Gradient Descent vs. Graphical Models

Linas Vepstas

6 July 2018; working draft of 24 October 2018

Abstract

This text provides a broad sketch of how deep-learning/neural-net approaches are quite similar to symbolic approaches to machine learning, knowledge representation and AI. Taken from the right viewpoint, they can be seen to be two variants of the same structure.

To keep the development focused and concrete, the presentation is limited models of natural language, and thus compares Word2Vec, Skip-Gram or AdaGram-style vector-space approaches to traditional symbolic linguistics approaches. To maintain concreteness, Link Grammar is used as a stand-in for a prototypical dependency grammar. Superficially, these systems appear to have nothing in common. On closer examination, it becomes evident that both employ a vector representation of words-in-context. The context is an N-gram, skip-gram or adagram, in the neural-net case, and a dependency linkage disjunct in the symbolic case.

The similarity becomes most apparent when the word+context is viewed 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 low-rank matrix factorization algorithms, information-theoretic clustering, and, most importantly co-clustering.

Although vector spaces are linear, semantics isn't; not really. Vector spaces (representing the different grammatical contexts associated with each word) can be stitched together into a unified whole, using concepts from sheaf theory. This completes the exposition here.

1 Introduction

Deep learning and neural nets are all the rage, today, and have displaced symbolic AI systems in most applications. It's commonly believed that the two approaches have nothing to do with each other; that they're just completely different, and that's that. But this is false: there are some profound similarities; they are not only variants of one-another, but, in a deep way, they have commonalities that render them essentially identical. This text attempts to explain how.

The clearest starting point for seeing this seems to be natural language, where neural net methods have made great strides, but have not surpassed traditional (symbolic) linguistic theory. However, once this similarity is understood, it can be ported over to other domains, including deep-learning strongholds such as vision. To keep the discussion anchored, and to avoid confusing abstractions, most of what follows will focus on linguistics; it is up to you, the reader, to imagine other, more general settings.

The starting point for probabilistic approaches (including deep learning) is the Bayesian network[1]: a probability $P(x_1, x_2, \ldots, x_n)$ of observing n events. For language, the x_k are taken to be words, and n is the length of the sentence, so that $P(x_1, x_2, \ldots, x_n)$ is the "probability" of observing the sequence $x_1, x_2, \ldots, x_n$ of words. The technical problem with this viewpoint is the explosively large space: if one limits oneself to a vocabulary of 10 thousand words (and many people don't) and sentences of 20 words or less, that's $(10^4)^{20} = 10^{80} = 2^{270}$ probabilities, an absurdly large number even for Jupiter-sized computers. If n is the length of this text, then it really seems impossible. The key, of course, is to realize that almost all of these probabilities are effectively zero; the goal of machine learning is to find a format, a representation for grammar (and meaning) that effortlessly avoids that vast ocean of zero entries.

Traditional linguistics has done exactly that: when one has a theory of syntax, one has a formulation that clearly states which sentences should be considered to be grammatically valid, and which ones not. The trick is to provide a lexis (a lookup table), and some fairly small number of rules that define how words can be combined; i.e. arranged to the left and right of one-another. You look up a word in the lexis (the dictionary) to find a listing of what other words are allowed to surround it. Try every possible combination of rules until you find one that works, where all of the words can hook up to one-another. For the purposes here, the easiest and the best way to visualize this is with Link Grammar[2, 3, 4], a specific kind of dependency grammar. All theories of grammar will constrain allowed syntax; but Link Grammar is useful for this comparison because it explicitly identifies words to the left, and words to the right. Each lexical entry is like a template, a fill-in-the-blanks form, a jigsaw-puzzle piece, telling you exactly what other words are allowed to the left, and to the right, of the given word. This left-right sequence makes it directly comparable to what neural-net approaches, such as Word2Vec[5, 6] or SkipGram[7, 8] do.

What does Word2Vec do? Clearly, the 2^{270} probabilities in a twenty-word sentence is overwhelming; one obvious simplification is to look only at N-grams: that is, to only look at the closest neighboring words, working in a window that is N words wide. For $N = 5$, this gives $(10^4)^5 = 10^{20} = 2^{65}$ which is still huge, but is bearable. When scanning actual text, almost all of these combinations won't be observed; this is just an upper bound. In practice, a table of 5-grams fit in latter-day computer RAM. The statistical model is to map each N-gram to a vector, use that vector to define a Boltzmann distribution ($P = \exp(\vec{v} \cdot \vec{w})/Z$), and then use gradient ascent (hill-climbing) to adjust the vector coefficients, so as to maximize find a maximum of the probability P.

How are Word2Vec and Link Grammar similar? The above description of Link Grammar should have already planted the idea that each lexical entry is a lot like an N-gram. Each lexical entry tells you which words can appear to the right, and which words to the left of a given word. Its a bit less constrained than an N-gram: there's no particular distance limitation on the dependency links. It can also skip over words: a lexical entry is more like a skip-gram. Although there is no distance limitation, lexical entries still have a small-N-like behavior, not in window size, but in attachment complexity. Determiners have a valency[9] of 1; nouns a valency of 2 or 3 (a link to a verb, to an adjective, to a determiner); verbs a valency of 2, 3 or 4 (subject, object, etc.). So lexical entries are like skip-grams: the window size is effectively unbounded, but the size of the context remains small (N is small).

Are there other similarities? Yes, but first, a detour. What happened to the 2^{270} probabilities? A symbolic theory of grammar, such as Link Grammar, is saying that nearly all of these are zero; the only ones that are not zero are the ones that obey the rules of the grammar. Consider the verb "throw" ("Kevin threw the ball"). A symbolic theory of grammar effectively sates that the only non-vanishing probabilities are those of the form

$$P(x_1, x_2, \cdots, x_k = \text{noun}, \cdots, x_m = \text{throw}, \cdots, x_p = \text{object}, \cdots, x_n)$$

and that all the others must be zero. Now, since nouns make up maybe $1/2$ of all words (the subject and object are both nouns), this constraint eliminates $10^4 \times 10^4/(2 \times 2) = 2^{24}$ possibilities (shrinking 2^{270} to $2^{270-24} = 2^{246}$). Nothing to sneeze at, given that its just one fairly simple rule. But this is only one rule: there are others, which say things like "singular count nouns must be preceded by a determiner" (so, "the ball"). These constraints are multiplicative: if the determiner is missing, then the probability is exactly zero. There are only a handful of determiners, so another factor of $10^4 = 2^{13}$ is vaporized. And so on. A relatively small lexis quickly collapses the set of possibilities. Can we make a back of the envelope estimate? A noun-constraint eliminates half the words (leaving 5K of 10K possibilities). A determiner constraint removes all but 10 possibilities. Many grammatical classes have only a few hundred members in them ("throw" is like "hit", but is not like "smile"). So, realistically, each x_k can have only about 100 possibilities; there are only about $100^{20} = 10^{40} = 2^{130}$ grammatically valid sentences that are 20 words long, and these can be encoded fairly accurately with a few thousand lexical entries.

In essence, a symbolic theory of grammar, and more specifically, dependency grammars, accomplish the holy grail of Bayesian networks: factorizing the Bayesian network. The lexical rules state that there is a node in the network, for example,

$$P(x_1, x_2, \cdots, x_k = \text{noun}, \cdots, x_m = \text{throw}, \cdots, x_p = \text{object}, \cdots, x_n)$$

and that the entire sentences is a product of such nodes: The probability of

"Kevin threw the ball" is the product

$$P(x_1 = \text{Kevin}, x_2 = \text{verb}, \cdots, x_n)$$
$$P(x_1, x_2, \cdots, x_k = \text{noun}, \cdots, x_m = \text{throw}, \cdots, x_p = \text{object}, \cdots, x_n)$$
$$P(x_1, x_2, \cdots, x_i = \text{the}, \cdots, x_p = \text{noun}, \cdots, x_n)$$
$$P(x_1, x_2, \cdots, x_n = \text{ball})$$

Stitch them all together, you've got a sentence, and its probability. (In Link Grammar, $-\log P$ is called the "cost", and costs are additive, for parse ordering.) To be explicit: lexical entries are exactly the same thing as the factors of a factorized Bayesian network. What's more, figuring out which of these factors come to play in analyzing a specific sentence is called "parsing". One picks through the lookup table of possible network factors, and wires them up, so that there are no dangling endpoints. Lexical entries are look like subsets of a graph: a vertex, and some dangling edges hanging from the vertex. Pick out the right vertexes (one per word), wire them together so that there are no dangling unconnected edges, and viola! One has a graph: the graph is the Bayesian network. Linguists use a different name: they call it a dependency parse.

The Word2Vec/SkipGram model also factorizes, in much the same way! First, note that the above parse can be written as a product of factors of the form $P(\text{word}|\text{context})$, the product running over all of the words in the sentence. For a dependency grammar, the context expresses the dependency relations. The Word2Vec factorization is identical; the context is simpler. In it's most naive form, its just a bag of the N nearest words, ignoring the word-order. But the word-order is ignored for practical reasons, not theoretical ones: it reduces the size of the computational problem; it speeds convergence. Smaller values of N mean that long-distance dependencies are hard to discover; the skipgram model partly overcomes this by keeping the bag small, while expanding the size of the window. If one uses the SkipGram model, with a large window size, and also keep track of the word-order, and restrict to low valencies, then one very nearly has a dependency grammar, in the style of Link Grammar. The only difference is that such a model does not force any explicit dependency constraints; rather, they are implicit, as the words must appear in the context. Compared to a normal dependency grammar, this might allow some words to be accidentally double-linked, when they should not have been. Dependency grammar constraints are sharper than merely asking for the correct context. To summarize: the notions of context are different, but there's a clear path from one to the other, with several interesting midway points.

The next obvious difference between Link Grammar and Word2Vec/SkipGram are the mechanisms for obtaining the probabilities. But this is naive: in fact, they are much more similar than it first appears. In both systems, the starting point is (conceptually) a matrix, of dimension $W \times K$, with W the size of the vocabulary, and K the size of the context. In general, K is much much larger than W; for Word2Vec, K could get as large as W^N for a window size of N, although, in practice, only a tiny fraction of that is observed. Both Link Grammar and Word2Vec perform approximate matrix factorization on this matrix.

The way the approximations are done is different. In the case of Word2Vec, one picks some much smaller dimension M, typically around $M = 200$, or maybe twice as large; this is the number of "neurons" in the middle layer. Then, all of W is projected down to this M-dimensional space (with a linear projection matrix). Separately, the K-dimensional context is also projected down. Given a word, let its projection be the (M-dimensional) vector $\vec{u}$. Given a context, let it's projection be the vector $\vec{v}$. The probability of a word-in-context is given by a Boltzmann distribution, as $\exp(\vec{u} \cdot \vec{v})/Z$ where $\vec{u} \cdot \vec{v}$ is the dot product and Z is a scale factor (called the "partition function"[10]). The basis elements in this M-dimensional space have no specific meaning; the grand-total vector space is rotationally invariant (only the dot product matters, and dot products are scalars).

