

Skip-grams vs. Disjuncts

Linas Vepstas

6 July 2018

Abstract

This document attempts to compare and contrast the various ways in which (adaptive)-skip-grams and disjuncts are similar, and how they differ. The comparison is not always obvious, as these two representations seem to be quite different, the goals and tasks that each are solving seem to be different, and the methods to obtain each seem to be quite different. In fact, many of these differences are superficial; there is more in common than there might seem.

More generally it tries to develop and contrast the differences between graphical methods and gradient-descent methods in deeper layers of language abstraction. In particular it begins to compare the extraction of semantic meaning using gradient descent algorithms, such as Low Rank Matrix Approximations, to graphical factorizations driven by information-theoretic metrics.

This is a work in progress. It's approximately half-done, I think.

1 Introduction

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

1.1 Link Grammar

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

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

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

connectors are conjoined. A given word can have multiple disjuncts associated with it, these are disjointed from one-another, thus the name “disjunct”.

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

ran: girl- & home+;

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

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

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

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

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

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

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

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

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

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

The correct conceptual model for the observational data is that of a large network graph, with observation counts attached to each vertex. This network graph can be understood as a sheaf (see reference). The above examples show

how certain sections of that graph can be made to correspond to vectors. Obviously, these vectors are not independent of one-another, as they are all different slices through the same dataset. Rather, they provide a local, linear view of the language graph, reminiscent of the tangent-space of a manifold.

1.2 Statistical Models

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

$$\hat{P}(w_1^T) = \prod_{t=1}^T P(w_t | w_1^{t-1})$$

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, $w_i^n = (w_i, w_{i+1}, \dots, w_n)$. Thus, the text as a whole is denoted by w_1^T , and so $\hat{P}(w_1^T)$ is an approximate model for the probability $P(w_1^T)$ of observing the text (the carat over P serving to remind that approximations are being made; that the model is an approximation for the “true” probability.)

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

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

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

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

$$\hat{P}(w_1^T) = \prod_{t=1}^T p(w_t | w_{t-c}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c})$$

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

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

1.3 N-Gram Model

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

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

1.4 Model Building

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

There is a fairly rich variety of models. Reviewed immediately below are two common foundational models: the so-called CBOW model, and the SkipGram model. The general goal of this paper is to demonstrate that Link Grammar,

and thus dependency grammars in general, can be understood to also fit into this same class of probabilistic models. What differs is the mechanism by which the models are trained; the Link Grammar training algorithm, already sketched above, is not a hill-climbing/deep-learning technique. A proper comparison will be made after the initial review of the CBOW and SkipGram models.

1.5 CBOW

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

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

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

In the CBOW model, the mapping from the space of words to the space of vectors \vec{v}_w is linear; there are no non-linear functions, as there would be in a neural net. That is, the mapping is given by a matrix π of dimension $D \times W$. Maps from higher to lower dimensional spaces are called “projections”. (The notation of the lower-case Greek letter π for projection is common-place in the mathematical literature, but uncommon in the machine-learning world. It’s convenient here, as it avoids burning yet another roman letter.) The projection matrix π is unknown at the outset; the goal of training is to determine it.

The CBOW is a model of the conditional probability

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

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 $I = \{-c, -c+1, \dots, -1, +1, \dots, +c\}$. By abuse of notation, one might also write, for offset t or for word w_t , that $I = \{t-c, t-c+1, \dots, t-1, t+1, \dots, t+c\}$ or that $I = w_I = \{w_{t-c}, w_{t-c+1}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+c}\}$; 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

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

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

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[5].

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

1.6 SkipGram

The SkipGram model is very similar to the CBOW model, and is commonly presented as it’s opposite. It uses essentially the same Boltzmann distribution as CBOW, except that it is now looking at the probability $p(w_I | w_t)$ of the context I given the target word w_t . Explicitly, the model is given by

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

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

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

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

1.7 Perceptrons as Neural Nets

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

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

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

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

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

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

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

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

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

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

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

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

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

2 Similarities and Differences

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

2.1 Disjuncts as Context

Consider the probability

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

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

`ran: girl- & home+;`

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

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

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

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

is simply the sum over all words in the vocabulary.

