New Distributed Probabilistic Language Models

Yoshua Bengio
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal
Montréal, Québec, Canada, H3C 3J7
Yoshua.Bengio@umontreal.ca

Technical Report 1215, Dept. IRO, July 31st, 2002

Abstract

Our previous work on statistical language modeling introduced the use of probabilistic feedforward neural networks with shared parameters in order to help dealing with the **curse of dimensionality**. This work started with the motivation to speed up the above model and to take advantage of prior knowledge e.g., in WordNet or in syntactically labeled data sets, and to better deal with polysemy. With the objective of reaching these goals, we present here a series of new statistical language models, most of which are yet untested.

1 Introduction

In our previous work (Bengio, Ducharme and Vincent, 2002), we have shown that the probability of word sequences can be efficiently represented by various artificial neural network architectures, where *efficiently* refers here to the statistical sense, i.e. these models generalize well and have low perplexity. However they are computationally much more expensive than *n-grams* (Katz, 1987; Jelinek and Mercer, 1980), both for training and for probability computation. The basic ideas were the following: a neural network represents $P(w_t|w_{t-1}, \ldots w_{t-n})$, using an intermediate distributed representation for each word in the vocabulary (e.g. as a learned real *feature vector* in $\mathbb{R}^{30}$). The probability function depends smoothly on the feature vectors of the observed words, thus when one (or more) word is replaced by another with a close feature vector, the output probability does not change much. This is obtained without clustering, rather by automatically learning a notion of similarity between words, simply through maximum likelihood training of the probability function (by stochastic gradient ascent). In principle, this type of representation (which is **distributed** rather than **local**) also opens the door to learning dependencies between many high-dimensional discrete objects. For example, the neural network can easily be extended beyond the usual 2 words of contexts (up to 8 in our experiments), without the ensuing overfitting typically observed with n-grams.

In this technical report, we reflect on the above model and recent progress in statistical language modeling, speculate on the ingredients needed to make significant progress in statistical language modeling, and propose a few new models towards those goals.

2 Discussions and Speculations

The first part of this report analyzes some previous results and speculates on what is needed for progress in statistical language modeling.

2.1 High Capacity Needed

It is now quite clear for most researchers in the machine learning and statistical language modeling communities that a *very large capacity* is required to build high-performance language models, and this clearly requires *lots of data* (millions to billions of examples).

Some clear evidence towards that conclusion is the solid success of *n-grams* and *lexicalized stochastic grammars* (Charniak, 1999; Collins, 1997; Collins, 1999; Chelba and Jelinek, 2000; Ratnaparkhi, 1999) compared to non-lexicalized (stochastic or not) grammars, and other *data-poor low-capacity* (earlier) approaches.

Unfortunately, most of the research in statistical machine learning (e.g. see the NIPS proceedings) of the last decade (including much of the author's own research) has been focussing on small data sets (up to a few thousand examples), and on algorithms (e.g. for model selection) that tend to make a significant difference when the number of examples is small.

For statistical machine learning to continue making a strong impact on the field of language modeling, it is important to develop algorithms suited to learning large and complex models on very large data sets. For example, whereas overfitting is often the issue on small data sets, underfitting can be a major obstacle for large models trained on large data sets. This is not to say that the old problem of the curse of dimensionality should be forgotten, especially for non-parametric approaches. Indeed language is intrinsically high-dimensional, albeit in a way quite different from that usually considered in many analyses of high-dimensional data, because of its discrete nature.

2.2 Curse of Dimensionality, Distributed Representation

Associating probabilities (or other numbers) with all the possible combinations of words (or other high-dimensional discrete objects) in one (or more) sentence obviously *blows up exponentially with the number of words*, apparently unless drastic conditional independence are assumed (which prevent us from capturing some dependencies that we know to matter!). This well-known problem plagues n-grams (Katz, 1987; Jelinek and Mercer, 1980; Goodman, 2001), lexicalized stochastic grammars (Charniak, 1999; Collins, 1997; Collins, 1999; Chelba and Jelinek, 2000; Ratnaparkhi, 1999), and many other probabilistic models (e.g. graphical models with dense cycles, in general, see (Jordan, 1998)), and it is well described, with many examples and analyses, in the book (Manning and Schutze, 1999). The usual solution is to combine the model with simpler models, e.g. by deleted interpolation or by backing-off to the simpler models according to some threshold rules. This helps making sure that no observation sequence gets a tiny probability, but this is a achieved at a high price, by distributing probability mass over vast volumes of data space.

2.3 Distributed Representations for High-Order Dependencies

The idea of using **distributed representations** has been one of the early contributions of the *connectionist* researchers of the 80's, e.g. see (Hinton, 1986). The hidden units of an artificial neural network encode information in a very efficient way (e.g. n binary neurons can represent 2^n different objects). More importantly, these hidden units can **capture the high-order dependencies that matter most**. Consider for example m binary inputs to a neural network. There are 2^m possible combinations of these inputs, and a linear / polynomial model would require $O(2^m)$ free parameters to capture dependencies involving all the variables (e.g. this can be done by providing to a linear model additional input features that are indicators associated with any combination of the variables). In general, to capture all dependencies of order d (e.g. between d-tuples) would require $O(m^d)$ parameters. This is a particular case of the curse of dimensionality. Things are of course worse with language, in which the "input variables" are not binary but high-dimensional objects (e.g. words chosen in a vocabulary of size 20,000). Here the advantage of artificial neural networks is that a small number of N hidden units can capture the N most important high-order dependencies between the inputs. The number of free parameters is only $O(Nm)$, yet dependencies of any order can be captured. To better see how high-order dependencies can be captured, consider the function computed by a single hidden unit of the form $\tanh(\sum_{j=1}^{n} w_{ij} x_j)$ with parameters w_i and input vector x. A Taylor expansion of this function in x shows that it depends on all the products of elements of the vector x, albeit with a very strong constraint since there are only n free parameters. Note that the output of this hidden unit will be "high" for a whole half-space of $\mathbb{R}^n$, and if x is a binary vector (all the data points are corners of a hypercube), it means that the hidden unit can isolate any one particular combination of values of the x_i's, or a larger subset if the half-plane is moved toward the middle of the hypercube.

The drawback is that the estimation of the parameters is much more difficult (the objective function is not convex and can in fact be quite complex) and requires more computation. This idea has been exploited to learn the probability function of high-dimensional discrete data in (Bengio and Bengio, 2000b; Bengio and Bengio, 2000a), and comparisons have been made with polynomial learners.