The primary task for Word2Vec/SkipGram is to discover the two projection matrices. This can be done by gradient ascent (hill-climbing), looking to maximize the probability. The primary output of Word2Vec are the two projection matrices: one that is $W \times M$-dimensional, the other that is $M \times K$-dimensional. In general, neither of these matrices are sparse (that is, most entries are non-zero).

Link Grammar also performs a dimensional reduction, but not quite exactly by using projection matrices. Rather, a word can be assigned to several different word-categories (there are *de facto* about 2300 of these in the hand-built English dictionaries). Associated with each category is a list of dozens to thousands of "disjuncts" (dependency-grammar dependencies), which play the role analogous to "context". However, there are far, far fewer disjuncts than there are contexts. This is because every (multi-word) context is associated with a handful of disjuncts, in such a way that each disjunct stands for hundreds to as many as millions of different contexts. Effectively, the lexis of Link Grammar is a sparse $C \times D$-dimensional matrix, with C grammatical categories, and D disjuncts, and most entries in this $C \times D$ dimensional matrix being zero. (The upper bound on D is L^V, where L is the number of link types, and V is the maximum valency – about 5 or 6. In practice, D is in the tens of thousands.) The act of parsing selects a single entry from this matrix for each word in the sentence. The probability associated to that word is $\exp(-c)$ where c is the "cost", the numerical value stored at this matrix entry.

Thus, both systems perform a rather sharp dimensional reduction, to obtain a much-lower dimensional intermediate form. Word2Vec is explicitly linear, Link Grammar is not exactly. However (and this is important, but very abstract) Link Grammar can be described by a (non-symmetric) monoidal category. This category is similar to that of the so-called "pregroup grammar", and is described in a number of places[11] (some predating both Link Grammar an pregroup grammar). The curious thing is that linear algebra is also described by a monoidal category. One might say that this "explains" why Word2Vec works well: it is using the same underlying structural framework (monoidal categories) as traditional symbolic linguistics. The precise details are too complex to sketch here, and must remain cryptic, for now, although they are open to those versed in category theory. The curious reader is encouraged to ex-

plore category-theoretic approaches to grammar, safe in the understanding that they provide a foundational understanding, no matter which detailed theory one works in. At the same time, the category-theoretic approach suggests how Word2Vec (or any other neural-net or vector-based approach to grammar) can be improved upon: it shows how syntax can be restored, with the result still looking like a funny/unusual kind of sparse neural-net. These are not conflicting approaches; they have far far more in common than meets the eye. A draft discussion is presented in[12].

A few words about word-senses and semantics are in order. It has been generally observed that Word2Vec seems to encode "semantics" in some opaque way, in that it can distinguish different word-senses, based on context. The same is true for Link Grammar: when a word is used in a specific context, the result of parsing selects a single disjunct. That disjunct can be thought of as a hyper-fine grammatical category; but these are strongly correlated with meaning. Synonyms can be discovered in similar ways in both systems: if two different words all share a lot of the same disjuncts, they are effectively synonymous, and can be used interchangeably in sentences.

Similarly, given two different words in Word2Vec/SkipGram, if they both project down to approximately the same vector in the intermediate layer, they can be considered to be synonymous. This illustrates yet another way that Link Grammar and Word2Vec/SkipGram are similar: the list of all possible disjuncts associated with a word is also a vector, and, if two words have almost co-linear disjunct vectors, they are effectively synonymous. That is, disjunct-vectors behave almost exactly like neuron intermediate-layer vectors. They encode similar kinds of information into a vector format.

This is also where we have the largest, and the most important difference between neural net approaches, and Link Grammar. In the neural net approach, the intermediate neuron layer is a black box, completely opaque and unanalyzable, just some meaningless collection of floating-point numbers. In Link Grammar, the disjunct vector is clear, overt, and understandable: you can see exactly what it is encoding, because each disjunct tells you exactly the syntactic relationship between a word, and its neighbors. This is the great power of symbolic approaches to natural-language: they are human-auditable, human understandable in a way that neural nets are not. (Currently; I think that what this essay describes is an effective sketch for a technique for prying open the lid of the black box of neural nets. But that's for a different day.)

The remainder of this essay is structured as follows: Section 2 provides a very quick sketch of Link Grammar, emphasizing how dependency grammars have vector-like aspects to them. Section 3 reviews the based CBOW, SkipGram and neural net models, setting up some basic notation. Section 4 defines a graph network model of language, emphasizing that statistical physics provides a well-developed toolset for working with graph networks. Section 5 reviews several different techniques by which statistical information can be gathered, and the graphical structure can be infered, given a large "flat" corpus of text. Section 6...

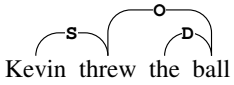
Many thanks and appreciation to Hanson Robotics for providing the time

to think and write about such things.

2 Link Grammar as a Model of Language

One of the barriers to understanding the commonality between symbolic and vector approaches is the notational difference. This section provides a highly abbreviated review of the Link Grammar[3, 4] notation, followed by a simple modification so that it's vector-like form becomes more apparent.

The figure below represents a (simplified) parse of the sentence "Kevin threw the ball":



The labeling is prototypical of a dependency grammar: the arc labeled "S" denotes a subject-verb dependency; "O" a verb-object dependency, and "D" links a noun to a determiner. The required lexical entries, in alphabetical order, are

```
ball:   D- & O-;
Kevin:  S+;
the:    D+;
threw:  S- & O+;
```

The capital letters, together with a +/- sign, are called "connectors"; a pair of connectors form a "link". The +/- sign indicates a left-right directionality: for example, the `S+` connector can only connect to the right; it must mate with an `S-` connector to form an `S` link. A valid parse exists if and only if all available connectors are paired up. The construction `S+ & O-` is called a "disjunct"; the name has a historical basis that is of no particular concern here. During parsing, all connectors in a disjunct must be satisfied.[1]

Each lexical entry is a word-disjunct pair; they are of the general form

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

Because every link connects a pair of words, it can be thought of as the set of all possible word-pairs allowed by that connector. Thus, in the example above, the link `O` is effectively equivalent to the singleton set `{threw-ball}`. The above dictionary is so simple, that there are no other word-pairs in this set; in general, this is not the case. Thus, the capital-letter link types are actual types, in the type-theoretic sense.

At this point, it is notationally convenient to replace each type by the set, like so:

[1] Not immediately obvious from this abbreviated sketch is that the Link Grammar formulation of a dependency grammar is sufficiently strong to encode long-range coordination or enforce the correct usage of set phrases/phrasemes with "holes" in them. It can. There is a set of effects that can be encoded in this way, from phonetic structure to to morphology. The referenced sources open the doors to understanding how this can be done.

```
ball:   {the-ball}- & {threw-ball}-;
Kevin:  {Kevin-threw}+;
the:    {the-ball}+;
threw:  {Kevin-threw}- & {threw-ball}+;
```

Each lexical entry above redundantly repeats one of the words in each word-pair. To obtain a more compact notation, drop the redundant word, to get

```
ball:   the- & threw-;
Kevin:  threw+;
the:    ball+;
threw:  Kevin- & ball+;
```

This is encoding represents the exact same dictionary as the first one, but now using the attachment-words as connectors, instead of single letters. The resulting sentence parse is exactly the same as before. The link types appear to be lost; these can be partly restored, as long as some other location records that `{threw-ball}` should be replaced by the link `O`.

The above representation for the dictionary is verbose, and is not very practical for hand-crafted dictionaries. Link types really are much more convenient. However, the above representation helps make it brutally clear that disjuncts effectively resemble adaptive N-gram word-contexts. The pseudo-disjunct, where the connector types are replaced by instances of individual words, can be used as a word-in-context. For example, a quick glance at the pseudo-disjunct

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

makes it clear that it might be observed in a sentence such as "the girl ran home". Neither the subject nor the object are explicitly labeled; however, one can maintain statistical observation counts using this method. It provides a direct bridge from dependency grammars to corpus linguistics. The vectorial representation arises naturally when accumulating observation counts of different disjuncts. A word-vector is then simply a count of all the different observed pseudo-disjuncts. For example, one might observe the vector $\vec{v}_{\text{ran}}$, which may be represented as

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

This will naturally arise if the sentence "the girl ran home" was observed three times, and "the girl ran away" was observed twice.

From this point on, the standard panoply of vector-based techniques become available. Normalizing vectors to be of unit-length implies that the frequency-counts can be re-interpreted as frequency-probabilities. Once in possession of vectors, the standard tricks of vectorially-based semantic similarity apply. For example, a different vector $\vec{v}_{\text{walked}}$ represented as

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

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

similar ways. Syntactic similarity is generally associated with semantic similarity (synonymy). Later in this text, it is strongly argued that cosine-products are *not* the correct way to measure similarity (synonymy), and that the information-theoretic Kullback-Leibler divergence is much more appropriate. A precise formulation will be given then. For now, it is enough to note that the availability of a vector means that all usual vector tricks can be applied.

