sparsity is due to forbidden grammatical constructions.

3.2 Word Classes

In operational practice, dependency grammars work with word-classes, and not with words. That is, one carves up the set of words into grammatical classes, such as nouns, verbs, adjectives, etc. and then assign words to each. Each grammatical class is associated with a set of disjuncts that indicate how a word in that class can attach to words in other classes. This can be made notationally precise.

Given a word w and the disjunct d it was observed with, the goal is to classify it into some grammatical category g. The probability of this usage is $p(w, d, g)$, and it should factorize into two distinct parts:

$$p(w, d, g) = p'(w, g)\, p''(g, d)$$

None of the three probabilities above are known, a priori, and not even the number of grammatical classes are known at the outset. Instead, one has the observational data, that

$$p(w, d) = p(w, d, *) = \sum_{g \in G} p(w, d, g)$$

where G is the set of all grammatical classes. The goal is then to determine the set G and to perform the matrix factorization

$$p(w, d) \approx \sum_{g \in G} p'(w, g)\, p''(g, d) \tag{1}$$

Ideally, the size of the set G is minimal, in some way, so that the matrices $p'(w, g)$ and $p''(g, d)$ are of low rank. In the extreme case of G having only one element, total, the factorization is the same as the outer product, or tensor product, of two vectors.

In the following, the prime-superscripts are dropped, and the probabilities are written as $p(w, g)$ and $p(g, d)$. These are two different probabilities; which in turn are not the same as $p(w, d)$. Which is which should be apparent from context.

There are two ways of performing the factorization of eqn 1: by applying graphical methods (such as clustering) or by applying gradient descent methods (typically associated with neural net algorithms). These two approaches are explored below.

3.2.1 Learning Word Senses

The goal of the factorization is to capture semantic information along with syntactic information. Typically, any given (w, d) pair might belong to only one grammatical category. So, for example, the pair

girl: the−;

would be associated with $g = <\texttt{common}-\texttt{count}-\texttt{nouns}>$. This captures the idea that girls, boys, houses and birds fall into the same class, and require the use of a determiner when being directly referenced. This is distinct from mass nouns, which do not require determiners. This suggests that, to a large degree, the factorization might be approximately block-diagonal, at least for the words; that $p(w, g)$ might usually have only one non-zero entry for a fixed word w.

But this assumption should break down, the larger the size of the set G. Suppose one had classes $g = <\texttt{cutting}-\texttt{actions}>$ and $g = <\texttt{looking}-\texttt{verbs}>$; the assignment of

saw: I− & wood+;

would have non-zero probabilities for both g's. For a large number of classes, one might expect to find many distinctions: girls and boys differ from houses and birds, and one even might expect to find sex differences: girls pout, and boys punch, while houses and birds do neither.

Put differently, one expects different classes to not only differentiate crud syntactic structure, but also to indicate intensional properties. Based on practical experience, we expect that most words would fall into at most ten, almost always less than twenty different classes: this can be seen by cracking open any dictionary, and counting the number of word senses for a given word. Likewise for intentional properties: birds sing, tweet and fly and a few other verbs. Houses mostly are, or get something (get built, get destroyed). That is, we expect $p(w, g)$ to be sparse: there might be thousands (or more!) elements in G, but no more than a few dozen $p(w, g)$, and often much less, will be non-zero, for a fixed word w.

3.3 Clustering

A side effect of the matrix factorization of eqn 1 is that it is a *de facto* form of clustering. Whenever one has $p(w, g) > 0$, one can effectively say "word w has been assigned to cluster g". Thus, solving eqn 1 can be seen as an alternative to deploying a clustering algorithm to assign words to word classes.

3.3.1 Multiple class membership

One important distinction between traditional clustering and matrix factorization is that traditional clustering algorithms naively assign a word to only a single cluster, whereas here, one can have multiple $p(w, g) > 0$. One can partly overcome this difficulty with traditional clustering by decomposing the input vector into two: a component parallel to the cluster centroid, and a perpendicular component, and assigning the parallel component to the cluster, while leaving behind the perpendicular component to be assigned to other clusters. The decomposition need not be strict about parallelism: one might choose to merge the component that lies within some angle (or distance) of the cluster centroid. Each such merge then gradually shifts the centroid over. If the left-over perpendicular component is sufficiently small, it can be discarded as noise,

or treated as an anomaly awaiting additional data.

This splitting of a vector into components, and then placing each component in a different cluster might perhaps be reminiscent of fuzzy clustering, where one object may be placed into two clusters. However, what is proposed above is *not* fuzzy clustering. The goal is disambiguate a word precisely into the intended sense, and to assign that sense to a cluster. The goal is not to say "oh, maybe it is this, and maybe it is that." A more precise formulation of this statement is taken up in a later section.

3.3.2 k-means clustering

Clustering, and in particular, k-means clustering, can be shown to be equivalent to matrix factoriztion.[7] In particular, the equations that define k-means as a relaxation problem, of aligning vectors to the closes centroids, can be written explicitly as matrixes. This equivalence is reviewed in the section after the next, after a sufficient amount of other mathematical devices have been set up. Most important of these is the choice of an appropriate information metric (instead of cosine distance) for the clustering norm, together with a justification of why this is necessary. This needs to be coupled to an appropriate mechanism for performing multiple class membership.

Once this is done, clustering can be re-interpreted as more of a graphical method, as opposed to an optimization method.

3.3.3 MST (Agglomerative) Clustering

MST clustering is a form of greedy clustering, attempting to first connect all points in the dataset with a tree that minimizes the the distance between the points, and then removing some of the longest edges, leaving behind a set of connected components. MST clustering can be fairly efficient, as one can find provisionally minimal trees with a fairly small number of distance evaluations, using a greedy algorithm; the provisional minimal tree can then be adjusted using local relaxation.

Grygorash *et al[8]* review multiple variants for deciding which edges to remove from an MST graph, and describe several particularly effective variants. Standard MST removes the longest edges; but one can instead remove edges that differ the most from thier neighbors; or one can remove edges that are outliers from the typical edge-length mean.

3.3.4 Non-issues

There are a variety of criticisms of different clustering algorithms, pointing at various drawbacks. Some of these criticisms do not apply to the current problem, and so are not to be used in selecting a better algorithm.

- Convex clusters. This is a standard criticism levelled against k-means clustering: the clusters can only ever be convex. This is important criticism, when working with low-dimensional data (2D, 3D data) where perhaps

most practical examples require clusters that are not convex. For the langauge learning problem, the data lives in an extremely high-dimensional space, with clusters that are almost surely convex.

3.4 Low Rank Matrix Approximation

Factorizations of the form of ean 1 are not uncommon in machine learning. They generally go under the name of Low Rank Matrix Approximation (LRMA). The rank refers to the size of the set G – it is the rank of the matrices in the factorization. The factorization is only an approximation to the original data; thus, one says LRMA and not LRMF.

Closely related is the concept of non-negative matrix factorization (NMF or NNMF),[9] where the focus is on keeping matrix entries positive, as would be appropriate for probabilities. Furthermore, a matrix of probabilities is not just non-negative; it also has non-negative rank; *viz.* every non-negative linear combination of the rows or columns must also be non-negative.

It is known that the factorization of non-negative matrices with non-negative rank is an NP-hard problem. Factorization can be seen as generalizing k-means clustering, which is known to be NP-complete.