Note that this idea of learning to pay attention only to the most relevant combinations of the variable values is exploited explicitly in so-called Maximum Entropy (Berger, Della Pietra and Della Pietra, 1996) language models, which are of the form

$$P(Y = y|X = x) = \frac{e^{\sum_i \theta_i f_i(y,x)}}{\sum_{y'} e^{\sum_i \theta_i f_i(y',x)}}. \quad (1)$$

Indeed, the application of these models typically involves a **feature selection** learning algorithm, which greedily selects a subset of "features" f_i out of a very large enumerable "basic set" obtained by considering many combinations of values of the discrete variables of interest. However, this feature selection quickly becomes unwieldy and search heuristics (that are computationally feasible) are possibly myopic when the number of variables to consider increases (because the “basic set” quickly becomes huge). Typically, to simplify the search, the features discovered for Maximum Entropy models are binary and involve a small subset of the variables, whereas the features discovered in the hidden units of a neural network typically involve most of the variables (although at different degrees, since the output of the hidden unit is not binary).

There has also been a lot of interest in recent years in developping inference algorithms for generative (causal) **graphical models** (Jordan, 1998) with distributed representations (i.e. there are many hidden variables whose joint space is exponentially richer than for example the hidden variables found in hidden Markov models). Unfortunately, interesting graphical models with distributed representations usually yield to intractable exact inference. Many approaches to approximate inference have been proposed in recent years, which offer hope to learn and perform inference on such models. Two particulary interesting avenues are the variational approximations (Jordan et al., 1999; Jaakkola and Jordan, 1999) and the variants of loopy belief propagation (Weiss, 2000).

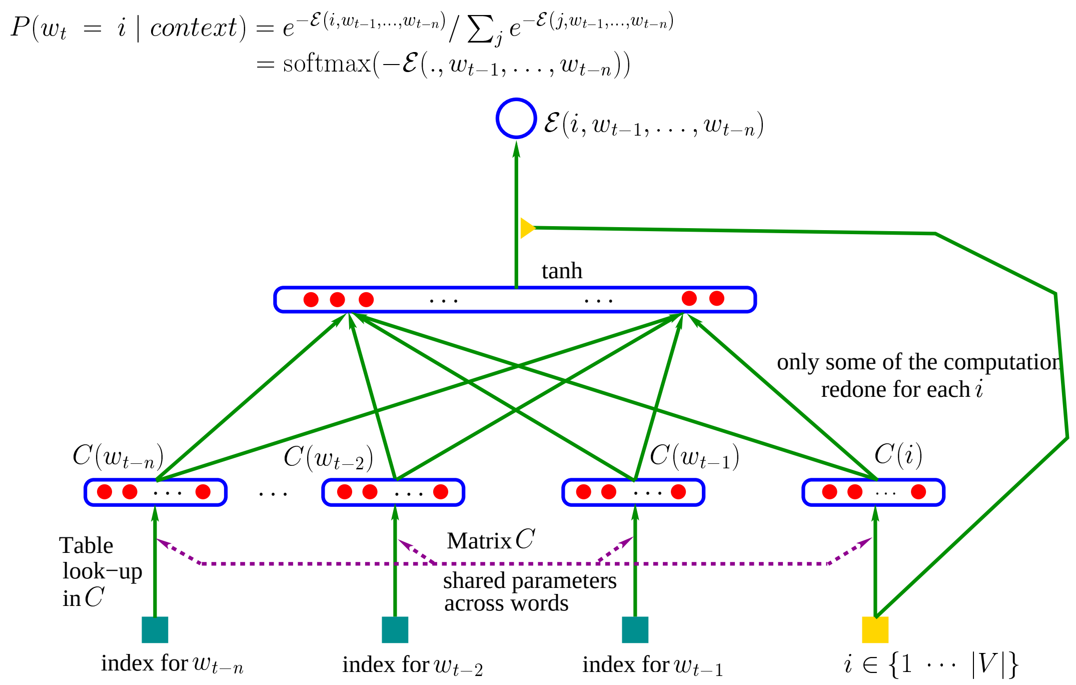
In terms of representation (i.e., once they are learned), distributed representations (e.g. in neural networks, distributed representation graphical models, or in Maximum Entropy models) can be exponentially more efficient then “localist” representations, allowing to consider many higher-order dependencies. However, they are more difficult to optimize (e.g. analytic or quickly converging algorithms like EM are not applicable), and they may involve expensive computations, like the estimation of the partition function (normalizing constant) in Maximum Entropy models.

2.4 Neural Architecture for Representing High-Dimensional Distributions

Neural networks have been studied for language modeling for many years already (Miikkulainen and Dyer, 1991), but it is only recently that large scale experiments have demonstrated quantitatively their advantage in statistical language modeling. This has been achieved with the neural network already described in (Bengio, Ducharme and Vincent, 2002), which has the basic architecture shown in Figure 1. This is related to previous work using neural networks to model sequences of characters (Schmidhuber, 1996), and the use of distributed representations to learn symbolic relations (Paccanaro and Hinton, 2000). The advantage of distributed representations in statistical language modeling has also been demonstrated clearly with the successes of eigen-decomposition methods (Schutze, 1993) for information retrieval (Deerwester et al., 1990) (Latent Semantic Indexing). The idea of a vector-space representation for symbols in the context of neural networks has also previously been framed in terms of a parameter sharing layer, e.g. in (Riis and Krogh, 1996) for secondary structure prediction and in (Jensen and Riis, 2000) for text-to-speech mapping.

The output of the neural network depends on the next word w_t and the previous words $h_t = (w_{t-1}, w_{t-2}, \ldots, w_{t-n})$ as follows. In the *features layer*, one maps the words to a lower-dimensional continuous subspace:

$$\begin{aligned} z_i &= C_{w_{t-i}}, \quad i \in \{0, 1, \ldots, n\}, \\ z &= (z_0, z_1, \ldots, z_n) \end{aligned}$$



$P(w_t = i \mid context) = e^{-\mathcal{E}(i,w_{t-1},\ldots,w_{t-n})} / \sum_j e^{-\mathcal{E}(j,w_{t-1},\ldots,w_{t-n})}$
$= \text{softmax}(-\mathcal{E}(.,w_{t-1},\ldots,w_{t-n}))$
$\mathcal{E}(i,w_{t-1},\ldots,w_{t-n})$
tanh
only some of the computation redone for each i
$C(w_{t-n})$
$C(w_{t-2})$
$C(w_{t-1})$
$C(i)$
Table look−up in C
Matrix C
shared parameters across words
index for w_{t-n}
index for w_{t-2}
index for w_{t-1}
$i \in \{1 \cdots |V|\}$

Figure 1: It currently takes days to train and run on a parallel machine, but this architecture can yield perplexity improvements 20%-30% wrt interpolated n-grams. Best results are obtained when the output weight vector is conditionned on the next word i.

where C_j is the j-th column of the **word features** matrix of free parameters, and z is the input vector for the next layer, the hidden layer:

$$a = \tanh(d + Wz) \tag{2}$$

where d is a vector of free parameters (hidden units biases), W is a matrix of free parameters and a is a vector of hidden units activations obtained by applying the hyperbolic tangent function (a scaled and shifted sigmoid) element by element to the $d + Wz$ vector. Finally the output is a scalar energy function

$$\mathcal{E}(w_t, h_t) = b_{w_t} + U_{w_t}.z + V_{w_t}.a$$

where b is a vector of free parameters (called biases), and U and V are matrices of free parameters with one column U_i or V_i per word (note that U or V can be zero in variants of this model).