However, unlike (adaptive) N-gram techniques, the disjunct approach also enables a very different kind of vector to be defined. Given the above corpus of five sentences, one can formulate a different vector, for the word "home", as follows. Unlike the earlier examples, the representation is a bit more awkward:

```
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. The attachment-point or "germ" can be high-lighted by replacing it with a star:

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

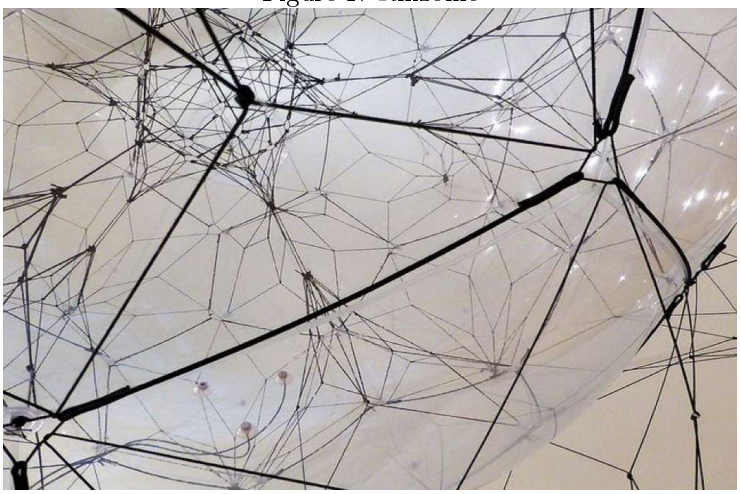
This is a fundamentally different kind of vector than the earlier examples. The ordinary (adaptive) N-gram vectors do not have this kind of representational ability; they all collapse down to the same representation.

This difference turns out to be important, and has deep repercussions. A quick sketch can be given. These two different vector types can be distinguished by employing superscripts D and C: *viz.* write $\vec{v}^D_{\text{ran}}$ for the disjunct-based vector, and $\vec{v}^C_{\text{home}}$ for the connector-based vector. Given any word w, there will in general always be vectors $\vec{v}^D_w$ and also $\vec{v}^C_w$. Even more: there will be several forms of $\vec{v}^C_w$, with the word occupying different slots in the disjunct, thus $\vec{v}^{C_1}_w$ and $\vec{v}^{C_2}_w$ and $\vec{v}^{C_3}_w$ and so on. So, since the germ was located in the second slot, one should write $\vec{v}^{C_2}_{\text{home}}$ instead of $\vec{v}^C_{\text{home}}$.

These vectors can be taken together as $\vec{v}^D_w \oplus \vec{v}^{C_1}_w \oplus \vec{v}^{C_2}_w \oplus \cdots$ which inhabit orthogonal subspaces of $\vec{V}^D \oplus \vec{V}^{C_1} \oplus \vec{V}^{C_2} \oplus \cdots$. The dependency-grammar constraints embodied in the connectors imply that these vector spaces can be "glued together"; the dependency-grammar constraints imply that these vector subspaces can be glued together or stitched together in a highly non-linear fashion. The rules for gluing are in fact identical to the gluing axioms of sheaf theory and algebraic topology. In essence, the algebra of natural language is the algebra of a sheaf. The disjunct vectors can be understood as a kind-of germ or stalk, similar to the "tangent vector" of a kind-of "manifold" of natural language. Unfortunately, algebraic geometry is a somewhat abstract branch of mathematics, and is normally quite distant from linguistics. This text is not appropriate for further exploration; see [12] for details.

To simplify these last claims somewhat, they effectively say this: the correct conceptual model for the observational data is that of a large network graph, with vectors of observation counts attached to each vertex. When the vectors are consistently glued to one-another, so that only the syntactically-allowed sentences are possible, the network begins to look more like a bramble or rhizome,

Figure 1: Rhizome



This image illustrates the general concept of a rhizome. It is meant only to provide a suggestive inspiration for visualizing a graphical sheaf. Lines correspond to dependency grammar link contraints between tangent vector spaces. Since any given word in a sentence can have dependency relationships to a number of other words, each vertex here (a word) has multiple edges attached to it. Any grammatically valid dependency parse is then a subtree of this image. This image is at best only suggestive; a true sheaf would have stalks, missing in this illustration, with stalks corresponding to the vectors (sections) above a point in the base space (points in the base space being words). (Photograph of a sculpture at the Copenhagen Art Musem; photo credit Jenny Mackness (2014). License: Attribution-NonCommercial-ShareAlike 2.0 Generic (CC BY-NC-SA 2.0) (Flickr))

---

with each thread in the rhizome corresponding to a grammatically-correct sentence. The algebraic structure of this rhizome is a sheaf, explicitly in the sense of sheaf theory. Figure 1 illustrates a rhizome.

3 Statistical Network 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:[6]

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

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 mentally constructed nearly whole before being written or spoken, 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[7], 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

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

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 again, it is, of course, "obviously" wrong. At the sentence level, in English, we commonly do not start sentences with verbs. At the text level, 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, so that furniture-words become uncommon, while nature-words occur more frequently. The translational invariance only becomes plausible in the limit of $N \to \infty$ 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.

In general, it seems reasonable to assume that the distribution of words, when taken over sliding windows of different sizes, varies in a scale-free fashion with the window size. So, consider a long book, and a sliding window of width $N = 1000$. The distribution of the words within that window will be Zipfian; as the window slides from scene to scene, the probability distribution will be sensitive to that scene (indoors/outdoors, with/without some character...) The positional non-uniformity persists at all window-sizes N.

In what follows, we are primarily concerned with the grammatical structure of single sentences, and not the narrative structure of longer texts. Effectively, it is enough to work with a value of N that is less than a few dozen. The fact that sentences typically do not start with verbs can be "explained" by the rules of syntax; no explicit factorization for single-word probabilities is required. Thus, the translation-invariant "Bayesian network" factorization of eqn 1 is appropriate and suitable for the current task. As will be seen in later chapters, this factorization can be improved on, and performed much more cleanly and elegantly.

3.1 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 up to $W^N = 10^{12} = 2^{40}$ probabilities to be maintained: even if most 3-word sequences are never observed, this is still clearly near the edge of what is possible with present-day computers.

The model can be made computationally tractable in various ways. One well-discussed variant is to blend together, in varying proportions, the models

for $N = 0$, $N = 1$ and $N = 2$. Such an approach is of no particular interest for the subsequent development; grammar happens at larger values of N, and the goal is to create effective factorizations that encompass grammar.

3.2 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. If one can find a model that captures that all but one in ten-thousand such probabilities are zero, then tht model becomes compuitationally tractable again. An alternate model is to hypothesize that certain linear combinations of the probabilities are all equal to one-another. The latter approach is exactly that of the neural net and deep learning approaches: the linear combinations of equivalent probabilities are the ones represented by various layers in the neural net.

There is a fairly rich variety of such models. Reviewed immediately below are two 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 of training algorithms will be made after the initial review of the CBOW and SkipGram models.

3.3 CBOW

Mikolov, *etal*[8, 7] 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

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

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

$$p\left(w_t \,|w_I\right) = \frac{\exp \sum_{i \in I} \vec{u}_t \cdot \vec{v}_i}{\sum_{j \in W} \exp \sum_{i \in I} \vec{u}_j \cdot \vec{v}_i} \tag{2}$$

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 $1 = \sum_j p\left(w_j \,|w_I\right)$.

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" $E = -\log p(w_t | w_I)$. 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[13].

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 $\pi = \pi'$. Knowledgable readers are encouraged to correct my misperception.

3.4 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

$$p(w_I | w_t) = \frac{\exp \sum_{i \in I} \vec{u}_t \cdot \vec{v}_i}{\sum_{I \in W^N} \exp \sum_{i \in I} \vec{u}_t \cdot \vec{v}_i} \tag{3}$$

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.

3.5 Perceptrons and Neural Nets

The neural net model proposed by Bengio[6] 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 $\sigma(x) = \tanh x$ or $\sigma(x) = 1/(1 + e^{-x})$ 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

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

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, dimensional reduction.

4 Graph Network Models of Language

Superficially, the graphical approach of a symbolic dependency grammar, such as Link Grammar, seems to have no resemblance at all to the probabilistic approach of neural network models. The previous sections hinted at the presence of vectors in both systems; but the vectors are superficially different, and the role of probability in either remained less than entirely clear. The differences are less than they seem; different notation obscures similarity; different traditions emphasize different aspects so strongly that even minor hints of similarity are lost. This section attempts to articulate the commonality.

Both the neural network and the symbolic graphical approaches to language could be termed to two different faces of a common graphical network model of language. The structure of graph networks are a part of the standard curriculum of physics and statistical mechanics. This section begins with a reiteration that Link Grammar disjuncts resemble the factors of a factored Bayesian network. Next, the loss functions of the CBOW/SkipGram model can be seen

as a manifestation of standard (statistical mechanics) maximum entropy principles. Equiped with this insight, one can proceed along a common track in physics: derive the statistical models from a partition function. The partition function encodes "all possible knowledge" of a network graph; specific factors of this graph are called "Feynman diagrams". With the tools of physics thus deployed, a dependency parse, when obtained statistically, can be recognized as a kind of Feynman diagram. The dependency grammars of natural language are Feynman diagrams of statistical physics.

This section sets the stage for the next, which reviews several different approaches for obtaining the statistical information needed to correctly factor a graph network.

4.1 Disjuncts as Context

Consider the probability

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

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.[2] This can be remedied in a straightforward manner: write d for the disjunct, girl- & home+ in this example. One can then define the probability

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

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

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

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 understand that

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

[2]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 $-\log p\left(w|d\right)$. The hand-crafted dictionaries provide hand-crafted estimates for this cost/log-likelihood.

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. More precisely, the disjunct notation actively skips over some of the words in the context. Rather than specifying a fixed offset from the target word, the disjunct provides a relative squential ordering of the context-words, left to right, without specifying distance.

This change of notation allows the disjunct to do more than the index set notation: the disjunct effectively encodes syntactic information. How this is done was already detailed in section 2. With this seemingly minor change of notation, from indexes to sequential lists, one gains access to grammatical information. Do keep in mind that by working with disjunct notation, nothing is lost: if one wished, 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. The disjunct encodes "more information", in the information-theoretic sense of encoding "more bits of info" than the index notatin can.

4.2 Skip-Grams and Syntatic Structure

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.[3]

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.

[3] These statements are also perhaps misleading: conventional systems really do use a "bag of words", ignoring the word-order. Yet word-order is needed to write a disjunct. Thus, existing off-the-shelf software would have to be modified to track word-order, and this modification will require an order of magnitude more storage, possibly rendering the computation intractably large or slow. Keeping track of word order is theoretically interesting, but entirely misses the tractability issue that these models were invented to solve.

Such long-distance attachments are problematic for CBOW or Skip-Grams, in several ways. One is that the window N would have to 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[7] 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 in a naive correspondance with index notation 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.[4]

4.3 Statistical Mechanics and the Partition Function

The CBOW and SkipGram models of language, namely equations 2 and Skip-Gram 3, are commonly discussed as optimization problems, and the utility functions are sometimes called "softmax" functions, or "loss functions". From the point of view of statistical mechanics, these utility functions can be easily

[4] I assume that Parsey McParseFace overcomes all of these problems ins some way; I have not studied it.