A variety of techniques for performing this factorization have been developed. A lightning review is given below. The point of the review is less to edify the reader, than it is to point out which techniques, formulas and metrics are the most appropriate for the present situation, namely, eqn 1 for probabilities, coupled to the need for fairly crisp word-sense disambiguation.

3.4.1 Probabilistic Matrix Factorization

Probabilistic matrix factorization (PMF) assumes that the observation counts $N(w,d)$ are normally distributed (i.e. are Gaussian). The factorization is then obtained by minimizing the Frobenius norm of the difference of the left and right sides. That is, one defines the error matrix (or residual matrix)

$$E(w,d) = \left| p(w,d) - \sum_{g \in G} p(w,g)\, p(g,d) \right|$$

and from this, the objective function

$$U = \sum_{w,d} |E(w,d)|^2$$

After fixing the dimension $|G|$, one searches for the matrices $p(w,g)$ and $p(g,d)$ that minimize the objective function.

The primary drawbacks of probabilistic matrix factorization is that it does not provide any guarantees or mechanism to keep the factor $p(w,g)$ sparse. It's not built on information-theoretic infrastructure: it is not leveraging the idea that the p's are probabilities; it does not consider the information content of the

problem. From first principles, it would seem that information maximization would be a desirable property.

3.4.2 Nuclear Norm

Whenever the error matrix $E(w,d)$ can be decomposed into a set $\{\sigma_i\}$ of singular values, then the trace of the decomposition is $t = \sum_i \sigma_i$. The trace can be treated as the objective function to be minimized, leading to a valid factorization, differing from that obtained by PMF.

The word "nuclear" comes from operator theory, where the definition of a nuclear operator as one that is of trace-class, i.e. having a trace that is invariant under orthogonal or unitary transformations. In such cases, there is an explicit assumption that the operator lives in some homogeneous space,[10] where orthogonal or unitary transformations can be applied. In machine learning, the spaces are always finite dimensional, and are usually explicitly assumed to be real Euclidean space $\mathbb{R}^n$ – that is, the spaces behave like actual vector spaces, so that concepts like PCA and SVD apply.

A subtle point here is that the space in which $p(w,d)$ lives is *not* $\mathbb{R}^n$ (nor is it $\mathbb{R}^{|W|\times|D|}$, if one is a stickler about dimensions). Rather, $p(w,d)$ is constrained to live inside of a simplex (of dimension $|W| \times |D|$). Sure, one can blur one's eyes and imagine that this simplex is a subspace of $\mathbb{R}^{|W|\times|D|}$, and that is not entirely wrong. However, the only transformations that can be applied to points in a simplex, that keep the points inside the simplex, are Markov matrices. Any other transformations will typically move points into the inside from the outside, and move inside points to the outside. In particular, rotations (orthogonal transformations) cannot be applied to a probability, such that the result is still a probability. Applying the notion of a trace, which is implicitly defined as being invariant under orthogonal transformations, is inappropriate for the problem at hand. What would be appropriate is some sort of trace-like invariant that transforms as a scalar under Markov transformations.

3.4.3 Non-Negative Matrix Factorization

Non-negative matrix factorization (NMF) is similar to PMF, but with the additional constraint that the result has a non-negative rank. The non-negative rank constraint requires that not only do the factor matrixes have non-negative values in them, but also that non-negative linear combinations are also non-negative. This appears be an appropriate restriction for probabilities.

A reasonable review of NMF is given in[11], which also describes how one can control the sparsity of the resulting factors. Control over sparsity is one reason that clustering techniques are interesting; something as fast or faster that offers control over sparsity is appealing.

NNMF using the Kullback-Leibler divergence is equivalent to Probabilistic Latent Semantic Indexing (PLSI), although the two commonly used algorithms for each are quite different, and each is able to climb out of local minima of the other.[12]

The error term that needs to be minimized is the matrix-factorized form of the mutual information, which is just the Kullback-Leibler divergence between the factored and unfactored matrixes:

$$MI_{\text{factor}} = \sum_{w,d} p(w,d) \log \frac{p(w,d)}{\sum_{g \in G} p(w,g)\, p(g,d)} \tag{2}$$

In essence, this measures the information loss in moving from the full distribution $p(w,d)$ to the factorized version; for a good factorization, one wishes to minimize this information loss.

3.4.4 Neural Net Matrix Factorization

The factorization

$$\sum_{g \in G} p(w,g)\, p(g,d)$$

can be viewed as just one special function of the vector components indexed by g. More generally, one can consider the function

$$f(a_1, a_2, \cdots, a_n)$$

where $a_g = p(w,g)\, p(g,d)$ and $n = |G|$ the number of elements in G. Thus, the matrix factorization is just the function $f(a_1, a_2, \cdots, a_n) = a_1 + a_2 + \cdots + a_n$. The neural net matrix factorization[13] replaces the simple sum by a multi-layer feed-forward neural net.

The loss of linearity in this model seems like a rather extreme proposal. The fact that it is effective may in fact be a symptom of unknown, underlying structure. For example, there is nothing in the current formulation of the natural language problem that suggests this as an appropriate model of language. See, however, the next section, on factorization ambiguity, that suggests that the "shape" of language consists of a tight, highly-interconnected nucleus, attached to sparse feeder trees. That nucleus itself has additional structure. It might be the case that this complex structure can be captured by an *ad hoc* model consisting of some non-linear function f. However, in the end, this just suggests that the function f is just hiding or modelling or leaving unexplored some deeper linear structure.

3.4.5 Local Low Rank Matrix Factorization

Local Low Rank Matrix Factorization (LLORMA)[14] is a matrix factorization algorithm exhibiting accuracy and performance at near state-of-the-art levels. It uses a combination of two techniques: kernel smoothing[15] and local regression (LOESS)[16] to obtain smooth estimates for the two factors $p(w,g)$ and $p(g,d)$.

These two techniques, combined, prove to be almost ideal, when faced with incomplete data, and when the data is noisy. Specifically, the idea of incompleteness is that some values of the input dataset $p(w,d)$ are zero not because

they fundamentally should be (i.e. are forbidden by the grammar and syntax of natural langauge), but are zero simply because they have not yet been observed; some appropriate sentence does not occur in the sample dataset.

Thus, the use of the smoothing techniques seems highly appropriate, given that the input dataset $p(w, d)$ is both noisy and incomplete, as discussed in section 3.1.1. Unfortunately, the use of LLORMA, as strictly described, seems inappropriate, but only because the use of the Frobenius norm or the nuclear norm is inappropriate for this dataset. The correct norm must be that of eqn 2.

3.4.6 Other factorizations

There are other techniques for factorization, including

- NTN (Neural Tensor Network)
- I-RBM (Restricted Boltzmann Machine)
- I-AutoRec

These are not reviewed here.

3.5 Clustering as Matrix Factorization

One may show that k-means clustering is equivalent to a rank-k matrix decomposition with an extra orthogonality condition enforced; this is developed by Ding *et al.*[7] The orthogonality condition can be loosened, as the process of factorization is driven by a minimization condition that drives towards approximate orthogonality. In effect, k-means clustering is a special case of matrix factorization: its factorization with extra constraints.

Ding *et al* examine several forms of k-means clustering. The simplest form of clustering assigns vectors to clusters, and stops there. This is equivalent to assigning words to word-classes, without making any specific statements about what happened to the disjuncts. Alternately, one could say that the disjuncts just went along for the ride: they were associated with some word before-hand, and they are now still associated with that word, thrown into a bucket with the other disjuncts of similar words.