To obtain conditional probabilities, we normalize the exponentiated energies:

$$P(w_t|h_t) = \frac{e^{-\mathcal{E}(w_t,h_t)}}{\sum_i e^{-\mathcal{E}(i,h_t)}}.$$

The above neural architecture can be viewed as a generalization of Maximum Entropy models (see eq. 1), in which the features f_i are not discrete and they are learned at the same time as the weights θ_i to maximize a penalized log-likelihood criterion. Unfortunately, this prevents the use of computational tricks which can be used to speed-up parameter estimation with Maximum Entropy models.

The above neural architecture can also be viewed as a special case of *energy-based probability models* (advocated by Hinton (**?**; **?**; Hinton, 2002)), of the form

$$P(Y = y) = \frac{e^{-\mathcal{E}(y)}}{\sum_{y'} e^{-\mathcal{E}(y')}}$$

or, for conditional probabilities

$$P(Y = y|X = x) = \frac{e^{-\mathcal{E}(y,x)}}{\sum_{y'} e^{-\mathcal{E}(y',x)}}$$

where $\mathcal{E}(.)$ is a parametrized function which is low for plausible configurations of y (or (x, y)), and high for improbable ones. The difficulty with these energy models is in learning the parameters of the energy function, without an explicit computation of the partition function (normalizing denominator). Hinton has proposed the **contrastive divergence** (Hinton, 2002) approach to efficiently approximate the gradient of the log-likelihood with respect to these parameters, in the unsupervised (unconditional) case (where the sum involved in computing the partition function has an exponential number of terms).

Contrastive Divergence is based on a sampling approximation of the log-likelihood gradient, $\frac{\partial \log P(Y=y)}{\partial \theta}$. More generally, the gradient can be decomposed in **two parts**: *positive reinforcement for* $Y = y$ (we want to increase $P(Y = y)$ and *negative reinforcement for every* y' (we want to decrease $P(Y = y')$ weighted by $P(Y = y')$):

$$\frac{\partial \log P(y)}{\partial \theta} = -\frac{\partial \mathcal{E}(y)}{\partial \theta} + \sum_{y'} P(y') \frac{\partial \mathcal{E}(y')}{\partial \theta} \tag{3}$$

The idea of contrastive divergence is to approximate the average in the right-hand side above by a sample from a single step of a Monte-Carlo Markov Chain (e.g. Gibbs sampling, for the Products of Experts architecture) starting at y and that would converge to the stationary intractable distribution $P(y')$.

In this paper, we will propose several new variants of this idea for the case when P is our conditional or unconditional language model and a proposal distribution is available (e.g. an n-gram model), from which sampling is easy.

2.5 Products vs Sums of Probabilities

Like previous probabilistic models (e.g. weighted averages in the log-domain (Genest and Zideck, 1986; Heskes, 1998)), the Maximum Entropy model (Berger, Della Pietra and Della Pietra, 1996) and the neural models described above, can be interpreted as energy-based models that correspond to **normalized products**, as in Hinton's *Products of Experts* (Hinton, 2002) model:

$$P(y) = \frac{\prod_i P_i(y)}{Z} \tag{4}$$

where $P_i(y)$ are the individual expert models, and Z is a global normalizing constant (the **partition function**). Such a form is also found in Maximum entropy statistical language models (Berger, Della Pietra and Della Pietra, 1996), in which the individual "expert" has the form

$$P_i(y|x) = e^{\theta_i f_i(y,x)}/Z_i.$$

In the above neural network, instead, we have "experts" associated with hidden units (or word feature values), i.e. of the form

$$P_i(w|h) = e^{U_{iw} z_i(w,h)}$$

or

$$P_i(w|h) = e^{V_{iw} a_i(w,h)}$$

or

$$P_i(w|h) = e^{b_w}$$

(the latter being essentially "unigram" experts).

This type of model can be contrasted with *mixture models* such as HMMs and many other probabilistic (EM-trained) models. Comparisons between mixtures of experts and products of experts have been carried, and they suggest that products of experts can yield to significant improvements in terms of out-of-sample likelihood. For example, Foster's experiments (Foster, 2002) confront head-to-head a normalized product of probabilites (implemented by a Maximum Entropy model) with a weighted sum of probabilities. In this case the application is to statistical translation models and one would like to estimate

$$P(next\ target\ word|previous\ target\ words, source\ sentence)$$

in the process of building

$$P(target\ sentence|source\ sentence).$$

With an additive approach, one builds a mixture of $P(next\ target\ word|previous\ target\ words)$ and $P(next\ target\ word|source\ sentence)$. This corresponds more to a kind of **disjunction**: the probability of the mixture is "not low" if one **or** the other component is "not low". With a multiplicative approach (much more computationally expensive because of the normalization problem), the two probabilities are multiplied and then normalized. This corresponds more to a kind of **conjonction**: the probability of the product is "not low" only if both one **and** the other component are "not low". The results of this experiment are extremely strong, with a **reduction of perplexity by a factor of 2** using the normalized product (Foster, 2002).

Another suggestive set of comparative results is provided by Charkniak's experiments (Charniak, 1999) with stochastic grammars. Many comparisons are performed with different variants of a lexicalized stochastic grammar. One of the comparisons involves the choice between a deleted interpolation scheme (which is a mixture) and a Maximum Entropy scheme (which is a product) in order to combine several sources of information (conditioning events to predict the rule probability). The performance measure is the average performance/recall of syntactic structure, starting from an error rate of about 12%. Charniak finds that whereas the Maximum Entropy scheme yields an absolute improvement of 0.45%, the deleted interpolation in this case worsens performance by 0.6%, an overall difference of more than 1%, which is quite a lot near the 12% error mark.

Why do we observe such improvements? we conjecture that the reason is that in high-dimensional spaces, a mixture is generally too wasteful of probability mass. In the translation example, it is clear that a conjunction is more appropriate than a disjunction: we want to accept a translated word if it is **both** consistent with the previously translated words **and** with the source sentence. In general, the product gives rise to a sharper likelihood, whereas a mixture **can only dilute probability mass**. If each expert only has part of the information (e.g. looking only at certain aspects or certain variables), then it makes more sense to use a product. The experts in the product can be seen as **constraints to be satisfied**. This argument has been made by Hinton (Hinton, 2002).