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

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

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

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

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

2.2 Skip-Grams and Grammar

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

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

Such long-distance attachments are problematic for CBOW or Skip-Grams, in several ways. One is that the window N must be quite large to skip over the post-nominal modifier. Counting punctuation, one must look at least seven words to the right, in the above example. If the window is symmetric about the target word, this calls for $N \geq 14$, which is a bit larger than currently reported results; for example, Mikolov[?] reports results for $N = 5$. The point here is that

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

can be trivially re-interpreted as the dictionary entry

girl: the– & ran+;

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

girl: the- & upset+ & ran+;

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

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

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

Another difficulty is that dependency grammars are naturally “adaptive” by design: verbs tend to have more attachments than 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

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

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

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

2.3 Training

One important difference between the earlier description of Link Grammar, and the deep-learning algorithms should be immediately apparent: the Link Grammar probabilities are obtained by direct counting, and not by any training, relaxation or hill-climbing technique. That is,

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

so that the probability is properly normalized: $p(*, *) = 1$. The $N(w, d)$ is just the observational count of observing the pair (w, d) in text; the probabilities is just the frequentist probability. This differs sharply from the probability in the CBOW/SkipGram models, which is obtained by maximizing an objective function (the “loss function”) built from (defined in terms of) the probabilities.

3 Model Building and Vector Representations

The key driver behind the deep-learning models is the replacement of intractable probabilistic models by those that are computationally efficient. This is accomplished in several stages. First, the full-text probability function $P(\text{sentence} | \text{fulltext})$

is replaced by the much simpler probability function $P(\text{word} | \text{fulltext})$. 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 $P(\text{word} | \text{bag})$ 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 $P(\text{word} | \text{fulltext})$ and simplify it by using instead a sliding-window probability function $P(\text{word} | \text{window})$, thus giving the N-gram model. The characteristic idea here is to explicitly set $P(\text{word} | \text{other-words}) = 0$ 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 $P(\text{word} | \text{window})$ can be ignored or conflated. Conceptually, the models propose computing $P(\text{word} | \text{context})$ with the context being a projection to a low-dimensional space. These ideas can be illustrated more precisely. Let

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

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

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

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

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

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

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

where the matrix S is a concatenation of identity matrices.

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

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

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

Its worth getting an intuitive feeling for the size of these numbers: following Mikolov, assume that $W = 10^4$ although this sharply underestimates the size of the vocabulary of English. Assume $N = 5$ and $D = 300$. The size of the input vector space is thus $W^N = 10^{20}$, this is being modeled by a vector space of size 300. The sparsity is thus

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

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

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

3.1 Sparsity

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

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

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

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

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

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

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

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

3.1.1 Zipf’s Law; Noise and Incompleteness

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 every 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 matter 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 due to forbidden grammatical constructions.

3.2 Word Classes

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

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

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

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

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

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

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

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 would result in the outer product, or tensor product, of two vectors.

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

3.2.1 Learning Word Senses

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

girl: the-;

would be associated with $g = \langle \text{common} - \text{count} - \text{nouns} \rangle$. 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 $g = \langle \text{cutting} - \text{actions} \rangle$ and $g = \langle \text{looking} - \text{verbs} \rangle$; the assignment of

saw: I- & wood+;

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

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

3.3 Low Rank Matrix Approximation

Factorizations of this sort 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.

A variety of techniques for performing this factorization have been developed. A lightning review is given below.

3.3.1 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

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

and from this, the objective function

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

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

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

3.3.2 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, where orthogonal or unitary transformations can be applied. In machine learning, the spaces are always finite dimensional, and are usually explicitly assumed to be real Euclidean space \mathbb{R}^n – that is, the spaces behave like actual vector spaces, so that concepts like SVD apply.

A subtle point here is that the space in which $p(w, d)$ lives is *not* \mathbb{R}^n (nor is it $\mathbb{R}^{|W| \times |D|}$, if one is a stickler about dimensions). Rather, $p(w, d)$ is constrained to live inside of a simplex (of dimension $|W| \times |D|$). Sure, one can blur one's eyes and imagine that this simplex is a subspace of $\mathbb{R}^{|W| \times |D|}$, and that is not entirely wrong. However, the only transformations that can be applied to points in a simplex, that keep the points inside the simplex, are Markov matrices. Any other transformations will typically move points from the outside, into the inside, and move inside points out. In particular, rotations (orthogonal transformations) cannot be applied to a probability, with the result still being a

probability. Applying the notion of a trace, which is implicitly defined in terms of orthogonal transformations, is inappropriate for the problem at hand.