Bipartite graph clustering (aka "co-clustering") recognizes that the input data can be viewed as a bipartite graph (fig 1, left image), and that one can perform seperate, distinct, but simultaneous clustering on the columns, and seperately, the rows. This is closer to the desired factorization model for language, so the proof is reviewed here. It is still k-means clustering, just that one performs two clusterings, not one, on the columns, and on the rows.

The matrix to be factorized is B and the desired factorization is $B \approx LR^T$. Comparing this to the factorization of eqn 1, the matrix elements of B are $p(w, d)$; those of L are $p(w, g)$; those of R^T are $p(g, d)$. Key to the proof is the reinterpretation of L and R as membership matrixes, so that each row and

column indicate the membership of a vector to a cluster. That is L consists of column vectors $L = \left[\vec{l}_1, \vec{l}_2, \cdots, \vec{l}_k\right]$ where each column vector is of the form

$$\vec{l}_j = (0, 0, \cdots 0, 1, 1, \cdots, 1, 0, \cdots, 0)^T$$

with the 1's indicating the cluster membership. That is, the matrix elements of L are

$$L_{ij} = \begin{cases} 1 & \text{item } i \text{ belongs to cluster } j \\ 0 & \text{otherwise} \end{cases}$$

Equivalently, the factorization is

$$p(w|g) = \begin{cases} 1/|g| & \text{word } w \text{ belongs to wordclass } g \\ 0 & \text{otherwise} \end{cases}$$

The normalization is such that every word belongs to some class, with 100% probability (hard clustering). The matrix R is presented analogously.

The optimization problem is then to find the maxima of two objective functions, subject to some constraints:

$$\max_{L \geq 0; L^T L \sim I} \text{tr} L^T B B^T L \quad \text{and} \quad \max_{R \geq 0; R^T R \sim I} \text{tr} R^T B^T B R$$

The constraint $L^T L \sim I$ means that not only should $L^T L$ be a diagonal matrix, but that it should be a multiple of the identity matrix I – that is, have equal values along the diagonal. Here, the tr operator is the matrix trace. The trace of a matrix is, of course, equal to the sum of the eigenvalues of the matrix; thus, optimizing the above is equivalent to performing a singular value decomposition (SVD) of the traced matrix, and then summing the singular values. This is, of course, just the nuclear norm discussed previously.

With some relatively straightforward algebraic manipulation, the above can be shown to be equivalent to optimizing

$$\min \left\| B - LR^T \right\|^2$$

subject to the same constraints. The norm $\|\cdot\|^2$ is the Frobenius norm. In this sense, hard k-means clustering is identical to matrix factorization. Removing some of the constraints shows that k-means clustering is a special case of the more general factorization problem. Ding *et al* show that if one removes the orthogonality constraints, the resulting factors are still approximately orthogonal, since the Frobenius norm contains terms that drive the factors towards orthogonality.

Three previously identified issues arise with the above:

- The hard-clustering assignment of a word to only a single word-class prevents words from having multiple meanings, and thus forces word-sense disambiguation to somehow happen somewhere else.

- The use of the Frobenius norm (or the nuclear norm) implicitly forces assumptions of rotational invariance, in the form of orthogonality constraints on the membership indicator matrixes L, R. As discussed previously, rotational invariance (and thus, orthogonality) is inappropriate when L and R are interpreted as joint probability distributions.

- There is no room for a central factor matrix $M(g, g')$ which can capture the non-sparse complexities of the language. The need; indeeed, the inevitability of such a matrix is developed in the next section.

The proposed fix to these issues is multi-fold:

- Use an information-theoretic metric, namely, the Kullback-Leibler divergence of the factored solution to the input data.

- Perform greedy clustering, so as to minimize the number of number of non-zero entries in the left and right factor matrixes

- Decompose vectors into components that align with clusters, so that clusters correspond to word-senses, and word-sense disamgiguation (WSD) is an inherent, inbuilt part of the clustering step.

3.6 Factorization Ambiguity

When considering the factorization of eqn 1, the sum $\sum_{g \in G}$ can be seen as a specific function, *viz*, the inner product of two vectors $(\vec{w})_g = p(w, g)$ and $\left(\vec{d}\right)_g = p(g, d)$. Aside from considering just the function $f\left(\vec{w}, \vec{d}\right) = \vec{w} \cdot \vec{d}$, one might consider other functions $f\left(\vec{w}, \vec{d}\right)$ of $\vec{w}$ and $\vec{d}$. For example, a single-layer feed-forward linear neural net would consist of a $|G| \times |G|$-dimensional weight matrix M such that

$$f\left(\vec{w}, \vec{d}\right) = \vec{w}^T \cdot M \cdot \vec{d}$$

This, in itself, because it is linear, does not accomplish much, because the matrix M can be re-composed on the left or the right, to re-define the vectors $\vec{w}$ or $\vec{d}$. That is, one may write $\vec{w}' = M^T \vec{w}$ to get a different product $\vec{w}' \cdot \vec{d}$, or, alternately $\vec{d}' = M\vec{d}$ for a product $\vec{w} \cdot \vec{d}'$. The dot-product in the factorization is ambiguous; the point is that the factorization of eqn 1 is not unique.

The low-rank matrix-factorization literature expresses this idea by noting that a dot-product is invariant under orthogonal rotations, and so one can choose an arbitrary orthogonal matrix O with $O^T O = I$ and write

$$\vec{w} \cdot \vec{d} = \left(\vec{w} O^T\right) \cdot \left(O\vec{d}\right) = \vec{w}' \cdot \vec{d}'$$

Since the vectors $\vec{v}$ and $\vec{d}$ are naturally probabilities, the above is exactly what we do *not* what to do! As already noted, orthogonal rotations applied to a probability turn it into something that is not a probability. Instead, we want

to stick to a single (ambiguous) matrix M that is Markovian, so that, when contracted to the left or to the right, the resulting vectors are still probabilities.

This becomes more clear if written in in components:

$$p(w,d) = \sum_g \sum_{g'} p(w,g)\, M(g,g')\, p(g',d)$$

This shows that the factorization is ambiguous; as long as M is Markovian, preserving the sums of probabilities over rows and columns, it can be contracted to the left or the right. Indeed: M itself can be factored into an arbitrary product of Markovian matrixes, which can then be merged to the left and right.

Thus, to get a meaningful factorization, one can must introduce additional constraints. A seemingly natural one, to be developed later, is to choose M such that $p(w,g)$ and $p(g',d)$ are both maximally sparse. One would like to assign a word to at most a handful of different word-classes, corresponding to the synonym classes for each word-sense attached to that word.

A quick review of the concept of a Markovian matrix is in order. A matrix can be Markovian on the left side, the right side, or both. It is Markovian on the right if

$$1 = \sum_g M(g,g') \text{ for all } g'$$

This assures that a transformed probability

$$p'(g,d) = \sum_{g'} M(g,g')\, p(g',d)$$

is still a valid probability distribution; namely, that $p'(*,*) = 1$.

If one has such a factorization, so that $p(w,g)$ and $p(g',d)$ are maximally sparse, then the matrix M will likely be very highly connected, i.e. will have many or most of its matrix entries be non-zero. Conceptually, one can visualize the matrix M as a highly connected graph, while the factors $p(w,g)$ and $p(g',d)$ are low-density feeder tree-branches that connect into this tightly-coupled central component. This is visualized in figure 1.