2.6 How Do N-grams Generalize?

By *N-grams* this title means any one of the variants of deleted interpolation or back-off models that combine multiple models with a simple form $P(w_t|w_{t-1}, \ldots w_{t-n})$ and the different values of n up to a maximum N.

A useful way to view these models is simply as **non-parametric** probabilistic models that are restricted to depend only on the frequency statistics of tuples of length up to $N+1$ (that's the N in N-gram). As such one can get an intuition for what they are representing and how they generalize by thinking about them as **generative models**. How would one sample a sentence from an **N-gram** model? How to N-grams generalize from training sentences to new sentence?

A new sentence can be generated by "pasting" overlapping subsequences of length 1, 2... $N+1$ taken from the training set, as shown in 2.

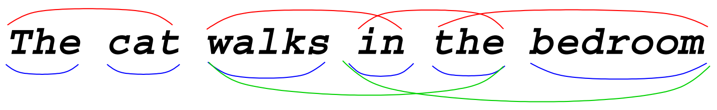


Figure 2: An illustration of how trigrams generalize and how they can generate a new sentence by pasting overlapping subsequences seen frequently in the training set.

The mode of generalization used by N-grams is very powerful, but there are certainly other regularities in language that one should be able to take advantage of in building statistical language model.

2.7 Generalizing Through Word Similarity

A very obvious way to generalize should be to take advantage of the fact that one can replace a word by another, similar word, and preserve much of the meaning, or at least preserve the linguistic plausibility of the sentence.

For example, we would like to generalize from a sentence like

The cat walks in *the bedroom*

to *A dog* runs in *a room*

and likewise to *The cat* runs in *the room*

A cat runs in *a bedroom*

The dog walks in *the bedroom*

...

and a myriad of other combinations!

This could be achieved if we knew that dog and cat played similar roles (semantically and syntactically), and similarly for (the,a),(bedroom,room), (runs,walks).

Note that simply merging 'similar' words in a single category (clustering) would go towards that objective but it might lose a lot of useful information. A smooth notion of similarity might be better, and the neural network described in the previous section was designed taking that goal into account. The idea is the following: if two words have similar word features (i.e. $||C_i - C_j||$ is small), then the energy function would be almost unchanged when replacing input word i by word j, by virtue of the smoothness of the mapping from word features (z in the above equations) to energy $\mathcal{E}$.

2.8 Generalization by Spreading Probability Mass

For learning probability models, generalization amounts to deciding how to spread probability mass from the training instances to the rest of the data domain.

Let us try to list the main modes of generalization, i.e. the main ways to spread probability mass, that have been found useful (or that we expect to be useful) in statistical language models:

1. Simple smoothing by interpolation / back-off to simpler models. This includes smoothing schemes to deal with never seen words and very low-frequency words, by spreading some probability mass to them. To better understand what all these smoothing techniques essentially do, consider the simple case of interpolating or backing-off from a bigram to a unigram. The bigram is is large $M \times M$ table with an entry for each pair of words. However, many entries have zero empirical frequency. These smoothing methods basically take some mass from the observed case at position (i, j) and distribute it to a whole row. In high-dimensional spaces, these techniques amount to spreading probability mass uniformly in some directions, which is quite wasteful but guarantees that no entry will end up having a zero probability.

2. Focussing on short-term context (e.g. N-grams), which amounts to giving probability mass to new sentences by by pasting short frequent and overlapping pieces from the training data. The intuition behind this idea has been explained previously in section 2.6.

3. Substituting words with replaceable ones. This includes clustering schemes as well as the word features trick used in the above neural network. This idea has been explained previously in section 2.7.

4. Grammatical substitution of phrases with replaceable ones. This is what stochastic grammars are after. See more discussion on stochastic grammars in section 3.7.

5. Generalizing from meaningful sentences to other meaningful sentences. This is much harder... but raises the really interesting question of capturing the **joint** statistics of the "senses" associated with each meaningful word in a sentence. This is a much higher-dimensional space (e.g. the space of meanings for sentences), that machine learning research should focus on.

2.9 Is Unsupervised Learning from Text Sufficient?

We conjecture that learning about semantics using only text is extremely inefficient, and that much better results would be obtained by using data that associates text with other modalities.

An extreme argument, suggested by Lyle Ungar[1], is the following. Consider an impoverished language with a very large number of "word-utterances", but where each word represents a whole semantic unit of information (e.g., like a sentence in English), that can stand on its own independently of the previous or of the next. Since there is very little temporal structure in a sequence of such "word-utterances", the only reliable statistics that one could capture from such data are the frequencies of each utterance. Clearly, it would thus be impossible to capture most of the meaning behind each of the "word-utterances".

Obviously, human languages have more structure, which gives us hope to capture quite a bit of semantic information just from text (and we are already able to do that, e.g. consider unsupervised word clustering). However, there might be some semantic elements that are either impossible to capture or would require much more data than if we could associate text with other modalities, if this additional data X is dependent on the text W_1^T through the $Meaning$ of the text:

$$W_1^T \leftrightarrow Meaning \leftrightarrow X$$

Where could we get such multi-modal data? to have any chance to capture such structure, we probably need very large amounts of such paired data, and good models of the other modality X. Here are some currently plausible sources of large amounts of paired data:

- bilingual texts (lots are available!)
- image databases (with labels or comments on the images)
- videos
- web pages

The last three require a substantial modeling effort for the "other modality", bilingual texts are easier to deal with (same type of data as a single text) but probably not as "powerful" in revealing semantics as the last three.

Our statistical models would then have to be flexible enough to capture information from multiple sources and multiple modalities: raw text, WordNet (Fellbaum, 1998), bilingual texts, text associated with other modalities, all tied through *common internal representations of meaning* (not just the meaning of individual words, but also of sequences of words).

[1]personal communication, April 2002.

3 Current and Future Work

3.1 Sampling Approximations of the Log-Likelihood Gradient

When the vocabulary size is M, the full computation of the log-likelihood gradient essentially involves M forward (calculating $\mathcal{E}$) and backward passes (calculating $\frac{\partial \mathcal{E}}{\partial \theta}$) through the neural network (but there are some computations that do not need to be re-done: the part of Wz that depends only on h_t, see equation 2).