3.3.3 Neural Net Matrix Factorization

The factorization

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

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

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

where $a_g = p(w, g) p(g, d)$ and $n = |G|$ the number of elements in G . Thus, the matrix factorization is just the function $f(a_1, a_2, \dots, a_n) = a_1 + a_2 + \dots + a_n$.

The neural net matrix factorization[6] replaces the simple sum by a multi-layer feed-forward neural net.

There is nothing in the current formulation of the 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. It might make sense to model the tight central nucleus as a neural network of some sort.

3.3.4 Local Low Rank Matrix Factorization

Local Low Rank Matrix Factorization (LLORMA)[7] 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[8] and local regression (LOESS)[9] to obtain smooth estimates for the two factors $p(w, g)$ and $p(g, d)$.

The use of the smoothing techniques seems highly appropriate, given that the input dataset $p(w, d)$ is both noisy and incomplete, as discussed previously. 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 appears to be an information-theoretic norm, as discussed in upcoming sections.

3.3.5 Other factorizations

Other factorizations, including

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

3.4 Factorization Ambiguity

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 $(\vec{d})_g = p(g, d)$. Aside from considering just $\vec{w} \cdot \vec{d}$, one might consider other functions $f(\vec{w}, \vec{d})$ of \vec{w} and \vec{d} . A single-layer feed-forward linear neural net would consist of a $|G| \times |G|$ -dimensional weight matrix M so that

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

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 $\vec{w}' = M^T \vec{w}$ or $\vec{d}' = M \vec{d}$. Written in components:

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

and so one can contract M either to the left or to the right. 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. Thus, one can legitimately introduce another constraint: given the freedom in M , choose it so that $p(w, g)$ and $p(g', d)$ are both maximally sparse.

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

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

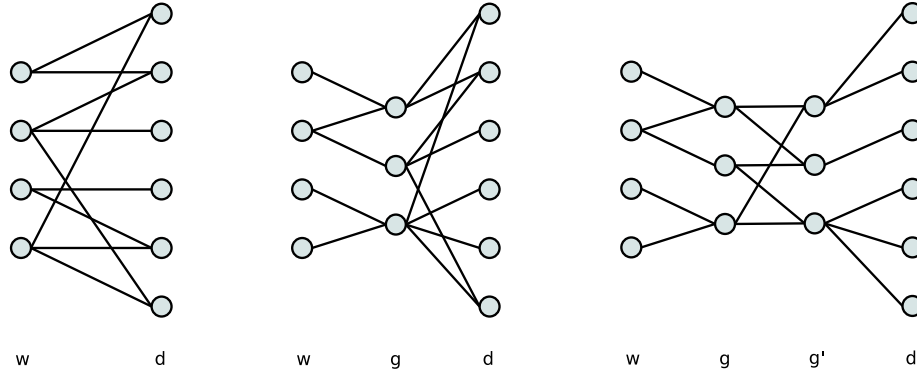
is still a valid probability distribution; namely, that $p'(*, *) = 1$.

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

3.4.1 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 rough approximations for the log likelihood $-\log p$, and so the contents of ‘words.n.2.s’ is effectively a representation of the matrix $p(w, g)$

Figure 1: Factorization



This figure attempts to illustrate the process of factorization. The left-most image is meant to illustrate $p(w, d)$ as a sparse matrix. 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.

for $g = \langle \text{plural-count-nouns} \rangle$ 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 $g' = \langle \text{noun-main-p} \rangle$. It appears multiple times throughout the file. In one case, it is associated with the word-list ‘words.n.2.s’, which makes sense, as ‘<noun-main-p>’ is describing one of the linkage behaviors of plural common nouns. The contents of the file ‘4.0.dict’ should be understood to be a specification of $M(g, g')$, although it is not so cleanly organized: it also includes all of the “macros” $p(g', d)$ and also includes word-lists when these are small.

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 factored into smaller, tightly-connected blocks; *viz.*, that the graph contains bipartite cliques.

3.5 Gradient Descent vs. Graph Algorithms

The goal of the matrix factorization, illustrated in figure 1, is to not only obtain the three factors $p(w, g)$, $M(g, g')$ and $p(g', d)$, but to obtain them such that the first and last components contain essentially no bipartite cliques, and all of the structural complexity is limited to $M(g, g')$. A naive gradient descent does not seem to achieve this. The situation is analogous to that discussed in section 2: naive gradient descent, such as SkipGram/CBOW, do not provide the needed grammatical graph structure. By contrast, the MST parsing algorithm does provide that structure.