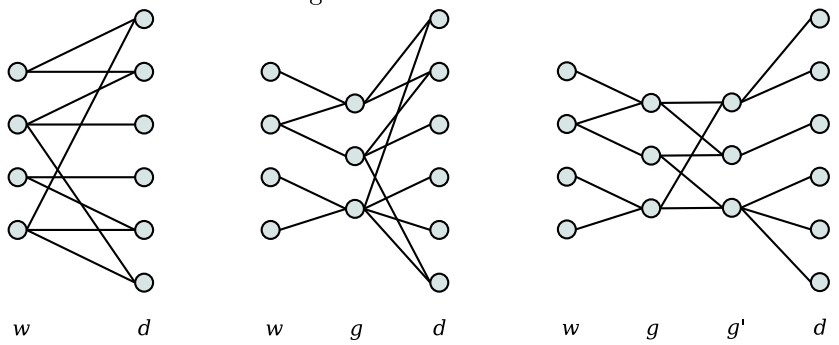
In the above factorization, matrix M was made explicitly Markovian, so as to preserve the left and right factors as joint probabilities. However, it seems to be particularly important to the semantic structure of the language: it captures the complexity of language, whereas the left and right factors merely funnel spelled-out word-strings into the actual semantic cagtegories. Thus, it is convenient to instead write the left and write factors as Markov matrices, while taking the central factor to be a joint probability. This can be acheived by factoring as

$$p(w,d) = \sum_g \sum_{g'} p(w|g)\, p(g,g')\, p(d|g') \tag{3}$$

where the left and right factors are conditional probabilities. That is,

$$p(w|g) = \frac{p(w,g)}{p(*,g)}$$

Figure 1: Factorization



This figure attempts to illustrate the process of factorization. The left-most image is meant to illustrate $p(w,d)$ as a sparse matrix. Edges indicate those values where $p(w,d)$ is not zero. Not every w is connected to every d, but there are a sufficient number of connections that the overall graph is confused and tangled. The middle image is meant to illustrate the factorization $\sum_g p(w,g)\,p(g,d)$. In this factorization, the matrix $p(w,g)$ not only becomes more sparse, but has a very low out-degree for fixed w: only one or a handful of entries in $p(w,g)$ are non-zero for fixed w. The rightmost image attempts to illustrate the factorization $\sum_{g,g'} p(w,g)\,M(g,g')\,p(g',d)$. Here, the factor $p(g',d)$ has low in-degree for any fixed d. All of the tangle and interconnectedness has been factored out into the matrix $M(g,g')$ connecting word-classes to disjunct-classes.

In hard-clustering, these low-degree requirements are automatically satisfied: a word w can belong to only one word-cluster g, and so there is only one line in the figure connecting w to anything. Likewise, a disjunct d can only be assigned to one cluster g'. Due to the fact that words are combinations of word-senses, hard-clustering in this fashion is undesirable; by contrast, it seems that word-senses could be validly hard-clustered.

---

and likewise for $p(d|g') = p(g', d)/p(g', *)$. These are explicitly Markovian, in that

$$\sum_g p(w|g) = 1$$

and likewise $\sum_{g'} p(d|g')$. This factorization is just a rescaling of the earlier factorization:

$$p(g, g') = p(*, g) M(g, g') p(g', *)$$

and is done so that $p(g, g')$ can be seen as a joint probability:

$$\sum_{g,g'} p(g, g') = 1$$

The ambiguity of the matrix M is removed, if one assumes hard clustering. In this case, each w can belong to only one g, and each d to just one g', and, once the clusters are chosen, there is no confusion about the left and right factors, and thus, the central factor is fixed.[2]

A factorization of this sort is known as co-clustering, bi-clustering or sometimes block clustering. An explicit development of this, including an explicit iterative algorithm guaranteed to converge, is given by Dhillon et al.[17] and is reviewed in a subsection below. Of course, for natural langauge, we want to decompose words into word-senses, and so naive hard-clustering will not work. It does, however, illuminate the path ahead.

3.6.1 Factorization in Link Grammar

This factorization is *de facto* observed in the hand-built dictionaries for Link Grammar. Examination of the `4.0.dict` file in the Link Grammar file distribution will clearly show how words are grouped into word-classes. For example, `words.n.2.s` is a list of plural count-nouns. The Link Grammar costs are very rough approximations for the log probability $-\log p$, and so the contents of `words.n.2.s` is effectively a representation of the matrix $p(w, g)$ for $g = \langle$plural-count-nouns$\rangle$ and a uniform probability across this class. The file `4.0.dict` also defines a large number of "macros", with names such as `<noun-main-p>`. These macros are stand-ins for lists of disjuncts, often given equal weight, but also not uncommonly assigned different costs. In essence, `<noun-main-p>` should be understood as an example of $p(g', d)$ for $g' = \langle noun - main - p \rangle$. It appears multiple times throughout the file. In one case, it is associated with the word-list `words.n.2.s`, which makes sense, as `<noun-main-p>` is describing one of the linkage behaviors of plural common nouns. The contents of the file `4.0.dict` should be understood to be a specification of $M(g, g')$, although it is not so cleanly organized: it also includes all of the "macros" $p(g', d)$ and also includes word-lists when these are small.

[2] In physics, such an ambiguity of factorization is known as a global gauge symmetry; fixing a gauge removes the ambiguity. In natural language, the ambiguity is sponteneously broken: words have only a few senses, and are often sysnonymous, making the left and right factors sparse. For this reason, the analogy to physics is entertaining but mostly pointless.

A more careful examination of the use of the macros in `4.0.dict` shows that these are often cascaded into one-another. For example, `<noun-main-s>` is used in the definition of `<proper-names>`, `<entity-entire>` and `<common-noun>`, each of which service word classes that are similar and yet differ syntactically and semantically. This is not just some strange artifact of hand-building a dictionary encoding grammar. It is *prima facie* evidence of important substructures inside of $M(g, g')$, essentially pointing at the idea that $M(g, g')$ can be further factored into smaller, tightly-connected blocks; *viz.*, that the graph of $M(g, g')$ contains strongly-coupled bipartite cliques.

This explicit co-clustering in Link Grammar is not driven by any sort of theoretical arguments or foundation. Rather, it is a natural, intuitive outcome of how the linguists who author the dictionaries wish to tackle the problem. A good factorization saves time and effort for the author, and is easier to debug. The urge to factorize is not limited to English: a look at any of the dictionaries shows this structure clearly. It becomes even more pronounced in dictionaries with morphology, e.g. in the Russian dictionaries, where word-stems (which carry most of the meaining) are factored away from the suffixes (which provide the needed tense, gender, number and person agreement across sentences).

3.6.2 Tensor Product Factorization

The idea that $M(g, g')$ contains important substructures is important enough to point out a second time. Based on explicit experience with Link Grammar, those substructures are likely to require a tensor product factorization (as opposed to a matrix product factorization) to make sense of them. Its possible to speculate that a Tucker factorization may provide a reasonable first approximation to this factorization. Nothing further on this can be said at this point, until a reliable way to obtain a stable, reproducible and accurate $M(g, g')$ is in hand.

3.6.3 Rank and Dimension

Based on the example of the actual English lexis in Link Grammar, there is no need to assume that the number of word classes $|G|$ should someohow be equal to the number of syntactic usage patterns $|G'|$. Indeed, the dimension of the matrix $M(g, g')$ has to be $|G| \times |G'|$; but these two dimensions are not known from any *a priori* principles.