recognized as Gibbs distributions. This allows all of the traditional intuition and tools of statistical mechanics to be brought to bear on the language problem. Specifically, the probabilities are obtainable from a partition function.

Given that equations 2 and 3 are conditional probabilities, one can deduce that the joint probability in these models is given by

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

This can be obtained by applying variational principles to the partition function

$$Z[J] = \sum_{w \in W} \sum_{I \in W^N} \exp \left(\sum_{i \in I} \vec{u}_w \cdot \vec{v}_i + J_{wI} \right) \tag{4}$$

In physics literature, the $J_{w_t,I}$ are called "sources" or the "current"[5], and can be understood as parameters that are nominally zero. That is, they are set to zero "in the real world", but serve as placeholders within the partition function so that variational principles can be applied. Doing so 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 framework of maximum entropy principles. This is no accident, of course; the "softmax" function was used precisely because it gives the maximum entropy distribution.

The last statement can be made even more precise. A language model is a probability distribution $p(w_1, w_2, \cdots)$ defined over a sequence of words $w_1, w_2, \cdots$. The set of all such sequences is termed (in the historical literature of physics and thermodynamics) an "ensemble". The entropy of a particular language model is a sum over the ensemble

$$S[p] = -\sum_{w_1, w_2, \cdots} p(w_1, w_2, \cdots) \log_2 p(w_1, w_2, \cdots)$$

The principle of maximum entropy states that the above should be solved to find the probability distribution p that maximizes the entropy $S[p]$. This can be solved without the need to provide any addtional statements or constraints. Standard texts on statistical mechanics exhibit the solution; it is the Boltzmann distribution, namely

$$p(w_1, w_2, \cdots) = \frac{1}{Z} \exp -E(w_1, w_2, \cdots) \tag{5}$$

with the partition function Z being given by

$$Z = \sum_{w_1, w_2, \cdots} \exp -E(w_1, w_2, \cdots)$$

[5] Mathematically, they resemble electric charges or currents.

To make ends meet with eqn 4, one writes $Z = Z[J = 0]$ so that if the current J is zero, one simply does not write it, as it is just a device handy for algebraic manipulations, but having no particularly deep significance for the language model.

Thus, a maximum entropy model of natural language is any model that provides an energy function $E(w_1, w_2, \cdots)$ for a sequence of words $w_1, w_2, \cdots$. Clearly, CBOW/SkipGram is such a model. There are others.

4.4 Ising Models of Grammar

The vector product $\sum_{i \in I} \vec{u}_w \cdot \vec{v}_i$ appearing in the partition function 4, and indirectly in the CBOW and SkipGram models 2 and 3 is pretty much the grand-total extent or content of these language models. The statement is effectively that vectors provide a pretty good model of natural language, and the vectors are able to capture important features of natural language, including semantics and compositionality (such as the "King" - "man" + "woman" = "Queen" example[8, 7]). This is an interesting effect, because the vectors themselves seem to be some kind of effectively structureless black boxes; they capture some kind of latent structure of language, but how this is captured is entirely opaque. This is entirely at odds with traditional theories of syntax, as developed by more than half a century of linguistics research. How can the latent content of the feature vectors be reconciled with structural theories of syntax?

The notion of the Ising model can be used to bridge this gap. In the original formulation of the Ising model, the loss function (the Hamiltonian; the energy) was written as

$$E(\sigma) = \sum_i h_i \sigma_i + \sum_{i,j} J_{ij} \sigma_i \sigma_j$$

with the σ_i being the value of a "spin" at a "lattice position" i, the h_i being the "magnetic field" (at lattice position i), and the J_{ij} being the interaction energy between neighbors (typically, the nearest neighbors, and typically dependent only on the distance $|i - j|$ between neighbors).[6] In the present context, the σ_i are to be reinterpreted as w_i, the word at position i in a sentence.

The Ising model is usually taken to be "translation invariant" or "shift invariant", dependent only on relative positions between points in the lattice, rather than their absolute position. For the one-dimensional Ising model, where the lattice points are in a linear sequence, the shift invariance implies a number of interesting connections to Markov chains, shift sequences, and finite state machines. This, in turn, implies that the Ising model, or a variant thereof, occurs in many natural models. One example is the distribution of sequences of amino acids in Zebrafish antibodies.[14] In that model, each σ_i can be one of twenty-one different amino acids, and one is interested in describing the distribution of a wide variety of different sequences that antibodies employ to fight off infection.

[6] The J_{ij} used here has no relationship whatsoever to the current J of the previous section; rather, there are not enough letters in the alphabet, and by convention, J is used for both tasks.

Another example is that of protein sequences, specifically, of sensor kinase and response regulator proteins in bacteria[15], where the goal is to identify which sensor amino-acid sequences directly trigger response proteins, as opposed to merely being correlated with a response.

The point to be made here is that the Ising model also provides a natural way to unify the syntactic structure of language with the syntax-free vector models. The reinterpretation is that lattice position i corresponds to the i'th word in a linear text. The spin σ_i is to be reinterpreted as the word instance w_i located there. The correct interpretation of the one-point function h_i (the "magnetic field") is explored in the next section; for now, it can be ignored (taken to be zero). The part of the model responsible for grammar are the two-point functions J_{ij}, which need to be generalized into n-point functions that capture the grammatical relationships between words. Given a sequence of lattice values $\sigma_i, \sigma_{i+1}, \cdots$ (that is, a sequence of words $w_i, w_{i+1}, \cdots$) one has an "interaction energy" $J_{w_i,w_{i+1},\cdots}$ that is small when the sequence of words $w_i, w_{i+1}, \cdots$ is likely, and is large (or infinite) when the sequence of words is grammatically incorrect. In effect,

$$J_{w_i,w_{i+1},\cdots} = -\log p\left(w_i, w_{i+1}, \cdots\right)$$

is the loss function. This is just eqn 5 in slightly different form.

The statement that Ising model provides a model of natural language might superficially appear to be content-free, an empty and trite statement (there's a "so what" aspect to it: "show me something I did not already know", as the Ising model just seems to be a standard maximum entropy model in faint disguise.) What makes it not entirely trivial is the claim that the interaction energies (loss functions) are additive. That is, given a sequence of words $w_i, w_{i+1}, \cdots$, the "grammatical validity" of that sequence can be described in terms of (statistically) independent, single real-valued numbers $J_{w_i,w_{i+1},\cdots}$ which are additive: they are to be summed (and not combined in some other, more complex way).

The additive model of grammatical structure also exposes the limit of this language model: we do not speak in a word-salad of grammatically valid sentences; rather, there is always a message conveyed in an utterance. The statistically independent numbers $J_{w_i,w_{i+1},\cdots}$ are sufficient to capture the grammar and syntactic structure of the language, but not (at this level) to capture the message itself. The n-point functions $J_{w_i,w_{i+1},\cdots}$ are to be taken as the coding of the language, in the sense of "coding theory" of signal processing; they convert plaintext to cryptext. The plaintext is a bag-of-concepts; the cryptext is the word-sequence that encodes the plaintext.

4.5 MST models of Language

The importance of the additive aspect of the Ising model of natural language is best illustrated by reverting to the simpler two-point model of natural language, by restricting the interaction J to occur only between pairs of words. This effectively gives the Maximum Spanning Tree (MST) model of language,

explored by Yuret[16] and by McDonald[17, 18]. Here, the relationship between words is presumed to be entirely pair-wise; after somehow obtaining pair-wise word statistics $p(w_{left}, w_{right})$, one constructs a maximum-spanning-tree parse so as to maximize the total entropy.

The use of a maximum spanning tree, such as the supervised training models of McDonald *etal.*, is a key insight. McDonald states this in terms of a score $s(x, y)$ that the sentence x is described by parse tree y. The score is taken to be additive over word pairs:

$$s(x, y) = \sum_{(i,j)\in y} s(i, j)$$

so that the sum ranges over all words-pairs (i, j) occurring in the parse-tree y (and the individual words i, j being the words of sentence x). How does one obtain the score $s(i, j)$? McDonald *etal.* propose a supervised training algorithm, where the score is induced by means of a gradient descent from an *a priori* training corpus of parse trees.

It is here that Yuret does one better: he proposes that the score be given by the mutual information (MI) between word-pairs; that is (aligning the different notation)

$$s(i, j) = MI(w_i, w_j) = \log_2 \frac{p(w_i, w_j)}{p(w_i, *)\, p(*, w_j)}$$

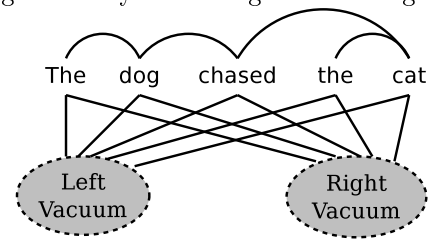
How does this square with the Ising model? Superficially, it seems to be quite different: the Ising model suggests that one should have only pair-wise interactions $J_{ij} = \log p(w_i, w_j)$ whereas the mutual information has an additional funny denominator. This can be reconciled by saying that there is a bulk interaction: each individual word-site i also interacts with the "bulk" of all possible other words that could ever occur in some other speakable, grammatically correct sentence.

Put differently, there are two bulk "magnetic fields", or one-point interactions $h(w_i, *) = -\log_2 p(w_i, *)$ and $h(*, w_j) = -\log_2 p(*, w_j)$ that enter into the Ising model Hamiltonian, so that, for sentence x and parse tree y, one has a "score" (following McDonald) or "energy" (following statistical physics) of

$$\begin{aligned} s(x, y) &= \sum_{i\in x} h_i + \sum_{(i,j)\in y} s(i, j) \\ &= \sum_{i\in x} [h(w_i, *) + h(*, w_i)] + \sum_{(i,j)\in y} \log_2 p(w_i, w_j) \end{aligned}$$

When Yuret chooses to look for the spanning tree that maximizes the sum of the mutual information between word-pairs, what he is actually doing is looking for the parse tree that maximizes the entropy within the background context (the "bulk magnetic field") of the language model. This is an absolutely key, fundamental insight of the Ising model of language: the relationship between words matters not so much because of the individual relationships, but because

Figure 2: Feynman Diagrams in Language



This figure illustrates the energy functional for a particular English langauge sentence, chosen out of the statistical ensemble of all possible English language sentences. The arcs above the sentence show the standard unlabeled dependency parse of the sentence. It is conventional to consider such a parse as standing alone, implicitly with some language model that is not diagrammatically represented. When drawn stand-alone, the arcs are presumed to have some unspecified cost, such that the desired parse tree is preferentially selected. Here, the arcs are intended to very explicitly correspond to "propagators" or "Feynman rules" with a value of $h = \log_2 p(w_i, w_j)$. The two vacuum bubbles below the sentence, and the straight lines connecting each word to each vacuum are used to explicitly represent the language model. The straight lines connecting to the left vacuum are explicitly given by the propagator $h = -\log_2 p(*, w)$ while those going to right vacuum are given by $h = -\log_2 p(w, *)$. The total action (Hamiltonian, in this case) is given by $H_{total} = \sum_{arcs} h$ with the sum being over all lines and arcs (with appropriate multiplicity). This summation makes explicit what is otherwise an implicit langauge model: namely, that

$$H_{total} = \sum_{arcs} h = \sum_{(w_i,w_j)\in tree} MI(w_i, w_j)$$

where the right-hand side is the sum over mutual information between word-pairs. That is, the total mutual information in a parse tree is exactly equivalent to the total entropy of that parse tree, taken within the context of the entire language model.

The use of the name of Feynman in this context is not meant to be some vague, hand-waving abuse of terminology, peacock feathering on the part of the author. The rules given here are *bona fide* Feynman rules, in the strict, straight and narrow sense. They are a set of rules used to give explicit numerical values to a hand-drawn diagram, making express that the diagram is the same thing as a specific equation. Here, by applying the rules, one can see that the diagram is an explict representation of the equation that states that H_{total} is the total energy, given as the sum over all arcs in the diagram.

---

these relationships have to be taken with regards to the background of the entire language, in full.