Again, let $h_t = (w_{t-1}, \ldots, w_{t-n})$. With our "energy-based" models with output $\mathcal{E}$,

$$P(w_t|h_t) = \frac{e^{-\mathcal{E}(w_t,h_t)}}{\sum_{w'} e^{-\mathcal{E}(w',h_t)}}$$

and the gradient has *two parts*: positive reinforcement for w_t and negative reinforcement for every word w', weighted by $P(w'|h_t)$:

$$\frac{\partial \log P(w_t|h_t)}{\partial \theta} = -\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta} + \sum_{w'} P(w'|h_t) \frac{\partial \mathcal{E}(w',h_t)}{\partial \theta} \qquad (5)$$

Because of the large number of words M in the vocabulary, there is clearly a huge potential for speeding up by replacing the above weighted average by Monte-Carlo samples. Since we are basically training these models by stochastic (or mini-batch) gradient descent, we don't need an exact estimation of the gradient. An unbiased estimator (as long as it is not too noisy) should converge about as fast (noting that the instantaenous gradient is itself a noisy unbiased estimator of the "complete" gradient, which is an average across all the data points).

The basic idea of the procedures that we are exploring here is summarized in Algorithm 1, for each training example (w_t, h_t).

Algorithm 1 Monte-Carlo approximation of gradient

```
(1) add pos. contribution: $\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta}$
(2) repeat $N$ times
      sample neg. example $w' \sim P(w'|h_t)$,
      add its neg. contribution: $-\frac{1}{N}\frac{\partial \mathcal{E}(w',h_t)}{\partial \theta}$
(3) update $\theta$ from sum of above contributions
```

The potential speed-up of this procedure is M/N (if there is no reduction in convergence speed due to a noisier gradient). Since M is typically several tens of thousands, this is appealing. Unfortunately, we can't actually perform a (cheap) ordinary Monte-Carlo approximation of $\sum_{w'} P(w'|h_t) \frac{\partial \mathcal{E}(w',h_t)}{\partial \theta}$ (step (2) above) because we don't know how to cheaply sample from $P(w'|h_t)$ (doing it properly would require as much work as doing the full sum itself).

Fortunately, in the case of language modeling, we have the opportunity to take advantage of approximations Q of P, from which it is easy (cheap) to sample, e.g. those provided by a unigram or by n-grams in general. In particular, such an approximation can be used as a **proposal distribution** in a Monte-Carlo sampling algorithm. Several sampling algorithms exist that can take advantage of a proposal distribution.

3.1.1 Independent Metropolis-Hastings

A Monte-Carlo Markov Chain (MCMC) converging to $w' \sim P(w'|h)$ as $N \to \infty$ can be obtained using the Independent Metropolis-Hastings method, shown applied to our case in Algorithm 2.

Algorithm 2 Independent Metropolis-Hastings Gradient Approximation

```
(1) add pos. contribution: $\frac{\partial \mathcal{E}(w_t, h_t)}{\partial \theta}$
(2) Initialize $w' = w_t$
(3) for (k=0;k<$N_1 + N_2$;)
   Sample $j$ from $Q(j|h_t)$
   Let $r = \min(1, \frac{e^{-\mathcal{E}(j,h_t)}}{e^{-\mathcal{E}(w_t,h_t)}} \frac{Q(w_t|h_t)}{Q(j|h_t)})$
   with probability $r$
     k++; $w' = j$;
     if k$\geq N_1$:
       add neg. contribution: $-\frac{1}{N_2} \frac{\partial \mathcal{E}(w', h_t)}{\partial \theta}$
```

The first N_1 samples are used to get the Markov chain to converge close enough to P, and the last N_2 samples are used to form the gradient average. Note that because this is an MCMC, we obtain an unbiased sample only as $N_1 \to \infty$.

When we choose N_1 small (e.g. 0 or 1), we don't wait at all for the convergence of the MCMC. This is following the intuition of Hinton's *contrastive divergence* (which is based on Gibbs sampling), that a good approximation of the gradient can be obtained after only one step of the chain, if we start the chain at the observed input (w_t). However, because of the nature of "word space" and its simple-minded representation here, it is not clear that the above sampling scheme has a strong preference for "neighboring" words. If on the other hand we were sampling words based on their continuous representation (in the word features $C(i)$), such a scheme might work. But how to efficiently obtain such a sample is not yet clear.

3.1.2 Importance Sampling Approximation

Importance sampling is a Monte-Carlo scheme which does not involve a Markov Chain, i.e. every sample is an unbiased estimator. It is used when one cannot sample from P but has an approximation Q such that $P > 0 \to Q > 0$. Whereas the ordinary Monte-Carlo estimator of $\int g(y)p(y)dy$ is

$$\frac{1}{N} \sum_{y_i \sim p} g(y_i)$$

with N samples $\{y_i\}$ generated with probability $p(y_i)$, the corresponding importance sampling estimator is

$$\frac{1}{N} \sum_{y_i \sim q} g(y_i) \frac{p(y_i)}{q(y_i)}.$$

with N samples $\{y_i\}$ generated with probability $q(y_i)$. It is easy to show that the above is an unbiased estimator of the integral. Strictly speaking, we cannot use ordinary unbiased importance sampling since we don't want to compute $p(y_i)$ exactly (only up to the normalization constant Z). However, we can use an approximate scheme in which Z is also estimated. Compared to the Metropolis method, this has the clear advantage of not being an MCMC, i.e. we don't need to worry about convergence of the Markov chain.

The partition function can be estimated using importance sampling with Q the proposal distribution instead of the uniform over the M words:

$$Z(h_t) = \sum_{w'} e^{-\mathcal{E}(w',h_t)} = M \sum_{w'} (1/M) e^{-\mathcal{E}(w',h_t)}$$

so the estimator is

$$\begin{aligned} \approx \hat{Z}(h_t) &= \frac{M}{N} \sum_{w' \sim Q(w'|h_t)} \frac{e^{-\mathcal{E}(w',h_t)}}{MQ(w'|h_t)}, \\ \hat{Z}(h_t) &= \frac{1}{N} \sum_{w' \sim Q(w'|h_t)} \frac{e^{-\mathcal{E}(w',h_t)}}{Q(w'|h_t)} \end{aligned} \tag{6}$$

Using this estimator, we can apply (biased) importance sampling to the average gradient estimation:

$$\sum_{w'} P(w'|h_t) \frac{\partial \mathcal{E}(w',h_t)}{\partial \theta} \approx \frac{1}{N} \sum_{w' \sim Q(w'|h_t)} \frac{e^{-\mathcal{E}(w',h_t)}}{Q(w'|h_t)\hat{Z}(h_t)} \frac{\partial \mathcal{E}(w',h_t)}{\partial \theta}$$

so the overall gradient estimator for example (w_t, h_t), using the set $\mathcal{W}$ of N samples from $Q(.|h_t)$:

$$\frac{\partial \mathcal{E}(w,h_t)}{\partial \theta} - \frac{\sum_{w' \in \mathcal{W}} \frac{e^{-\mathcal{E}(w',h_t)}}{Q(w'|h_t)} \frac{\partial -\mathcal{E}(w',h_t)}{\partial \theta}}{\sum_{w' \in \mathcal{W}} \frac{e^{-\mathcal{E}(w',h_t)}}{Q(w'|h_t)}}$$

Note that since $E[\frac{A}{B}] \neq \frac{E[A]}{E[B]}$, this is a biased estimator (the more so for small N). However, because we use the same samples in the numerator and the denominator (i.e. the same weights $\frac{e^{-\mathcal{E}(w',h_t)}}{Q(w'|h_t)}$), I conjecture that a better estimator is obtained: when the weights are "too large" in the numerator, they are also "too large" in the denominator, and this variation partially cancels out. Otherwise, another strategy would be to use uniform samples to approximate $\hat{Z}$, instead of the samples from Q as in equation 6.