More careful vocabulary is needed here: one can say that $|G|$ is the number of "word classes". The number of of syntactic usage patterns $|G'|$ could be called the number of "grammatical classes", although historic usage conflates these two terms. Thus, the term "syntactic classes" for G' seems the most appropriate.

The number of word classes $|G|$ must surely be fairly high, as they must capture not only the predicate-argument structure,[18, 19] but the resulting syntax constraints must force the selection of the predicate-argument structure.[20] Roughly speaking, the number of word classes should correspond to the number of different synonym classes one might expect to find. This is confused by the situation of common nouns: there are vast numbers of these, and while most

are not synonyms, most are syntactically interchangable, even when forcing predicate-argument agreement.

The appropriate number of syntactic classes $|G'|$ is presumably a lot lower. At a minimum, it corresponds to the number of classical head-phrase structure grammar non-terminals, such as S, NP, VP, PP, D, A, V, N, *etc.* A more complete set can be found in the dependency grammar relations subj, obj, iobj, det, amod, advmod, psubj, pobj, *etc.* but even this seems too low. There are just over 100 different link types in Link Grammar, growing to the thousands, when one considers various subtypes (subscripts). However, the number of distinct macros in the English lexis provides a different lower bound. At any rate, the size of $|G'|$ cannot be smaller than 100 for a realistic model of the English language, and an accurate model is likely to require $|G'|$ of at least a few thousand.

3.6.4 Akaike Information Criterion

How many word classes and syntactic classes should there be? Aside from making various *a priori* guesses, one can apply the Akaike information criterion (AIC). Essentially, the grammatical classes can be taken as the parameters of the model. The AIC can be used as a guide to determine how many of them are required. This is easy to say in principle; a computationally efficient mechanism is not yet clear.

3.6.5 Removing Ambiguity in Factorization

As noted in the earlier subsection, the ambiguity in the factorization can be removed by hard clustering. Practical experience with hands-on similarity measures in language data indicate that there should not be much of a problem: most words that are similar are obviously-so, using an appropriate pair-wise similarity function.

Suppose this was not easily the case? To guide the factorization, and to maximize the sparsity of the left and the right factors, while maximizing the complexity of the central factor, one can appeal to Tegmark's formulation of Tononi integrated information as a guide. That is, one wishes to factorize in such a way that the total amount of integrated information in the left and right factors are minimized, while the integrated information of the central factor is maximized.

In essence, factorization is a reorganizeation of the graph so as to always maximize the integrated information of an important central core. All edges where mutual information is weak are to be pruned away.

3.6.6 Information Loss

The goal of the factorization is to minimize the information loss between the input data and the factorization. The objective function is then The Kullback-

Leibler divergence, a minor variant on the previous eqn 2:

$$MI_{Loss} = \sum_{w,d} p(w,d) \log_2 \frac{p(w,d)}{\sum_{g \in G} \sum_{g' \in G'} p(w|g)\, p(g,g')\, p(d|g')} \tag{4}$$

By minimizing this divergence, one minimizes the total loss incurred by the factorization.

The above information loss estimate is identical to that described by Dhillion *et al.*[17] in thier treatment of hard co-clustering. That reference provides an extensive and detailed review of the factorization problem being addressed in this section, with the exception that the current need for word-sense disambiguation violates thier hard-clustering assumption. Words cannot be hard-clustered; word-vectors must be decomposed into word-sense vectors first.[21] Nonetheless, many formulas are still relevant, and the reference gives detailed motivation for them, and provides multiple articulations and derivations. Various results are recapped here.

For the special case of hard clustering, where each word or disjunct is assigned to only one word class/grammatical class, one has that (eqn (6) if Dhillon)

$$p(g,g') = \sum_{w \in g} \sum_{d \in g'} p(w,d) \tag{5}$$

From this, it follows that (eqn (4) of Dhillion)

$$MI_{Loss} = MI(W,D) - MI(G,G')$$

and so eqn 4 really is the information loss from factorization. For hard clustering, the loss can be written in terms of only the left, or the right clusters, so that (lemma 4.1 of Dhillon)

$$\begin{aligned} MI_{Loss} &= \sum_g \sum_{w \in g} p(w,*) \sum_d p(d|w) \log_2 \frac{p(d|w)}{p(d|g)} \\ &= \sum_g \sum_{w \in g} \sum_d p(w,d) \log_2 \frac{p(w,d)}{p(w,*)} \frac{p(g,*)}{p(g,d)} \end{aligned}$$

This allows an iterative algorithm to be performed, clustering only rows (or only columns), that is, only words (or only disjuncts).

3.6.7 Biclustering

It is worth reviewing the algorithm that Dhillion *etal* present. It is an iterative hill-climbing algorithm, alternating between three steps: the computation of marginals, and the assignment of new row clusters, and the assignment of new column clusters. The marginals are recomputed after every reclustering. Dhillon provides a proof that, for hard clustering, the information loss is monotonically decreasing; viz, that iteration always moves to a better solution.

Given provisional cluster assignments G and G', so that every word and every disjuncts can be placed into some specific cluster, all three factors $p(w,g)$, $M(g,g')$ and $p(g',d)$ are available (are computable) given the cluster assignments. The central factor is given by eqn 5. The left and right factors are obtained as marginals:

$$p(w,g) = \begin{cases} p(w,*) & \text{if } w \in g \\ 0 & \text{otherwise} \end{cases}$$

and

$$p(g',d) = \begin{cases} p(*,d) & \text{if } d \in g' \\ 0 & \text{otherwise} \end{cases}$$

The above are always known and well-defined, since, by assumption, the provisional cluster assignments G and G' are known at each step of the iteraction. The conditional probabilities are as noted before:

$$p(w|g) = \frac{p(w,g)}{p(*,g)}$$

and likewise

$$p(d|g') = \frac{p(g',d)}{p(g',*)}$$

where $p(*,g) = \sum_{w \in g} p(w,g)$ and $p(g',*) = \sum_{d \in g'} p(g',d)$.

The new cluster assignments are obtained by minimizing the information loss, once for the rows, and once for the columns. In one step, one searches for cluster assignments $w \in g$ such that

$$MI_{word\,loss} = \sum_d p(d|w) \log_2 \frac{p(d|w)}{p(d|g')\,p(g'|g)}$$

is minimized. After recomputing marginals, one then reclusters the disjuncts, by seaching for the culstering $d \in g'$ that minimizes the loss

$$MI_{disjunct\,loss} = \sum_w p(w|d) \log_2 \frac{p(w|d)}{p(w|g)\,p(g|g')}$$

This looks like an entirely reasonable algorithm, concrete and specific and implementable, until one refers back to the table in section 3.1. There are in excess of 10^5 words to provisionally assign to clusters, and 10^7 or more disjuncts. Each provisional clustering requires extensive recomputation of marginal probabilities. Exhaustive search for the best clustering clearly cannot scale to the current datasets. One might be able to make some forward progress by means of genetic algorithms, as one is performing hill-climbing on a well-defined utility function. The hard cluster membership can be encoded as a very long bit-string: this is exactly the scenario for which genetic algorithms were designed to solve: optimizing large bit strings, given a utility function on them.

But never mind: the assumption of hard clustering breaks word-sense disambiguation. Generic algorithms are not enough. Back to the drawing board.

3.7 Graph Algorithms vs. Gradient Descent

In the previous section, the Word2Vec style algorithms were characterized as "gradient descent algorithms", and were contrasted with "graph algorithms" that explicitly and overtly focus on the graphical relationships between items. The same contrast can be made here: clustering is a form of a graph algorithm, whereas the matrix factorization algorithms are all driven by a form of gradient descent.