Figure 2 attempts to drive this point home, by explicitly showing the vacuum contribution (the "bulk magnetic field" contribution) of the language model. By convention, when linguists draw dependency-parse diagrams, they omit the bottom half of this figure. This is a reasonable omission: it not only clutters the diagram, but, from the point of view of linguistics, it almost seems like tautological nonsense.[7] Of course a language model is taken within the context of the language. However, in order to bridge from a purely linguistic model, to a statistical model based on probabilities taken from first principles from Boltzmann distributions derived from partition functions, those implicit terms must be made explicit. The single-point propagators have an explicit numerical effect in the determination of the best parse tree for a given sentence.

4.6 Vacuum Contributions and Exact Solutions

Pair-wise mutual information is not the same thing as the two-point function J_{ij}; the MI includes one-point terms in it. For many applications, including chemistry, the one-point vacuum contributions to the MI are clearly inappropriate. Atoms, amino acids and protiens don't respond to other atoms that "might have been there, but weren't". For those applications, MI can only provide a rough approximation for a starting point for a more acurate solution of the "true" Ising model, one which contains a two-point function, but no one-point functions. That is, MI is not a substitute for accurately solving the Ising model, when a solution of the Ising model is actually needed.

For example, for protein folding and protein mating, the solution to the Ising model obtained by message passing becomes a very accurate predictor of the three-dimensional physical structure of the protein, and indicates which amino acids actually become physically close to one-another. It is considerably more accurate than the MI-based estimate.[15] There are several different algorithms for solving the Ising model, including Monte Carlo and message passing algorithms.[19, 15]

4.7 n-point Functions and Word-Sense Disambiguation

The preceeding two sections potentially sow a point of confusion: Should one consider a "pure" Ising model of natural language, one which omits the one-point contributions? Would it be more accurate than an MST-parse? Maybe. And maybe not. The author is not aware of any pure-Ising models of natural language that could be compared to the results of MST parsing.

For natural language, most words have valancies of 3, 4 or 5, and so it seems that they would be better described by 3-point, 4-point or 5-point Ising

[7] The word "vacuum" comes from quantum field theory, where there was a similar realization that the tautologically-empty vacuum is not so empty after all, but has explicit contributions to the probabilities of various physical processes. The vacuum is an implicit context for physical processes, which on occasion must be made explicit in order to get correct predictions.

interaction energies, as opposed to a bag of 2-point interactions. How true is this? Consider a transitive verb: it has a subject and an object. If the subject and object are truly independent of one-another, then two pair-wise interactions would be enough: a noun-verb link for the subject, and a verb-noun link for the object. Are subjects and objects independent of one-another? Consider the verb "throw", and a sentence of the form "subject throws object". In this case, "subject" can only be chosen from the class of animate beings, capable of throwing things. The object must be chosen from the class of physical objects that can be thrown. Although these two classes are fairly narrow, they still seem uncorrelated. This illusion can be broken: further restricing the class of subjects to humans forces the restriction of objects to the class of physical items weighing between a few grams and ten kilograms. But there is only one word "throw", and not some hypothectical throw_1 and throw_2 and throw_3, where throw_2 is used only with human subjects. The verb subjects and objects are correlated; when this is the case, a pair of two-point functions is not enough; these require a three-point function for an accurate description.

Perhaps, after word-sense disambiguation, one can discover multiple word senses, so that, for example, throw_1 applies to metal machines that throw heavy objects, throw_2 pertains to human subjects, and throw_3 applies to horses. In this case, the three-point function might indeed factorize. Written as an algebraic expression, one might be able to observe the factorization

$$\begin{aligned} p\left(w_{subj}, \text{throw}, w_{obj}\right) = & p\left(w_{machine}, \text{throw}_1\right) p\left(\text{throw}_1, w_{heavy}\right) \\ & + p\left(w_{human}, \text{throw}_2\right) p\left(\text{throw}_2, w_{graspable}\right) \\ & + p\left(w_{horse}, \text{throw}_3\right) p\left(\text{throw}_3, w_{rider}\right) \end{aligned}$$

so that any kind of machine can throw_1 any kind of heavy object, in a statistically uncorrelated sense, whereas any human can throw_2 any object that can be grasped by hand, with there being no statistical correlation between the human subject and the human-thrown object. The third factorization is likewise: only horses or other mountable creatures can throw_3 thier riders, but there is no statistical correlation (no three-point function) between the thrower, and the rider.

This kind of factorization into components might even provide a plausible automated means of performing word-sense disambiguation, given only the undifferentiated, measured 3-point frequency $p\left(w_{subj}, \text{throw}, w_{obj}\right)$. Exactly how far this can be pushed, without grinding into the floor of statistically noisey data is unclear: only horses throw cowboys, and only elephants throw maharajis.

4.8 Disjunct Models of Language

Although Yuret's thesis clearly demonstrates that pair-wise word interactions give a reasonable model of natural language, it also demonstrates that the model is imperfect. Link Grammar[3, 4] provides a much better model of language, and it is anchored on top of half a century of tradition in linguistics research into structure. To place it within the current context, the key leap being taken

is to replace the pair-wise word-interactions $J_{ij} = -\log p(w_i, w_j)$ by disjuncts

$$J_{ijk\cdots} = -\log p(w_i; w_{left\,facing}; w_{right\,facing})$$

The notation above is awkward: the idea here is that some word w_i can connect to other words on the left, by means of left-pointing connectors (or links) and it can connect to words on the right, with right-pointing connectors. These are distinct sets, or sequences, since the word-order matters.

There are multiple ways of overcoming the notational awkwardness of the above; these are reviewed in a later section. In particular, the notation above differs sharply from that used in Link Grammar; this is a notational issue, and not a conceptual issue.

Before proceeding, the remark of "resting on a tradition of linguistics" should be clarified. Link Grammar proposes a dependency-grammar like model of language. Other popular models of language include constituency-tree grammars, such as Head-Phrase Structure Grammar (HPSG). Grammars such as Categorial Grammar (CG) combine aspects of constituency with aspects of dependency. There are others. In principle, each of these different grammar frameworks are convertible to one-another by fixed algorithms that terminate in finite time. That is, given a lexis in one grammar framework, there is a purely mechanical means of translating that lexis into the lexis required by a different theoretical framework. This perhaps over-simplifies the situation: different theories of grammar are often founded on subtle distinctions; for example Dick Hudson's Word Grammar[20, 21] replaces the no-links-crossing constraint of dependency grammars by a notion of landmark transitivity. We sweep such details under the rug, for now, wishing to rest on the broader statement that all grammar frameworks can be converted into one-another, while acknowledging that some frameworks give better insight into language than others.

The primary point here is that Link Grammar is interesting because it provides a bridge to both traditional, symbolic structural linguistics (per above) and while also allowing an interpretation as an Ising model of language. Following Yuret's ansatz, one includes energy terms of not only $J_{ijk\cdots}$ in the Hamiltonian, but also bulk terms, indicating how disjuncts interact in the background. In practice, what this means is that one should associate to each word-disjunct pair a mutual information

$$MI(w, d) = \log_2 \frac{p(w, d)}{p(w, *)\, p(*, d)}$$

with $p(w, d)$ being the joint probability of observing disjunct d on word w. The most likely parse of a sentence is one where the sum over mutual information is maximized.

The term $p(w, *)$ looks uneventful, but hides a more complex structure. One

has

$$\begin{aligned} p(w,*) &= \sum_d p(w,d) \\ &= \sum_c p(w;c) + \sum_{c_1,c_2} p(w;c_1,c_2) + \sum_{c_1,c_2,c_3} p(w;c_1,c_2,c_3) + \cdots \end{aligned}$$

where the sums over c_k are sums over all connectors appearing in some disjunct. Keep in mind that each connector can connect to the left or to the right: a connector is a word, plus a direction indicator. To bridge the notation back to that of the Ising model, one would write for the loss function

$$J_{ijk\ldots} = -\log_2 p(w,d) = -\log_2 p(w;(w_i,x_i),(w_j,x_j),(w_k,x_k),\cdots)$$

where $c_k = (w_k, x_k)$ is a connector with word w_k and direction x_k. Clearly, writing d for a disjunct is much simpler than writing out the mass of connectors; yet the notation can be deceiving because it hides the complexity of the model.

5 Graph Algorithms vs. Gradient Descent

The previous sections illustrate that the CBOW/SkipGram model is a form of a maximum-entropy model, and that the disjunct model of natural language is amenable to the same treatment. In one case, the word-context is an N-gram; in the other, it is a disjunct; but these are not so unlike one-another, they have much in common. Both langauge models share a common difficulty: finding an energy functional that correctly models the observed probability distribution.

There are several generic classes of algorithms that can be applied to obtain the energy functional. One class are termed Monte Carlo methods, driving either hill-climbing, gradient descent or "relaxation" algorithms; deep learning algorithms fall in this class. The other class of algorithms can be loosely called "graph algorithms". These accumulate frequency counts on low-arity observed relationships, and use the logarithm of these frequencies as a surrogate for the energy functional.

The class of Mone Carlo methods can be said to be "global", in that they are trying to navigate an extremely large energy functional landscape through random sampling. By contrast, the graph algorithms are manifestly local, and re-assemble the global landscape explicitly by means of Feynman diagrams and vacuum contributions, as illustrated in figure 2. The graphical algorithms expose the graph network structure immediately; in the above examples, as maximum spanning trees that can be found using greedy algorithms applied to pair-wise relationships. The Monte Carlo algorithms do not provide any manifest graphical structure, although perhaps one can be discerned by applying some sort of thresholding to eliminate weak links. Indeed, thresholding, using sigmoid functions to prune unwanted, weak weights from a neural net is a characteristic of most deep-learning models. Even so, the resulting weight vectors and weight matrices tend to be cryptic amorphous masses: large blocks of uninterpretable numbers whose precise role and importance seems impossible to discern.

5.1 Graph Algorithms vs. Monte Carlo for Word Pairs

The difference between graphical and Monte Carlo methods is perhaps best illustrated by example. Yuret assumes an Ising model of natural language, and considers only pair-wise word realtionships. To obtain an energy functional, it suffices to count the frequency $N(w_{left}, w_{right})$ of word-pairs, and then to take for granted that the energy functional is accurately estimated by

$$E(w_{left}, w_{right}) = -\log_2 \left[\frac{N(w_{left}, w_{right})}{N(*,*)} \right]$$

To obtain dependency parses that agree with those deemed acceptable by professional linguists, maximum spanning trees are constructed, taking into account the important role of the left and right marginal frequencies $N(*, w)$ and $N(w, *)$. The result is an explict dependency graph for every possible sentence. That the ensemble is an explict Boltzmann distribution is a primary point of figure 2.

One could have approached this problem from the Monte Carlo direction as well, as is done by McDonald[17, 18]: one trains up a model by defining a cost function, and then performs hill climbing or gradient descent on it. The result is still a pair-wise cost function, and a dependency parse is obtained by searching for an MST tree, just as before. For McDonald, there is no overt relationship to the principle of maximum entropy; it is either covert, or perhaps one could say its "more general" by not constraining to fit the cost function to a probabilistic framework. The need for this generality does not seem to be well-supported. There does not appear to be any data to indicate if some other pair-wise cost function provides a superior MST-tree, as compared to using the MI. Although this is an open question, there's no particular hint that this could be a fruitful direction to take.

5.2 Graph Algorithms vs. Gradient Descent for N-grams

The OpenCog language learning project[22] proposes the creation of a disjunct model of natural language by means of graphical algorithms. Specifically, this means that the probabilities of disjuncts are obtained by direct counting. After a large body of text, and obtaining an observational count $N(w, d)$ of how often the word w is assocaited with the disjunct d, one computes a normalized frequency of observations. For problems like the Ising model, we want to solve the inverse problem.

xxx fixme nonsequiter missing text xxx

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