The resulting procedure is summarized in Algorithm 3 below.

Preliminary experiments were performed by Jean-Sebastien Senecal using a version of Algorithm 3 that gradually increases N as training progresses, yielding very significant speed-ups (more than 15-fold). However, I think that much better results have yet to be obtained with sampling approximations.

3.1.3 Adapting the Proposal Distribution

Better results (i.e. requiring a smaller sampling size) might be obtained with a more appropriate proposal distribution. Surprisingly, the unigram worked better than the interpolated bigram or interpolated trigram as a proposal distribution in our experiments.

Algorithm 3 Importance Sampling Gradient Approximation

```
(1) add pos. contribution: $\frac{\partial \mathcal{E}(w_t, h_t)}{\partial \theta}$
(2) vector num=0; denom=0
(3) repeat $N$ times
     Sample $w'$ from $Q(w'|h_t)$
     Let $r = \frac{\mathcal{E}(w', h_t)}{Q(w'|h_t)}$
     denom += $r$
     num += $r \frac{\partial \mathcal{E}(w', h_t)}{\partial \theta}$
(4) add neg. contribution: $-\frac{num}{denom}$
```

One way to make our proposal distribution Q better would be to make it a better approximation of P. A possibly even better approach is to strive towards the minimum variance estimator. When integrating the scalar function $g(y)$ under $p(y)$, the minimum variance proposal distribution for importance sampling is

$$\frac{|g(y)|p(y)}{\int |g(y)|p(y)dy}.$$

Note one actually gets zero variance with the above distribution, if $g \geq 0$. However, we are integrating a vector and we want to use a common proposal distribution for all the elements of the vector. In that case it is easy to show that the minimum variance proposal distribution is

$$\frac{||g(y)||p(y)}{\int ||g(y)||p(y)dy}.$$

where $||g(y)||$ is the L2 norm of vector $g(y)$. This is obtained by writing the variance of the importance sampling estimator, adding a Lagrangian constraint for $\int p(y)dy = 1$, and setting the derivative with respect to $p(y)$ to zero.

Let us therefore consider how we could approximate a minimum variance proposal distribution, using a bigram estimator $B(w_t|w_{t-1})$ that depends only on the previous word (the rest of the context is dropped).

Let $\mathcal{V} = \{\text{sampled negative examples at } t\} \cup \{w_t\} = \mathcal{W} \cup \{w_t\}$, where context $h_t = (w_{t-1}, w_{t-2}, \ldots)$. Let $\bar{b} = \sum_{i \in \mathcal{V}} B(i|w_{t-1})$ be the total B mass of those words previously and $\bar{y} = \sum_{i \in \mathcal{V}} e^{-\mathcal{E}(i,h_t)}$ their NN total unweighted probability. We propose the following *adaptive reweighting of the bigram probabilities* $B(i|w_{t-1})$ to track $P(w_t|w_{t-1})$, as follows:

$$\forall i \in \mathcal{V},\ B(i|w_t) \leftarrow \alpha B(i|w_t) + (1-\alpha)||\frac{\partial \mathcal{E}(i, h_t)}{\partial \theta}||\frac{e^{-\mathcal{E}(i,h_t)}\bar{b}}{\bar{y}}$$

The idea is to redistribute probability mass between the sampled words so that for B their relative probability within $\mathcal{V}$ agrees with the ideal proposal distribution, so at convergence we get

$$E[\frac{B(i|w_t)}{\bar{b}}] = E[\frac{e^{-\mathcal{E}(i,h_t)}}{\bar{y}}].$$

The reason for the moving average with "learning rate" α is twofold: first, the parameters of the neural network slowly move, so B has to track the optimal proposal distribution. Second, B can never perfectly approximate the optimal proposal distribution because it depends only on the previous word w_{t-1}, while in general h_t contains words before w_{t-1}. Therefore we have to average across the different contexts h_t which contain w_{t-1}. To keep B sparse, it is probably not worth it storing $B(i|j)$ for very rare (i,j) couples (e.g. those not appearing in the data, or those which have not been sampled early on). For these one could just back-off to the unigram as the proposal distribution. Otherwise, B could grow to the full bigram matrix, which does not fit in memory.

3.1.4 Joint Importance Sampling

A quite different type of sampling approximation, which does not fit into the picture of Algorithm 1, is based on **joint sampling of w and h**.

The main motivation is that with the conditional sampling of w given h, the unknown normalizing constant Z is really a function of h, since it involves normalization only across w:

$$Z(h) = \sum_{w'} e^{-\mathcal{E}(w',h)}$$

If instead we were sampling from both w and h, we would be refering to their joint distribution according to the energy function,

$$P(w,h) = \frac{e^{-\mathcal{E}(w,h)}}{\sum_{w',h'} e^{-\mathcal{E}(w',h')}}$$

when the partition function

$$Z = \sum_{w',h'} e^{-\mathcal{E}(w',h')}$$

independent of h. Therefore, **we can use not only the samples for current example (w,h) but also the samples for the previous ones in order to approximate Z.** If the approximation of the partition function was the limiting factor in Algorithm 1, or 3 then this strategy could be very useful.

However, one should note two possible disadvantages of this approach:

1. The above suggested procedure amounts to using the gradient

$$\frac{\partial \log P(w_t|h_t)}{\partial \theta} = -\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta} + \sum_{w',h'} P(w'|h') \frac{\partial \mathcal{E}(w',h')}{\partial \theta}$$

instead of the one in equation 5, which corresponds to maximizing the **pseudo-likelihood**

$$\sum_t \log P(w_t,h_t)$$

instead of the correct log-likelihood for a whole sequence,

$$\sum_t \log P(w_t|h_t).$$

Note how the probability of each word is "repeated" in the pseudo-likelihood. However, we conjecture that this is not a serious drawback.

2. The second and probably more serious problem is that the **resulting gradient estimator has more variance** because we are sampling from a much larger space. By analogy to discriminant learning algorithms for classification tasks, in which one tries to maximize the probability of the correct class and minimize the probability of the incorrect classes, we should find this joint sampling method to be **less discriminant**, i.e. we probably have to sample more in order to gather as much information about how and where to change the probability function. This is because we will sample in many places far from the observed (w_t, h_t) pair. A "discriminant" algorithm typically learns faster by focussing on the "discrimination" between the "good guys" (target word) and the bad guys (other words) that compete with it.