The difference can be highlighted by noticing that cluster assignment is essentially a form of greedy algorithm: One looks for the best-fitting cluster, and accepts that first, effectively ignoring all of the other clusters. The relationship of word-to-cluster is explicit and overt; by contrast, the matrix factorization is only implicit: a cluster relationship exists whenever a matrix element is large.

In the current state of the art, the gradient-descent algorithms tend to outperform the clustering algorithms, when the size of hidden layers is limited to a computationally tractable size. This can be interpreted in several ways: for small hidden layers, the clustering algos just might be "too greedy", applying too strong a discrimination. Gradient descent algorithms also organize compute cycles differently, removing certain repeated calculations out of a tight loop.

One advantage of the clustering approach is that, since it makes the graph structure explicit, it provides a mechanism for controlling the shape of the graph. Another is that it seems to perhaps be more scalable, when the size of the hidden layers are not suitably small.

3.7.1 Control of Graphical Structure

The goal of the matrix factorization, illustrated in figure 1, is to not only obtain the three factors $p(w,g)$, $M(g,g')$ and $p(g',d)$, but to obtain them such that the first and last components contain essentially no bipartite cliques, and all of the structural complexity is limited to $M(g,g')$. A naive gradient descent does not seem to achieve this; it will provide numerical values, but will not go out of its way to maximally set as many of the $p(w,g)$ values to zero as possible. In keeping with the general trend: the goal here is to set as many matrix entries to zero as possible, thereby getting the greatest amount of data compression as possible. Yet, fidelity has to be maintained: just the right entries should be non-zero.

The point here is that it is the graph structure that is the most important; the actual numerical values associated with each edge are far less important. They are nice, and useful for improving accuracy and parse ranking, but provide little insight in and of themselves. The graph structure dominates. This is made clear in section 3.6.1, where a huge amount of progress can be made by means of manual factorization, and setting all edge weights to essentially just one value: present or absent.

Greedy clustering essentially provides a mechanism for controlling this structure: it inherently limits the number of grammatical classes that a word can belong to. It dis-incentivizes the partitioning of a word into a large number of

grammatical categories.

4 Factorizing the language model

The above developments now provide enough background to clearly state the problem. Inspired by the factorization of eqn 1, one wishes to find a collection of word classes, assigning words to a handful of classes, according to the in-context word-sense. The proper factorization needs to be of the form of eqn 3, as illustrated in figure 1. The appropriate measure of quality is to minimize the information loss in the factorization, as given by eqn 4.

More precisely, the factorization of the language model appears to require three important ingredients:

- A way of decomposing word-vectors into sums of word-sense vectors,
- A way of performing biclustering, so as to split the bipartite graph $p(w, d)$ into left, central and right components, holding the left and right parts to be sparse,
- Using an information-theoretic similarity metric, to preserve the probabilistic interpretation of the contingency table $p(w, d)$.

Combining all these parts in one go is daunting. Smaller steps towards the ultimate goal can be taken. One easy first step is to perform clustering using an information metric, instead of the cosine distance. This is done in the section below.

A second step is to replace hard clustering by an algorithm that treats word-vectors as sums of (as yet unknown) distinct word senses. Because this is a substantial topic in itself, this is handled in a distinct tract; see [21].

4.1 Information-theoretic Clustering

The previous section uses the words "information theoretic" and "clustering" in close proximity. This is a "thing", and so a lighting review of the literature is warrented. Two observations pop out:

- None of the systems dscribed in the literature assume that the input data can already be interpreted as a probability; rather, the clustering is being performed on data naturally occuring in some Euclidean space, typically, some space of low dimension (e.g. two-dimensional image data).
- Most approaches to information-theoretic clustering use the mutual information between members of a cluster and the cluster label. Unfortunately, this has an ambiguity: the information content is conserved by any Markov matrix that reassigns the cluster labels, so, for example, a permutation matrix that just shuffles the cluster labels. Ver Steeg *et al.[22]* propose a solution to this ambiguity: a return to first principles, requiring that information loss due to coarse-graining be minimized.

Several other notable points are discussed below.

4.1.1 Renyi Information Divergence

Gokcay and Principe[23] propose the Renyi entropy as an information divergence. It is essentially minus the logarithm of the cosine metric. Given two vectors $\vec{u}$ anad $\vec{v}$, the information divergence is defined as

$$D_{CEF}(\vec{u},\vec{v}) = -\log\cos(\vec{u},\vec{v}) = -\log\hat{u}\cdot\hat{v} = -\log\frac{\vec{u}\cdot\vec{v}}{\|\vec{u}\|\,\|\vec{v}\|}$$

where $\|\cdot\|$ is the l_2-norm. Although this is widely cited in the literature pertaining to information-theoretic clustering, there is no particular reason to beleive that it offers any advantage at all over the ordinary cosine metric, when applied to the task of clustering grammatical categories. In particular, it suffers from the same defects previously identified: it contains an inbuilt assumption that the data presents in a Euclidean, rotationally symmetric space, which is very much not the case for the langauge data. The vectors of in the contingency table that the language data is ogranized in are not Euclidean vectors: they are joint probabilities. They transform not under orthogonal rotations, but under Markov matrices.

There are also some practical, language-related reasons to lose interest in the above information divergence: When examining actual language data, as is done in [24], it becomes quite clear that there is a practical, common-sense cutoff for similary. If two words w_1 and w_2 have a similarity of $\cos(w_1,w_2) \lesssim 0.5$, they are very nearly unrelated; for a similarity of $\cos(w_1,w_2) \lesssim 0.1$, they are fantastically unrelated. It would be crazy to assign anything with such low similarities to a commn cluster. Extending the notion of similarity to truly wee values, as the logarithm enables, is meaningless. For cosine values greater than $1/2$, the logarithm is essentially linear: $-\log\cos x \approx 1-\cos x$ in the region of interest. The appearence of the logarithm does nothing to advance the situation.

4.1.2 Consistency Under Coarse-graining

To deal with the problem of factorization ambiguity, Ver Steeg *et al[22]* revert to first principles, and propose that the information divergence should be approximately invariant under the coarse-graining of the input data.

There is some appeal in this idea for the langauge problem: a large cluster of approximate synonyms should be factorizable into smaller clusters of more tightly-related synonyms. Conversely, one should be able to consolidate small blocks into bigger ones, with a minimum of information loss. However, based on the explorations in [24], this does not appear to be a problem that needs to be externally forced onto the system. Pair-wise word similarities seem to be of high quality; it would take some work to have this wrecked by the clustering algorithm. Still ... additional consideration might be warranted.

4.2 Information-driven Clustering

As shown by Ding *et al*[7] and reviewed in section 3.5, k-means clustering can be seen as a special case of matrix factorizing. This section generalizes to an information metric, as opposed to the usual cosine metric. The first subsection reviews the simpler single-sided form of clustering, as it would be applied to just words, instead of the bipartite clustering. The second subsection presents the information metric.