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. The energy functional is explicitly

$$E(w, d) = -\log_2 p(w, d)$$

The enemble of all possible grammatical word-sequences has the form of a sheaf.[12] The most likely relationship between the words in a sentence is the one that maximizes the entropy, including the contribution of the vacuum; and thus looks like a maximization of mutual information. Because disjuncts impose a complex relationship between words, one must use more complex algorithms, generically called "parsers", to discover how to attach disjuncts to one-another to obtain a valid nateork graph. The original Link grammar papers[3, 4] provide an eminently suitable parsing algorithm that is well-suited to the task.

By contrast, the CBOW/SkipGram models use gradient-descent to obtain a probability similar to $p(w,d)$. The notation used is different, so that $p(w_j|w_I)$ is written, with the N-gram context w_I as a stand-in for the disjunct d. Previous 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 grammatical structure that d encodes; there is some buried graph-like structure in w_I but it is not overt, and one must work to make it overt. One can use the disjuncts directly, with a suitable parsing algorithm, to obtain an explicit dependency parse of any given sentence. The actual probability function $p(w_j|w_I)$ is treated as an unknown, to be obtained by gradient descent algorithms, constrained only by the core assumption of either eqn 2 or eqn 3, so that the resulting probabilities are valid Boltzmann distributions.

5.3 Avoiding Maximum Entropy Principles

The previous sections explicitly focused on maximum entropy as a guiding principle, emphasizing that it is compatible with both a graphical algorithm approach, as well as a Monte Carlo or gradient descent approach to learning a language model. This guiding principle can be set to the side and ignored; there is nothing specific in either the graph algorithms or the training algorithms that require or assume it.

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 [16]. 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.[8] 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.

[8] Properly speaking, mutual information is not a metric, since it does not obey the triangle inequality. Thus, it should be called a divergence (it is essentially the Kullback–Liebler dirvergence) in that it distinguishes points by thier divergence, without providing a specific distance betwen them. Despite this, the divergence will be called either a matric or a distance in much of what follows, simply because it rolls off the tongue more easily.

- 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.

5.4 Summary

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 successful in what they do. The disadvantage is that they obscure the explicit graphical structure of natural language.

6 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.

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

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?

6.1 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[23] 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 number of non-zero pairs (w, d), 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.

Two more factors compound the confusion. One is that the observed text will necessarily contain grammatically-incorrect text: the occasional mis-spelled word, the occasional awkwardly worded phrase; omitted determiners, incorrect number, tense agreement. A far more serious issue is that the disjuncts are constructed by means of MST parsing, which has a reasonably large error rate: based on reports from Yuret[16] and others, one can expect parses that are 80% to 90% accurate; the fraction of incorrect disjuncts may range from 5% to 20%. Incorrect disjuncts decrease the observed sparsity: they make some observations more frequent than they should be, but only because they're wrong.

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.

6.1.1 No Large Data Limit

Natural language does not have a large-data limit. More generally, Zipf distributions cannot have a large-data limit. This is in contrast to normal distributions, where the large-data limit is a Gaussian.

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.

6.2 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

$$p(w,d) \approx \sum_{g \in G} p'(w|g)\, p''(g,d) \tag{6}$$

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 joint 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 one is which should be apparent from context. The probability $p(w|g)$ is a conditional probability: $p(w|g) = p(w,g)/p(*,g)$. This is used to ensure that the factorization of eqn 6, as well as the factors, all behave correctly as a joint probabilities:

$$\begin{aligned} p(*,d) &= \sum_w p(w,d) \\ &= \sum_w \sum_g p(w|g)\, p(g,d) \\ &= \sum_w \sum_g \frac{p(w,g)}{p(*,g)} p(g,d) \\ &= \sum_g p(g,d) \end{aligned}$$

and all joint probabilites normalize correctly: $p(*,*) = 1$.

There are two ways of performing the factorization of eqn 6: 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.

6.2.1 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.

6.3 Clustering

A side effect of the matrix factorization of eqn 6 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 6 can be seen as an alternative to deploying a clustering algorithm to assign words to word classes.

6.3.1 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.

6.3.2 k-means clustering

Clustering, and in particular, k-means clustering, can be shown to be equivalent to matrix factorization.[24] In particular, the equations that define k-means as a relaxation problem, of aligning vectors to the closes centroids, can be written explicitly as matrices. 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.

6.3.3 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*[25] 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 their neighbors; or one can remove edges that are outliers from the typical edge-length mean.

6.3.4 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 leveled 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 language learning problem, the data lives in an extremely high-dimensional space, with clusters that are almost surely convex.

6.4 Low Rank Matrix Approximation

Factorizations of the form of eqn 6 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),[26] 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 6 for probabilities, coupled to the need for fairly crisp word-sense disambiguation.

6.4.1 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.

The assumption that the matrix entries are normally distributed are also in a direct collision course with the known fact that the distribution of the matrix

entries are Zipfian, as argued in section 6.1.1. Although one can compute a mean or expectation value for the $p(w,d)$, creating a histogram around this mean value will not reveal a curve in the shape of a Gaussian.

Some words occur with frequencies that are many orders of magnitude larger than others. The Frobenius norm then becomes very strict for the former, and very loose for the latter. It leads to a situation where improving the factorization by 1% for a high frequency word is equivalent to being wrong by factors of 2 or 3 on low-frequency words. This does not seem like a plusible weighting scheme. The Frobenius norm seems less than ideal for the problem at hand.

6.4.2 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,[27] where orthogonal or unitary transformations can be applied. From an initial, naive point of view, this seems appropriate for machine learning. 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 can be applied.

The 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.

6.4.3 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 matrices 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[28], 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.

NMF 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.[29]

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 matrices:

$$MI_{\text{factor}} = \sum_{w,d} p(w,d) \log \frac{p(w,d)}{\sum_{g \in G} p(w|g)\, p(g,d)} \tag{7}$$

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.

6.4.4 Projective Probability Amplitudes

The previous sections argue that when vectors are interpreted as probabilities, then dot-products and invariance under orthogonal transformations are inappropriate, because these transform probabilities into things that cannot be interpreted as probabilities. Intuitively, information-theoretic manipulations that preserve the probability aspects seem wiser. However, there is another possibility: work with probability amplitudes.

A vector is a probability, if one has that

$$\sum_n p_n = 1$$

That is, a point $\vec{p}$ is constrained to live on the surface of a simplex. One can apply a simple trick: just write $a_n = \sqrt{p_n}$ and then the constraint becomes

$$\sum_n a_n^2 = 1$$

Clearly, the point $\vec{a}$ is constrained to live on the surface of a sphere. Now, orthogonal rotations do not violate this constraint.

This implies that nurclear norms, or Frobenius norms might be quite appropriate, if, instead of working with $\vec{p}$, one worked with $\vec{a}$. To be more specific, define $a(w,d) = \sqrt{p(w,d)}$ and then define an error matrix (residual matrix)

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

and then, to assign word w to word-sense cluster g, seek to minimize the objective function

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

This seems like a plausible objective function, if the $a(w,d)$ are normally distributed. But, as argued in section 6.1.1, this is a faulty assumption. As noted before, this error matrix causes high-frequency words to strongly dominate over low-frequency words in the factorization. Again, this seems like an implausbile objective function for natural language or any Zipfian distributions.

6.4.5 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

$$f(q_1, q_2, \cdots, q_n)$$

where $q_g = p(w|g)\, p(g,d)$ and $n = |G|$ the number of elements in G. Thus, the matrix factorization is just the function $f(q_1, q_2, \cdots, q_n) = q_1 + q_2 + \cdots + q_n$. The neural net matrix factorization[30] 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 modeling or leaving unexplored some deeper linear structure.

6.4.6 Local Low Rank Matrix Factorization

Local Low Rank Matrix Factorization (LLORMA)[31] 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[32] and local regression (LOESS)[33] 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 language), 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 6.1.1. 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 7. It seems reasonable to beleive that combining kernel smoothing with local regression and the information-theoretic norm might yeild an excellent algorithm for factoring joint probabilities. As an algorithm, it belongs to the gradient-descent class.

6.4.7 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.

6.5 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.*[24] 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 3, left image), and that one can perform separate, distinct, but simultaneous clustering on the columns, and separately, the rows. This is closer to the desired factorization model for language, and 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 6, 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 matrices, 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

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

Hard clustering means that any given i belongs to only one j, and so one trivially has that $\sum_j L_{ij} = 1$, which is a key property of a Markov matrix. That is, hard clustering has the Markov property. Equivalently, the factorization for the hard-clustering of words into single classes is

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

That is, every word belongs to some class, with 100% probability.

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

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

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 matrices 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; indeed, 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 similarity, 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 matrices

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

6.6 Factorization Ambiguity

When considering the factorization of eqn 6, 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 6 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

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

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 matrices, 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 joint probability

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

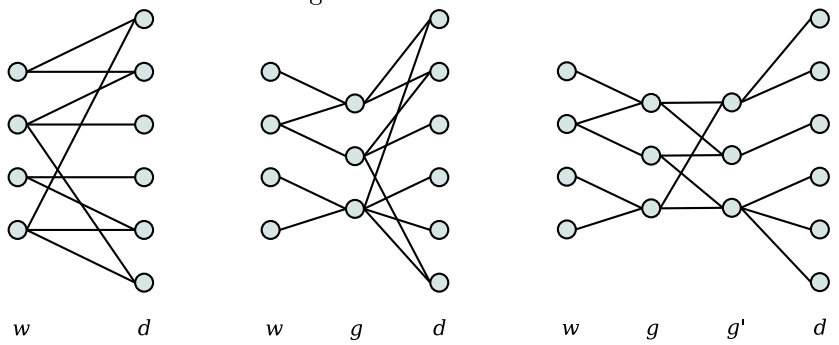
is still a valid joint probability distribution; namely, that $p'(*,*) = 1$.

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 3.

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 categories. 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 achieved by factoring as

$$p(w,d) = \sum_g \sum_{g'} p(w|g)\, p(g,g')\, p(d|g') \tag{8}$$

Figure 3: 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.

---

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

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

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 $\sum_{g'} p(d|g') = 1$. 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.[9]

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.[34] and is reviewed in a subsection below. Of course, for natural language, we want to decompose words into word-senses, and so naive hard-clustering will not work. It does, however, illuminate the path ahead.