One realization of this idea is an algorithm that keeps a *moving average* approximation $\hat{Z}$ of Z, and uses importance sampling as before (but sampling both context h' and next word w'), as shown in Algorithm 4.

Algorithm 4 Joint Importance Sampling Gradient Approximation

```
(1) add pos. contribution: $\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta}$
(2) s= 0
(3) for (i=0;i<$N$;i++)
      Sample $(j_i, h_i) \sim Q(j_i, h_i)$
      compute forward pass $\mathcal{E}(j_i, h_i)$
      $w_i = \frac{e^{-\mathcal{E}(j_i,h_i)}}{NQ(j_i,h_i)}$
      s += $w_i$
(4) $\hat{Z} \leftarrow \alpha \hat{Z} + (1-\alpha)$ s
(5) add $N$ neg. contributions: $-\frac{\partial \mathcal{E}(j_i,h_i)}{\partial \theta}\frac{w_i}{\hat{Z}}$
```

3.1.5 Alternative Constraint on $\hat{Z}$

Common sense suggests that positive example gradient contributions and negative examples gradient contributions should have the same size in average, otherwise some parameters might diverge. For example, consider output biases or other parameters whose energy gradient is always positive. If the "positive" part of the log-likelihood gradient dominates, then these parameters will diverge to $-\infty$, whereas if the "negative" part of the log-likelihood gradient dominates, these parameters would diverge to ∞.

Furthermore, we know for sure that at learning convergence, the positive and negative contributions should have exactly the same "size" because the average gradient must be zero:

$$E[\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta_i}] = E[\sum_{w',h'} P(w',h')\frac{\partial \mathcal{E}(w',h')}{\partial \theta_i}] = E[\sum_{w',h'} \frac{e^{-\mathcal{E}(w',h')}}{Z}\frac{\partial \mathcal{E}(w',h')}{\partial \theta_i}]$$

where the above expectations are on the training set empirical distribution. Note that this is a set of equations (one per parameter θ_i).

Since Z only appears once (on the right-hand-side of the above equation), and as a scalar, there are *many constraints* on it due to the above equation. One way in which to approximately satisfy these constraints, is equalize the norms of the left and right-hand sides in the above equation. In the joint sampling cas, this yields to an algorithm for estimating Z precisely and cheaply, using the ratio of the norm of the average positive gradient contributions to the norm of the average negative gradient contributions.

3.2 Using Hierarchical Clustering to Speed Things up

Suppose we have a good hierarchical clustering of words (or of senses, then see section 3.6). That could be used to considerably speed-up both training and recognition of the neural network language model. The basic idea is to decompose the probability of the next word into a sequence of conditional probabilities, each time over a small number of choices (which correspond to nodes in the clustering hierarchy). This idea has already been used successfully in statistical language modeling (Gao, Goodman and Miao, 2001).

Formally, we decompose $P(w_t|h_t)$ as follows. Suppose $parent(node)$ is the parent of a node in the clustering hierarchy, with leaves of the hierarchy being words, and $root$ the root of the tree. Then the conditional probability of interest can be factored as follows:

$$P(w_t|h_t) = P(w_t|parent(w_t), h_t)P(parent(w_t)|parent(parent(w_t)), h_t) \ldots P(ancestor(w_t)|root, h_t)$$

where at each step the number of choices (which is proportional to the cost of computing that probability with an energy function) is just the number of children of a node in the hierarchy. For example if the clustering tree was binary, the total computation cost would scale as $\log_2 M$ (this is the number of energy functions to be computed), instead of M, which is a very significant speed-up. A special case is the flat clustering (one level, not hierarchical):

$$P(w_t|h_t) = P(w_t|cluster, h_t)P(cluster|h_t),$$

where the maximum saving occurs when the number of clusters is $\sqrt{M}$: the computation then scales as $\sqrt{M}$. Note that we don't need to sum over choices of parents in the above factorizations because the clustering is deterministic (hard clustering). If it wasn't (e.g. to deal with polysemy), then one would have to sum over the different alternative senses.

Preliminary experiments with a flat clustering (with a separate neural network $P(w_t|cluster, h_t)$ for each cluster, plus a "gating network" $P(cluster|h_t)$) suggest that generalization is hurt, probably because we are not sharing parameters (and not even the word features) between the different neural networks.

One way around that problem is to take advantage of the energy function formulation. The idea is use the **same neural network** for representing all of the $P(node|parent(node), h_t)$ conditional probabilities. We simply construct an energy function

$$\mathcal{E}(concept, parentconcept, h_t)$$

where the *concept* and *parentconcept* inputs are taken in a large common set that includes words and nodes of the hierarchy (and both are represented in the same manner at the input of the neural network). Normalization over the children of a node is used to obtain the proper conditional probabilities:

$$P(node|parent(node), h_t) = \frac{e^{-\mathcal{E}(node, parent(node), h_t)}}{\sum_{n \in children(parent(node))} e^{-\mathcal{E}(n, parent(node), h_t)}}$$

In addition to sharing the same parameters across all the levels of the hierarchical clustering, this approach has the advantage that it is also learning a semantic feature vector representation for the categories of the hierarchical clustering, not just for the words.

As in section 3.6, we can put in the following interesting smoothing trick. In the previously described neural network model, one can interpret the input vector for each word as a long "one-hot" bit vector (all entries being 0 except one with 1, corresponding to the input word). Once we know about the hierarchical structure above a word, then we can add a few more entries to this bit vector: we set to 1 the entries corresponding to the ancestors of the word. This way we bias the model towards generalization across words that are cousins in the hierarchy (but it is the neural network that chooses to pay more or less attention to the higher order bits).

3.3 Ontology Smoothed Bigram

One thing that n-grams and the neural network model presented above have in common is that they are not particularly well-suited to deal with the problem of *polysemy*. In particular, the neural network associates a single feature vector with each word, even though it may correspond to several very different meanings, which probably muddies the representation of meaning with these features.

To better understand and study this polysemy issue we propose a new type of graphical models for statistical language modeling, in which hidden variables explicitly represent the particular sense associated with an observed word, in its context. This is a place where we can easily take advantage of prior knowledge on the different senses of a word, e.g. from WordNet (Fellbaum, 1998). WordNet associates each word with a (usually very small) set of possible senses, and vice-versa, each of these senses (called "synset") may be expressed using one or more synonym words.