4.2.1 Cosine clustering

The usual metric for judging similarity is the cosine metric:

$$\cos(\vec{u}, \vec{v}) = \hat{u} \cdot \hat{v} = \frac{\vec{u} \cdot \vec{v}}{\|\vec{u}\| \, \|\vec{v}\|}$$

where $\|\cdot\|$ is the l_2-norm. If the input data is taken as a collection of (row) vectors, then the input can be viewed as a matrix $X = [\vec{v}_1, \vec{v}_2, \cdots, \vec{v}_n]$. The product matrix $S = XX^T$ then has matrix entries $S_{ij} = \vec{v}_i \cdot \vec{v}_j$. If the vectors are already normalized, then the matrix entries S_{ij} are just the cosine distances between i and j. This is used to accomplish k-means clustering by finding a $n \times k$ matrix membership indicator matrix F such that the objective function

$$U = \mathrm{tr} F^T S F$$

is minimized. Some basic algebra (again, see [7]) shows that this is equivalent to minimizing the Frobenius norm

$$U = \left\| X - FF^T \right\|^2$$

which in turn is identical to minimizing the distance between cluster centroids $\vec{m}_i$ and each member $j \in C_i$ of the i'th cluster.

$$U = \sum_{i=1}^{k} \sum_{j \in C_i} \|\vec{v}_j - \vec{m}_i\|^2$$

The centroid $\vec{m}_i$ is the arithmetic mean of all of the vectors in the i'th cluster:

$$\vec{m}_i = \frac{1}{|C_i|} \sum_{j \in C_i} \vec{v}_j$$

where $|C_i|$ is the number of members in the i'th cluster.

4.2.2 Information clustering

The analog to cosine clustering can be arrived at with two changes. The cosine was built from the dot product $\vec{u} \cdot \vec{v}$ normalized by the vector lengths. Here, the

normalization is changed, so as to use the Kullback-Leibler divergence:

$$MI(\vec{u},\vec{v}) = \log_2 \frac{\vec{u}\cdot\vec{v}\left(\sum_{i=1}^{n}\sum_{j=1}^{n}\vec{v}_i\cdot\vec{v}_j\right)}{\left(\sum_{i=1}^{n}\vec{u}\cdot\vec{v}_i\right)\left(\sum_{j=1}^{n}\vec{v}_j\cdot\vec{v}\right)}$$

This unusual-looking expression can be made more recognizable by changing notation back to the joint probabilities $p(w,d)$ of observing word w with disjunct d (of course, this would also work if the disjunct d was replaced by the N-gram context of w). Write for $S_{ij} = \vec{v}_i \cdot \vec{v}_j$ in the equivalent form

$$S(w_1,w_2) = \sum_d p(w_1,d)\, p(w_2,d)$$

The information divergence $MI(\vec{u},\vec{v})$ is then

$$MI(w_1,w_2) = \log_2 \frac{S(w_1,w_2)\, S(*,*)}{S(w_1,*)\, S(*,w_2)}$$

where, as always, the stars $*$ denote the wild-card sums:

$$S(w_1,*) = \sum_{w_2} S(w_1,w_2)$$

By normalizing, one gets an expression that looks just like a joint probability:

$$Q(w_1,w_2) = \frac{S(w_1,w_2)}{S(*,*)}$$

with

$$MI(w_1,w_2) = \log_2 \frac{Q(w_1,w_2)}{Q(w_1,*)\, Q(*,w_2)}$$

For the clustering operation, one must replace the arithmetic mean by the geometric mean. The cluster centroid is now given by

$$m(w,d) = \sqrt[\frac{1}{|g|}]{\prod_{w\in g} p(w,d)}$$

The need for the geometric mean can be thought of in two ways: first, probabilities can only be added if they are truly independent, and here they are not; but probabilities can always be multiplied. More importantly, the logarithm of the geometric mean can be taken. This connects the pair-wise information distance to minimization of cluster sizes.

4.3 WSD-sensitive clustering

Since this paper is already too long, that conversation is moved over to reference [21]. Its bare-bones right now.

4.4 Information Maximizing Matrix Factorization

xxxx

Everything below here is incoherent and incomplete. Sorry; under construction.

4.4.1 Surprisal Analysis

The basic idea is this forumla:

$$-\log \frac{P(x)}{P_0(x)} = \sum_\alpha \lambda_\alpha G(\alpha)$$

The G's are the constraints. The λ's are the Lagrange multipliers. When there is only one G, then it is conventionally called "the energy", and the λ is called "the (inverse) temperature".

4.4.2 Partition function

, although it requires some background theory.

Consider again the nature of the hidden layer (the layer in which the vectors live) in the CBOW/SkipGram model. The "softmax" formulation allows the conditional probabilities to be written a certain way; the corresponding unconditional probability is

$$p(w_t, w_I) = \frac{\exp \sum_{i \in I} \vec{u}_t \cdot \vec{v}_i}{\sum_{w \in W} \sum_{I \in W^N} \exp \sum_{i \in I} \vec{u}_w \cdot \vec{v}_i}$$

This quantity is not normally available or computed in the CBOW/Skipgram models; here, it is written here as a theoretical quantity. It can be obtained from the partition function

$$Z[J] = \sum_{w \in W} \sum_{I \in W^N} \exp \left(\sum_{i \in I} \vec{u}_w \cdot \vec{v}_i + J_{wI} \right)$$

by applying the variational principle to one of the many "sources" $J_{w_t,I}$. This yields the standard Boltzmann distribution:

$$p(w_t, w_I) = \left. \frac{\delta \ln Z[J]}{\delta J_{w_t,I}} \right|_{J=0}$$

In other words, CBOW/SkipGram fit squarely into the standard maximum entropy theoretical framework. This is no accident, of course; the "softmax" function was used precisely because it gives the maximum entropy distribution.

There is no particular reason to diverge from the principle of maximum entropy when working with word classes. Specifically, this suggests that the word-disjunct pairs be viewed in the frame of the "loss function" (equivalently, the Hamiltonian or energy):

$$H(w, d) = -\log p(w, d)$$

is suggests that the

There is a set of word-classes $C = \{c\}$ and two projection matrices π^W and π^D such that $\vec{\eta}_w = \pi^W \hat{e}_w$ is a vector that classifies the word w into one or more word-classes c. That is, $\vec{\eta}_w$ is a C-dimensional vector. In many cases, all but one of the entries in $\vec{\eta}_w$ will be zero: we expect the word $w =$ the to belong to only one class, the class of determiners. By contrast, $w =$ saw has to belong to at least three classes: the past-tense of the verb "to see", the noun for the cutting tool, and the verb approximately synonymous to the verb for cutting. The hand-built dictionary for English Link Grammar has over a thousand distinct word-classes; one might expect a similar quantity from an unsupervised algorithm.

The projection matrix π^D performs a similar projection for the disjuncts. That is $\vec{\zeta}_d = \pi^D \hat{e}_d$, so that each disjunct is associated with a C-dimensional vector $\vec{\zeta}_d$. Most of the entries in this vector will likewise be zero. This vector basically states that any give disjunct is typically associated with just one, or a few word classes. So, for example, the disjunct

$$\text{the}-$$

is always associated with (the class of) common nouns. The only non-zero entry in $\vec{\zeta}_{\text{the-}}$ will therefore be

$$<\text{common–nouns}>:\ \text{the}-;$$

Given these two projection matrices, the probability can then be decomposed as an inner product:

$$E(w,d) = \vec{\eta}_w \cdot \vec{\zeta}_d$$

The word-classes

Sheaves

The previous section begins by stating that, ideally, one wants to wants to model the probability $P(\text{sentence} \mid \text{fulltext})$, but due to the apparent computational intractability, one beats a tactical retreat to computing $P(\text{word} \mid \text{context})$ in the CBOW/SkipGram model, and something analogous in the Link Grammar model. However, by re-casting the problem in terms of disjuncts, however, one can do better. Dependency parsing allows one to easily create low-cost, simple computational models for $P(\text{phrase} \mid \text{context})$ or even $P(\text{sentence} \mid \text{context})$. This is because disjuncts are compositional: they can be assembled, like jigsaw-puzzle pieces, into larger assemblages. If this is further enhanced with reference resolution, one has a direct path towards a computationally tractable model of $P(\text{sentence} \mid \text{fulltext})$, with, at the outset, seemed hopelessly intractable.