6.6.1 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 $g' = \langle noun\text{-}main\text{-}p\rangle$. It appears multiple times throughout the file. In one case, it is associated with

[9] 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 spontaneously 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.

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.

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 meaning) are factored away from the suffixes (which provide the needed tense, gender, number and person agreement across sentences).

6.6.2 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.

6.6.3 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 somehow 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,[35, 36] but the resulting syntax constraints must force the selection of the predicate-argument structure.[37] 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 interchangeable, 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.

6.6.4 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.

6.6.5 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 reorganization 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.

6.6.6 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 7:

$$MI_{Loss} = \sum_{w,d} p(w,d) \log_2 \frac{p(w,d)}{\sum_{g\in G}\sum_{g'\in G'} p(w|g)\, p(g,g')\, p(d|g')} \tag{9}$$

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

The above information loss estimate is identical to that described by Dhillon *et al.*[34] in their 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 their hard-clustering assumption. Words cannot be hard-clustered; word-vectors must be decomposed into word-sense vectors first.[38] 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) of Dhillon)

$$p(g,g') = \sum_{w\in g}\sum_{d\in g'} p(w,d) \tag{10}$$

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

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

and so eqn 9 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).

6.6.7 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 10. 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 iteration. 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 searching for the clustering $d \in g'$ 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 6.1. There are in excess of 10^5 words to provisionally assign to clusters, and 10^7 or more disjuncts. Each provisional clustering requires extensive re-computation 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. Genetic algorithms are not enough. Back to the drawing board.

6.6.8 Word-Sense Disambiguation

The primary issue with hard clustering is that it fails to correctly disentangle different word senses. There is an opportunity to do better.

A prototype example is given by the word "saw". It could be the noun, referring to the cutting tool; it could be the verb synonymous to cutting; it could be the past tense of the verb "to see". The observational data consists of occurrence counts $N(\text{"saw"}, d)$. The goal of word-sense disambiguation is to somehow factor this into three grammatical classes g, so that

$$N(\text{"saw"}, d) = N(\langle \text{tool} \rangle, d) + N(\langle \text{cutting} \rangle, d) + N(\langle \text{seeing} \rangle, d)$$

The general problem of how to accomplish this, and several tactics are discussed in [38]. That presentation focuses on cosine distances. This section provides an information-theoretic variant.

The actual observed frequencies $p(w, d)$ are, by definition, sums of word w used with every different possible word-sense s_k that the word could have. That is, explicitly, one has that

$$p(w, d) = \sum_k p(s_k, d)$$

In poetry and in word-play, a word might be used in such a way that multiple senses are simultaneously applied. The assumption above is that this is not the case: that the text assigns only a single meaning to each word use. In any given use, the word "saw" is either a noun, or one of the verbs; it cannot be some mixture of all of them.

Each word-sense can then be hard-clustered into a single grammatical category: "saw"-the-noun belongs to just one grammatical category, the one that holds synonyms for cutting tools. Thus, the hard-clustering formula applies:

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

The intent of the sense label s is that it is uniquely associated with just one word, and no others. The number of word-senses is strictly larger than the vocabulary: every vocabulary word has at least one sense. That is, $p(*, s) = p(w, s)$ holds for all $s \in w$. Thus, the conditional probability is

$$p(w|s) = \frac{p(w, s)}{p(*, s)} = \begin{cases} 1 & \text{if } s \in w \\ 0 & \text{otherwise} \end{cases}$$

The decomposition of words into senses is then (restating the earlier formula in slightly different notation:

$$\begin{aligned} p(w,d) &= \sum_s p(w|s)\, p(s,d) \\ &= \sum_{s \in w} p(s,d) \end{aligned}$$

To be consistent with the central factorization 8, one must have

$$\sum_{s \in w} p(s,g) = p(w,g)$$

and

$$\sum_{s \in w} p(s|g) = p(w|g)$$

Since a word-sense is associated with just one word, the notation $\sum_{s \in w}$ is superfluous: one can unambiguously write $\sum_s$ as all other entries are zero.

The central factorization now has the form

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

Not much has changed: the word-senses can now be legitimately hard-clustered; but the decomposition of words into word-senses is unknown. The clustering algorithm reviewed in section 6.6.7 is very nearly unaffected: one replaces the provisional clustering guesses of $w \in g$ by provisional guesses $s \in w$ and $s \in g$. The feasibility of the algorithm is further challenged by the fact that there are more word-senses than there are words, further increasing the computational space.

6.7 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.

6.7.1 Control of Graphical Structure

The goal of the matrix factorization, illustrated in figure 3, 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 6.6.1, 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.

7 Factorizing the language model

The above developments now provide enough background to clearly state the problem. Inspired by the factorization of eqn 6, 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 8, as illustrated in figure 3. The appropriate measure of quality is to minimize the information loss in the factorization, as given by eqn 9.

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 [38].

7.1 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 warranted. Two observations pop out:

- None of the systems described in the literature assume that the input data can already be interpreted as a probability; rather, the clustering is being performed on data naturally occurring 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.*[39] 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.

7.1.1 Renyi Information Divergence

Gokcay and Principe[40] propose the Renyi entropy as an information divergence. It is essentially minus the logarithm of the cosine metric. Given two vectors $\vec{u}$ and $\vec{v}$, 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 believe 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 language data. The vectors of in the contingency table that the language data is organized 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 [41], it becomes quite clear that there is a practical, common-sense cutoff for similarlity. 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 common 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 appearance of the logarithm does nothing to advance the situation.

7.1.2 Consistency Under Coarse-graining

To deal with the problem of factorization ambiguity, Ver Steeg *et al*[39] 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 language 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 [41], 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.

7.2 Information-driven Clustering

As shown by Ding *et al*[24] and reviewed in section 6.5, 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.

7.2.1 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 [24]) 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.

7.2.2 Pair-wise Information Divergence

The cosine was built from the dot product $\vec{u} \cdot \vec{v}$ normalized by the vector lengths. For information-based clustering, the normalization is changed, so as to use the Kullback-Leibler divergence between the vector-pair, and the individual vectors:

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

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

$$MI(w_1, w_2) = \log_2 \frac{Q(w_1, w_2)}{Q(w_1, *) Q(*, w_2)} \tag{11}$$

Intuitively. this can be thought of the information gain of bringing two vectors together, as compared to the entire dataset of all other vectors.

This global aspect makes this function very different from the cosine distance. The cosine distance is defined independently of all other vectors in the system; it is independent of the number of other vectors, and the directions in which they are pointing. On could have a system with 42 vectors, or with 23 thousand of them: the cosine metric doesn't care. One could have a system with two vectors pointing one way, and another six hundred pointing in the opposite direction: cosine doesn't care. The pair-wise MI above compares two vectors, within the context of what all of the other vectors in the system.

A different way to think of the pair-wise MI is that, by incorporating all other vectors into its value, it is effectively attempting to spread all of the (other) vectors into as uniform distribution as possible, magnifying and expanding those regions where cosine would say there are lots of densely packed vectors, while shrinking those regions where there is low density.

One way the above can be visualized is to consider two vectors, roughly collinear, and another six hundred, also all roughly collinear, but almost orthogonal to the first two. Consider then the MI values for different pairs chosen from this collection. Two vectors which might have the same cosine angle will have very different MI values, in this kind of dataset.

7.2.3 Scale invariance

The information divergence $MI(w_1, w_2)$ is scale invariant: replacing $p(w, d)$ by $kp(w, d)$ for some constant k does not alter the divergence. In particular, replacing the joint probability $p(w, d)$ by the conditional probability $p(d|w) = p(w, d) / p(w, *)$ does not alter the information divergence.

7.2.4 Arithmetic and Geometric Means

For vectors obtained from natural language, there is good reason to believe that the natural clusters are convex, rather than being oddly shaped. Therefore, it is meaningful to ask about and talk about the centroid of a cluster.

For a Euclidean space, possessing rotational invariance and some notion of uniform distribution, the arithmetic mean of the vectors in a cluster would

be a natural choice for the centroid. However, as is being repeatedly noted, vectors taken from a joint probability distribution do not live in a Euclidean, rotationally symmetric space. The arithmetic mean does not make sense. There are several candidates that are more suitable.

One is the geometric mean. The centroid for word-cluster g might be expressed by

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

This has the peculiar property that, if for any word w in the cluster, $p(w,d) = 0$, then $m(g,d) = 0$. The support for the vector $m(g,d)$ is the intersection of the supports for the vectors $p(w,d)$. In itself, this is not terribly objectionable or bad, but for one thing: The $p(w,d)$ are observational frequencies, and are not theoretical distributions. Thus, one might have that $p(w,d) = 0$ simply because it has not been observed, and not because it is grammatically forbidden. This was already noted in section 6.1: contingency tables fundamentally cannot distinguish between rare linguistic phenomena and grammatically forbidden phenomena. The Zipfian distribution prohibits this.

One can imagine several work-arounds. One would be to apply some form of kernel smoothing, or to apply some sort of local regression. How this might work is unclear. Consider instead the logarithm of the geometric mean:

$$\log_2 m(g,d) = \frac{1}{|g|} \sum_{w \in g} \log_2 p(w,d) \tag{12}$$

To be consistent with the previous expression, if $p(w,d)$ has not been observed, one should take $\log_2 p(w,d) = -\infty$. From the missing-data perspective, it would actually be better to take $\log_2 p(w,d) = 0$ – missing observations contribute nothing. Continuing in this vein of thinking, that infrequently-observed phenomena are being unfairly punished and are not being given due weight, one might consider the centroid to be described by

$$\log_2 m_\alpha(g,d) = \frac{1}{|g|} \sum_{w \in g} (\log_2 p(w,d))^\alpha$$

for some $\alpha \leq 1$, thus effectively making rare phenomena seem less rare, while common phenomena become less common.

An intuitive feel for the geometric mean can be argued: probabilities can only be added if they are truly independent, and here they are not; but probabilities, taken as discrete events, can always be multiplied. For a a probability distribution $P(X)$ on a space X, the frequentist interpretation of probability is founded on individual observations in the sequence space (aka the Cantor space) $X^\omega = X \times X \times \cdots$ that is the Cartesian product of an infinite number of copies of X. Frequentist probabilities are sigma-measures on the Borel set of cylinder sets on the Cantor space. Cylinder sets are the elements of the coarse topology on the product space. Cylinder sets correspond to the limits of the pushout diagram $\bullet \leftarrow \bullet \rightarrow \bullet$ that defines the Cartesian product.

7.2.5 Agglomerative Clustering