The gradient descent methods do provide one advantage: they provide a means of global optimization; by contrast, graph techniques, such as MST, work to identify local graph structure. The goal of this section is to use this insight, and provide a graph-based technique to obtain the factors $p(w, g)$, $M(g, g')$ and $p(g', d)$, as graphs, rather than matrices. 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.

TODO: work all this out in detail. This section, and the rest, are all under construction.

3.6 Information Maximizing Matrix Factorization

To guide the factorization, and to maximize the agreement between the left and the right sides, one can appeal to Tegmark’s formulation of Tononi integrated information. That is, one wishes to factorize in such a way that the total amount of information in the factors is maximized, so that the “cut” is done where the mutual information is weakest.

xxx

xxxx

, although it requires some background theory.

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

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

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

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

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

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

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

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

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

is suggests that the

There is a set of word-classes $C = \{c\}$ and two projection matrices π^W and π^D such that $\vec{\eta}_w = \pi^W \hat{e}_w$ is a vector that classifies the word w into one or more word-classes c . That is, $\vec{\eta}_w$ is a C -dimensional vector. In many cases, all but one of the entries in $\vec{\eta}_w$ will be zero: we expect the word w = the to belong to only one class, the class of determiners. By contrast, w = saw has to belong to at least three classes: the past-tense of the verb “to see”, the

noun for the cutting tool, and the verb approximately synonymous to the verb for cutting. The hand-built dictionary for English Link Grammar has over a thousand distinct word-classes; one might expect a similar quantity from an unsupervised algorithm.

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

Gluing axioms

The language-learning task requires one to infer the structure of language from a small number of instances and examples. Bengio *et al.* [3] describe this for continuous probabilistic models. First, one imagines some continuous, uniform space. Example sentences form a training corpus are associated with single points in this space: the probability mass is initially located at a collection of points. One then imagines that generalization consists of smearing out those points over an extended volume, thereby assigning non-zero probability weights to other “nearby” sentences. This suggests that there is a choice as to how

this smearing-out is done: one can spread the probabilities uniformly, in all “directions”, or one can preferentially spread probabilities only along certain directions. Bengio suggests that higher-quality learning and generalization can be achieved by finding and appropriately non-uniform way of smearing the probability masses from training.

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

References

- [1] Daniel Sleator and Davy Temperley., *Parsing English with a Link Grammar*, Tech. rep., Carnegie Mellon University Computer Science technical report CMU-CS-91-196, 1991, URL <http://arxiv.org/pdf/cmp-lg/9508004>.
- [2] Daniel D. Sleator and Davy Temperley, “Parsing English with a Link Grammar”, in *Proc. Third International Workshop on Parsing Technologies*, 1993, pp. 277–292, URL <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/link/pub/www/papers/ps/LG-IWPT93.ps>.
- [3] Y. Bengio, et al., “A neural probabilistic language model.”, *Journal of Machine Learning Research*, 3, 2003, pp. 1137–1155.
- [4] Tomas Mikolov, et al., “Efficient Estimation of Word Representations in Vector Space”, *arXiv/13013781v3*, 2013.
- [5] Alex Minnaar, “Word2Vec Tutorial Part II: The Continuous Bag-of-Words Model”, , 2015, URL http://mccormickml.com/assets/word2vec/Alex_Minnaar_Word2Vec_Tutorial_Part_II_The_Continuous_Bag_of_Words_Model.pdf.
- [6] Gintare Karolina Dziugaite and Daniel M. Roy, “Neural Network Matrix Factorization”, , 2015, URL <https://arxiv.org/abs/1511.06443>, arXiv abs/1511.06443.

- [7] Joonseok Lee, et al., “LLORMA: Local Low-Rank Matrix Approximation”, *Journal of Machine Learning Research*, 16, 2016, pp. 1–24.
- [8] “Kernel smoother”, [http : //en.wikipedia.org/wiki/Kernel_smoother](http://en.wikipedia.org/wiki/Kernel_smoother).
- [9] “Local regression”, [http : //en.wikipedia.org/wiki/Local_regression](http://en.wikipedia.org/wiki/Local_regression).