TODO flesh out this section.

Gluing axioms

The language-learning task requires one to infer the structure of language from a small number of instances and examples. Bengio *etal.[3]* describe this for

continuous probabilistic models. First, one imagines some continuous, uniform space. Example sentences form a training corpus are associated with single points in this space: the probability mass is initially located at a collection of points. One then imagines that generalization consists of smearing out those points over an extended volume, thereby assigning non-zero probability weights to other "nearby" sentences. This suggests that there is a choice as to how this smearing-out is done: one can spread the probabilities uniformly, in all "directions", or one can preferentially spread probabilities only along certain directions. Bengio suggests that higher-quality learning and generalization can be achieved by finding and appropriately non-uniform way of smearing the probability masses from training.

This description seems like a useful and harmless way of guiding one's thoughts. But it leaves open and vague several undefined concepts: that of the "space": is this some topological space, perhaps linear, or something else? That of "nearby sentences": the presumption (the axiom?) that the space is endowed with a metric that measures distances. Finally, the concept of "direction", or at least, a local tangent manifold at each point of the space. It seems reasonable to assert that language lives on a manifold, but then, the structure of that manifold needs to be elucidated and demonstrated. In particular, the "non-uniform spreading" of probability weights suggests confusion or inconsistency: Perhaps the spreading appears to be non-uniform, because the initial metric assigned to the space is incorrect? In geometry, one usually works with normalized tangent vectors, so that when one extends them to geodesics, each geodesic moves with unit velocity. It seems plausible to spread out probability weights the same way: spread them uniformly, and adjust the shape of the underlying space so that this results in a high-quality language model.

References

[1] Daniel Sleator and Davy Temperley., *Parsing English with a Link Grammar*, Tech. rep., Carnegie Mellon University Computer Science technical report CMU-CS-91-196, 1991, URL *http* : *//arxiv.org/pdf/cmp* − *lg*/9508004.

[2] Daniel D. Sleator and Davy Temperley, "Parsing English with a Link Grammar", in *Proc. Third International Workshop on Parsing Technologies*, 1993, pp. 277–292, URL *http* : *//www.cs.cmu.edu/afs/cs.cmu.edu/project/link/pub/www/papers/ps/LG*− *IWPT*93.*ps*.

[3] Y. Bengio, et al., "A neural probabilistic language model.", *Journal of Machine Learning Research*, 3, 2003, pp. 1137–1155.

[4] Tomas Mikolov, et al., "Efficient Estimation of Word Representations in Vector Space", *arXiv/13013781v3*, 2013.

[5] Alex Minnaar, "Word2Vec Tutorial Part II: The Continuous Bag-of-Words Model", , 2015, URL *http* : *//mccormickml.com/assets/word2vec/Alex*$_M$*innaar*$_W$*ord2Vec*$_T$*utorial*$_P$*art*$_I$*I*$_T$*he*$_C$*ontinuous*$_B$*ag*$-$*of* $-$ *Words*$_M$*odel.pdf*.

[6] Deniz Yuret, *Discovery of Linguistic Relations Using Lexical Attraction*, PhD thesis, MIT, 1998, URL *http* : *//www2.denizyuret.com/pub/yuretphd.html*.

[7] Chris Ding, et al., "On the equivalence of nonnegative matrix factorization and spectral clustering", *Proc SIAM Data Mining Conf*, 2005, URL *http* : *//franger.uta.edu/ chqding/papers/NMF* $-$ *SDM*2005*.pdf*.

[8] Oleksandr Grygorash, et al., "Minimum spanning tree based clustering algorithms", *International Conference on Tools with Artificial Intelligence (2006)*, 2006, URL *https* : *//static.aminer.org/pdf/PDF*/000/219/731/*computing*$_h$*ierarchies*$_o$*f*$_c$*lusters*$_f$*rom*$_t$*he*$_e$*uclidean*$_m$*inimu*

[9] "Non-negative matrix factorization", *https* : *//en.wikipedia.org/wiki/Non* $-$ *negative*$_m$*atrix*$_f$*actorization*.

[10] "Homogenous space", *https* : *//en.wikipedia.org/wiki/Homogeneous*$_s$*pace*.

[11] Julian Eggert and Edgar Kørner, "Sparse coding and NMF", *Proc of the IEEE International Joint Conference on Neural Networks IJCNN 2004*, 2004, pp. 2529–2533, URL *https* : *//www.researchgate.net/publication/*4117104$_S$*parse*$_c$*oding*$_a$*nd*$_N$*MF*.

[12] Chris Ding, et al., "On the equivalence between Non-negative Matrix Factorization and Probabilistic Latent Semantic Indexing", *Computational Statistics & Data Analysis*, 52(8), 2008, pp. 3913–3927, URL *http* : *//users.cis.fiu.edu/ taoli/pub/NMFpLSIequiv.pdf*.

[13] Gintare Karolina Dziugaite and Daniel M. Roy, "Neural Network Matrix Factorization", , 2015, URL *https* : *//arxiv.org/abs/*1511.06443, arXiv abs/1511.06443.

[14] Joonseok Lee, et al., "LLORMA: Local Low-Rank Matrix Approximation", *Journal of Machine Learning Research*, 16, 2016, pp. 1–24.

[15] "Kernel smoother", *https* : *//en.wikipedia.org/wiki/Kernel*$_s$*moother*.

[16] "Local regression", *https* : *//en.wikipedia.org/wiki/Local*$_r$*egression*.

[17] Inderjit S. Dhillon, et al., "Information-Theoretic Co-clustering", *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, 2003, pp. 89–98, URL *http* : *//www.cs.utexas.edu/users/inderjit/public*$_p$*apers/kdd*$_c$*ocluster.pdf*.

[18] "Predicate", *https* : *//en.wikipedia.org/wiki/Predicate*$_($*grammar*).

[19] "Argument", *https* : *//en.wikipedia.org/wiki/Arguments(linguistics*).

[20] "Selection", *https* : *//en.wikipedia.org/wiki/Selection(linguistics*).

[21] Linas Vepstas, "Stiching Together Vector Spaces", , 2018, URL *https* : *//github.com/opencog/opencog/raw/opencog/nlp/learn/learn − lang − diary/stitching.pdf*.

[22] Greg Ver Steeg, et al., "Demystifying Information-Theoretic Clustering", , 2014, URL *https* : *//arxiv.org/abs/*1310.04210, arXiv abs/1310.04210.

[23] Erhan Gokcay and Jose C. Principe, "Information Theoretic Clustering", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2), 2002, pp. 158–172, URL *https* : *//pdfs.semanticscholar.org/f*706/5*b*6*edb*59*fb*19*d*987*c*5*d*791*e*73514528*ce*0*ca.pdf*.

[24] Linas Vepstas, "Connector Set Distributions", , 2017, URL *https* : *//github.com/opencog/opencog/raw/opencog/nlp/learn/learn − lang − diary/connector − sets − revised.pdf*.