The primary reason to propose a pair-wise information divergence is to enable agglomerative clustering. The reason for this was made clear earlier: the sheer scale and quantity of data makes hill-climbing and gradient descent algorithms untenable. The search space is simply too large.

As a corollary, the results of hill-climbing or gradient-descent algorithms are insufficiently sparse; one wishes to approximate the sparsity that hard-clustering offers, with less overhead and complexity in finding the clusters.

7.3 Word-Sense Disambiguation

The section 6.6.8 describes the general, global framework for discussing word-senses in the factorization model. The challenge of agglomerative clustering is to obtain some estimate of word-sense assignments, given as a starting point the pairwise information divergence of eqn 11. This is where things start to get truly interesting.

7.3.1 Mihalcea Algorithm

Rada Mihalcea describes a word-sense disambiguation algorithm[42, 43, 44] that fits very naturally with the framework being discussed here. However, it requires whole-sentence contexts. For each instance w_j of the j'th word in a sentence, one assigns an (unknown) vector of word-senses $p(w_j, s_k)$ with the set of possible senses s_k given externally, as an *a priori* set, for example, taken from WordNet. In addition, one also assumes an externally provided *a priori* measure of similarity $d(s_k, s_m)$ of word-senses; again, this can be provided by WordNet. One then considers the full graph clique of all possible word-senses that might appear in the sentence, and their relation all other possible word-senses. That is, one considers the graph $M(s_k, s_m)$ with the k and m indexes corresponding to word-positions in the sentence; the matrix entries are just $d(s_k, s_m)$ normalized so that M is a Markov matrix (Markov chain). One then solves this chain to find the stationary vector π. This stationary vector can be interpreted as the desired word-sense assignment $p(w_j, s_k)$ of the probability that the j'th word has the k'th sense.

The algorithm readily generalizes to any system that can provide logical inference on concepts: one replaces the similarity-distance $d(s_k, s_m)$ by the likelihood of some particular logical inference performed over the sentence. Taken over multiple sentences, this can provide anaphora resolution and reference resolution.

The primary drawback of this algorithm is that the word senses must be externally supplied. For this project, this is even a fatal drawback: the goal is to infer word-senses.

7.3.2 Word-sense Similarity

Assume, for a moment, that distinct word senses can be inferred from raw text, in an unsupervised fashion. That is, assume that a joint probability $p(s, d)$ can be inferred from the text data. It is OK if this probability is of low quality, and generally inaccurate: disappointingly inaccurate, even. If this probability exceeds pure random chance, then the Mihalcea algorithm can be run "in reverse", to infer the joint probabilities $p(s, s')$ between different senses s and s'. One does so by assuming that, given the specific words in a specific sentence, that the word-senses between them are necessarily highly-related. Given that the sentence can be parsed into a string of disjuncts d, one can use $p(s, d)$ to assign a provisional probability s to each word, and then collect observational statistics $N(s, s')$ on a sentence-by-sentence basis, incrementing N by one for each observed pair (s, s') in a sentence. After observing a large number of sentences, one has a (sparse) database of sense-pair frequencies $p(s, s')$.

The goal of doing this is to amplify the signal-to-noise ratio. This is in more-or-less complete analogy to the case for word-pairs: given a small number of observations, they are noisy and inaccurate. Increasing the number of observations causes the noise to cancel out, and for a signal to emerge.

At this point, one can start doing some intriguing things: one can attempt to perform maximum-spanning-tree parses, to determine how one given word-sense is related to all other word-senses in a given sentence. That is, one repeats the process of extracting disjuncts, but this time, the connectors on the disjuncts are not words, but word-senses.

The reason for extracting sense-disjuncts is that they provide the doorway to obtaining the "Lexical Functions"[45] of Meaning-Text Theory (MTT).[46, 47] From there, one can extract the **DSyntR** structure of a sentence, which provides the natural input into a reasoning system based on formal logic. That is, one can now begin to assemble logical proofs by inferring sequences of deductions applied to sense-disjuncts. Or rather, more importantly, once can now start to infer the set of valid logical deductions that can be applied, rather than taking different forms of logical inference as being given *a priori*. To be more blunt: this provides the pathway for inferring the rules of Goertzel's PLN, or for inferring the rules Pei Wang's Non-Axiomatic Logic, rather than taking these rules as externally imposed.

7.3.3 Word-sense Factoring

Given a pair-wise information divergence, how can one infer word senses? This is addressed in the companion text.[38]. However, that text is currently written in terms of the cosine-distance; it needs to be re-expressed using the information divergence, and, in particular, by the use of eqn 12 for the cluster mean.

So XXX TODO. Unfinished things begin here.

8 Unfinished Thoughts

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

8.0.1 Ising Model

Solving the Ising model by gradient descent can be done; a good way to do this is by using message passing.[19] This is a particularly quickly-convergent mean-field method. The problem is that it fails to converge when the coupling is strong – in other words, when the network is glassy (in the sense of "spin glass"). The question: is language in a spin-glass state? Hypothesis: yes, it is. Question: how can this be demonstrated?

8.0.2 Surprisal Analysis

The basic idea is this formula:

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

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".

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

the−

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

<common−nouns>: 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 $P(\text{sentence}\,|\,\text{fulltext})$, but due to the apparent computational intractability, one beats a tactical retreat to computing $P(\text{word}\,|\,\text{context})$ 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 $P(\text{phrase}\,|\,\text{context})$ or even $P(\text{sentence}\,|\,\text{context})$. 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 $P(\text{sentence}\,|\,\text{fulltext})$, with, at the outset, seemed hopelessly intractable.

TODO flesh out this section.

The important parts of the sheaves idea are already covered in the "stitching" paper.[38] They need to be transcribed here.

Gluing axioms

The language-learning task requires one to infer the structure of language from a small number of instances and examples. Bengio *etal.*[6] 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] Wikipedia, "Bayesian Network", URL https://en.wikipedia.org/wiki/Bayesian_network. 2

[2] Wikipedia, "Link Grammar", URL https://en.wikipedia.org/wiki/Link_grammar. 2

[3] 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, 7, 26, 30

[4] 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-IWPT93.ps. 2, 7, 26, 30

[5] Wikipedia, "Word2Vec", URL https://en.wikipedia.org/wiki/Word2Vec. 2

[6] Y. Bengio, et al., "A neural probabilistic language model.", *Journal of Machine Learning Research*, 3, 2003, pp. 1137–1155. 2, 11, 15, 64

[7] Tomas Mikolov, et al., "Distributed Representations of Words and Phrases and their Compositionality", *arXiv/13104546*, 2013. 2, 11, 13, 19, 21

[8] Tomas Mikolov, et al., "Efficient Estimation of Word Representations in Vector Space", *arXiv/13013781v3*, 2013. 2, 13, 21

[9] Wikipedia, "Valency", URL https://en.wikipedia.org/wiki/Valency_(linguistics). 3

[10] Wikipedia, "Partition function", URL https://en.wikipedia.org/wiki/Partition_function_(mathemtics). 5

[11] Solomon Marcus, *Algebraic Linguistics; Analytical Models*, Elsevier, 1967, URL https://monoskop.org/images/2/26/Marcus_Solomon_editor_Algebraic_Linguistics_Analytical_Models_1967.pdf. 5

[12] Linas Vepstas, "Sheaves: A topoloogical Approach to Big Data", , 2017, URL https://github.com/opencog/atomspace/raw/master/opencog/sheaf/docs/sheaves.pdf, draft. 6, 9, 30

[13] Alex Minnaar, "Word2Vec Tutorial Part II: The Continuous Bag-of-Words Model", , 2015, URL http://mccormickml.com/assets/word2vec/Alex_Minnaar_Word2Vec_Tutorial_Part_II_The_Continuous_Bag-of-Words_Model.pdf. 15

[14] Thierry Mora, et al., "Maximum entropy models for antibody diversity", *Proceedings of the National Academy of Sciences*, 107, 2010, pp. 5405–5410, URL http://www.pnas.org/content/107/12/5405. 21

[15] Martin Weigt, et al., "Identification of direct residue contacts in protein–protein interaction by message passing", *PNAS*, 106, 2009, pp. 67–72, URL http://www.pnas.org/content/106/1/67. 22, 25

[16] Deniz Yuret, *Discovery of Linguistic Relations Using Lexical Attraction*, PhD thesis, MIT, 1998, URL http://www2.denizyuret.com/pub/yuretphd.html. 23, 30, 34

[17] Ryan McDonald, et al., "Non-projective Dependency Parsing using Spanning Tree Algorithms", in *HLT 05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2005, pp. 523–530, URL https://www.seas.upenn.edu/~strctlrn/bib/PDF/nonprojectiveHLT-EMNLP2005.pdf. 23, 29

[18] Ryan McDonald, et al., "Multilingual dependency analysis with a two-stage discriminative parser", in *CoNLL-X '06: Proceedings of the Tenth Conference on Computational Natural Language Learning*, Association for Computational Linguistics, Morristown, NJ, USA, 2006, pp. 216–220. 23, 29

[19] Marc Mézard and Thierry Mora, "Constraint satisfaction problems and neural networks: a statistical physics perspective", , 2008, URL https://arxiv.org/abs/0803.3061, arXiv abs/0803.3061. 25, 63

[20] Richard Hudson, *Word Grammar*, Oxford: Blackwell, 1984. 27

[21] Richard Hudson, *Language Networks: The New Word Grammar*, Oxford Linguistics, 2007. 27

[22] Ben Goertzel and Linas Vepstas, "Langauge Learning", , 2014, URL https://arxiv.org/abs/1401.3372, arXiv abs/1401.3372. 29

[23] Linas Vepstas, "Language Learning Diary", , 2013, URL https://github.com/opencog/opencog/raw/master/opencog/nlp/learn/learn-lang-diary/learn-lang-diary.pdf, unpublished. 33

[24] 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-SDM2005.pdf. 38, 43, 57, 58

[25] 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_hierarchies_of_clusters_from_the_euclidean_minimum_spanning_tree.pdf. 38

[26] Wikipedia, "Non-negative matrix factorization", URL https://en.wikipedia.org/wiki/Non-negative_matrix_factorization. 39

[27] Wikipedia, "Homogenous space", URL https://en.wikipedia.org/wiki/Homogeneous_space. 40

[28] 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_Sparse_coding_and_NMF. 41

[29] 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. 41

[30] Gintare Karolina Dziugaite and Daniel M. Roy, "Neural Network Matrix Factorization", , 2015, URL https://arxiv.org/abs/1511.06443, arXiv abs/1511.06443. 42

[31] Joonseok Lee, et al., "LLORMA: Local Low-Rank Matrix Approximation", *Journal of Machine Learning Research*, 16, 2016, pp. 1–24. 42

[32] Wikipedia, "Kernel smoother", URL https://en.wikipedia.org/wiki/Kernel_smoother. 42

[33] Wikipedia, "Local regression", URL https://en.wikipedia.org/wiki/Local_regression. 42

[34] 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_papers/kdd_cocluster.pdf. 48, 51

[35] Wikipedia, "Predicate", URL https://en.wikipedia.org/wiki/Predicate_(grammar). 50

[36] Wikipedia, "Argument", URL https://en.wikipedia.org/wiki/Arguments_(linguistics). 50

[37] Wikipedia, "Selection", URL https://en.wikipedia.org/wiki/Selection_(linguistics). 50

[38] Linas Vepstas, "Stiching Together Vector Spaces", , 2018, URL https://github.com/opencog/opencog/raw/master/opencog/nlp/learn/learn-lang-diary/stitching.pdf, draft. 51, 53, 56, 62, 64

[39] Greg Ver Steeg, et al., "Demystifying Information-Theoretic Clustering", , 2014, URL https://arxiv.org/abs/1310.04210, arXiv abs/1310.04210. 56, 57

[40] 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/f706/5b6edb59fb19d987c5d791e73514528ce0ca.pdf. 56

[41] Linas Vepstas, "Connector Set Distributions", , 2017, URL https://github.com/opencog/opencog/raw/master/opencog/nlp/learn/learn-lang-diary/connector-sets-revised.pdf, draft. 57

[42] Rada Mihalcea, et al., "PageRank on semantic networks, with application to word sense disambiguation", in *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, Association for Computational Linguistics, Morristown, NJ, USA, 2004, pp. –, URL http://web.eecs.umich.edu/~mihalcea/papers/mihalcea.coling04.pdf. 61

[43] Rada Mihalcea, "Unsupervised Large-Vocabulary Word Sense Disambiguation with Graph-based Algorithms for Sequence Data Labeling", in *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Morristown, NJ, USA, 2005, pp. 411–418. 61

[44] Ravi Sinha and Rada Mihalcea, "Unsupervised Graph-basedWord Sense Disambiguation Using Measures of Word Semantic Similarity", in *ICSC '07: Proceedings of the International Conference on Semantic Computing*, IEEE Computer Society, Washington, DC, USA, 2007, pp. 363–369. 61

[45] Wikipedia, "Lexical function", URL https://en.wikipedia.org/wiki/Lexical_function. 62

[46] Igor A. Mel'cuk and Alain Polguere, "A Formal Lexicon in Meaning-Text Theory", *Computational Linguistics*, 13, 1987, pp. 261–275. 62

[47] Sylvain Kahane, "The Meaning-Text Theory", *Dependency and Valency An International Handbook of Contemporary Research*, 1, 2003, pp. 546–570, URL http://www.coli.uni-saarland.de/courses/syntactic-theory-09/literature/MTT-Handbook2003.pdf. 62