WordNet also provides a useful source of information in order to enhance generalization: the ontology. It is a graph (which is almost a tree), in which each internal node represents a more general concept than its children. There are sub-graphs for noun, verbs, adverbs and adjectives (unfortunately, though currently only the noun ontology is very deep and well structured, the others are still lacking in structure). Let us call categories the set of nodes of this graph, and senses those which are directly associated with words, with every sense being a category. In a simplified and not very compact way, this **is-a** ontology provides a **distributed representation for senses**. Imagine a long binary vector associated with each sense, with one entry per semantic category. It is mostly full of zeros, except at the entry associated with that sense, as well as for all the entries associated with all its ancestors in the graph. This idea is exploited in sections 3.2 and 3.6.

Let us for now consider how we could take advantage of these informations in a probabilistic graphical model. To simplify we will begin by studying (and hopefully improving) upon a very simple and common "object" of statistical language models, i.e. the **bigram**.

The two observed random variables are the *previous word* H (for history) and the *next word* W. Each of them is associated with a "true sense": random variable Z will be the "true sense" of H and random variable S will be the true sense of W. One simple model, at this point, is illustrated with the following graphical model:

$$\begin{tikzcd}
Z \arrow[r] & S \arrow[d] \\
H \arrow[u] & W
\end{tikzcd}$$

The goal is to represent the bigram probabilities, $P(W|H)$. Note that WordNet radically limits the set of allowed senses for each word, so the $P(Z|H)$ and $P(W|S)$ probability matrices are extremely sparse (mostly zeros). This strongly constrains the learning, but could we possibly learn these parameters from purely unlabeled data (i.e. for which the correct senses are not known)? or from only a small set of labeled data and a large set of unlabeled data? A difficulty that comes up is that some words are associated with multiple senses while each these senses is not associated with any other word. Assuming that learning is able to correctly cluster the instances of that word into appropriate groups, these groups might not correspond at all to the senses defined by WordNet.

One way to probably solve most of the cases of this problem, and also to introduce more smoothing into the model is to take advantage of the ontology. Significantly different senses of a word are likely to have a different parent in the ontology. The idea is to smooth the empirical transition probability matrix by *spreading probability mass to ontological neighbors.* This "spreading" must be done in a sense-dependendent way, because some neighbors may be more "replaceable" (closer semantically), etc...

For this purpose, we introduce two new random variables, E and F, representing high-level concepts associated respectively with the senses Z and S. A simple way to link them in is the following graphical model:

$$\begin{tikzcd}
E \arrow[r] & F \arrow[d] & \textbf{ontological category level} \\
Z \arrow[u] & S \arrow[d] & \textbf{sense-level} \\
H \arrow[u] & W & \textbf{word-level}
\end{tikzcd}$$

Unfortunately, this creates a problem. When we choose E or F we are choosing a level in the hierarchy. The higher in the hierarchy we chose, the more smoothing we impose. Therefore in the above model, the $P(E|Z)$ and $P(F|E)$ matrices are smoothing parameters! Unfortunately, $P(F|E)$ is also a very large matrix (it is the least sparse in the above model). Note that we can't estimate smoothing parameters by maximum likelihood on the training set (otherwise the model will choose to do no smoothing at all). One way to deal with smoothing parameters is to estimate them by maximum likelihood on a separate data set (the validation set), not used to estimate the other parameters. Unfortunately, in the above model, $P(F|E)$ is both the biggest parameter and the smoother.

We can finesse this problem by introducing an additional hidden variable, M, which is going to be a cousin meaning of S (in the sense of having a near common semantic ancestor F). The transition probabilities are then from E to M. This is illustrated in the following graphical model:

$$\begin{tikzcd}
E \arrow[dr] & & F \arrow[d] & \textbf{ontological category level} \\
Z \arrow[u] & M \arrow[ur] & S \arrow[d] & \textbf{sense-level} \\
H \arrow[u] & & W & \textbf{word-level}
\end{tikzcd}$$

Note that all the parameter matrices in the above model are extremely sparse (constrained by WordNet) except the one for $E \to M$. Using EM, the following matrices are learned on the training set: $H \to Z$, $E \to M$, $F \to S$, $S \to W$. Again using EM, the following matrices are learned on the validation set: $Z \to E$, $M \to F$. The overall training algorithm is summarized in **??**.

Algorithm 5 One EM epoch of the Ontology-Smoothed Bigram Training.

```
(1) Iterate over training set $(w_t, h_t)$ pairs
    (2) forward pass:  for each $Y \in \{Z, E, M, F, W\}$ (in this order)
      and $X \in \{H, Z, E, M, F\}$ compute $P(Y|H = h_t)$ using $P(X|H = h_t)$,
    (3) backward pass:  for each $Y \in \{Z, E, M, F, W\}$ (in reverse order)
      and $X \in \{H, Z, E, M, F\}$ compute $P(X|W = w_t, H = h_t)$
      using $P(Y|W = w_t, H = h_t)$, $P(Y|H = h_t)$ and $P(X|H = h_t)$, and
      increment posterior counts for $P(Y|X)$ if $Y \in \{Z, M, S, W\}$.
(4) Update training set parameters $P(Z|H), P(M|E), P(S|F), P(W|S)$.
(5) Iterate over validation set $(w_t, h_t)$ pairs
    (6) forward pass:  for each $Y \in \{Z, E, M, F, W\}$ (in this order)
      and $X \in \{H, Z, E, M, F\}$ compute $P(Y|H = h_t)$ using $P(X|H = h_t)$,
    (7) backward pass:  for each $Y \in \{Z, E, M, F, W\}$ (in reverse order)
      and $X \in \{H, Z, E, M, F\}$ compute $P(X|W = w_t, H = h_t)$
      using $P(Y|W = w_t, H = h_t)$, $P(Y|H = h_t)$ and $P(X|H = h_t)$, and
      increment posterior counts for $P(Y|X)$ if $Y \in \{E, F\}$.
(8) Update validation set parameters $P(E|Z), P(F|M)$.
```

The forward pass for each arc $X \to Y$ in Algorithm 5 proceeds as follows:

$$P(Y = y|H = h) = \sum_x P(Y = y|X = x)P(X = x|H = h)$$

where $P(Y|X)$ is the parameter matrix for this arc (and note that in practice all of the above probabilities are sparse). The backward pass for each arc propagates posteriors $P(Y|H = h, W = w)$ and increments counts $N(Y, X)$ as follows. First a joint posterior is computed (but needs not be stored):

$$P(X = x, Y = y|W = w_t, H = h_t) = \frac{P(Y = y|X = x)P(X = x|H = h_t)P(Y = y|W = w_t, H = h_t)}{P(Y = y|H = h_t)}$$

from which the hidden variable posteriors and the counts can be computed:

$$\begin{aligned} P(X = x|W = w_t, H = h_t) &= \sum_y P(X = x, Y = y|W = w_t, H = h_t) \\ N(Y = y, X = x) &= \sum_t P(X = x, Y = y|W = w_t, H = h_t) \end{aligned}$$

The parameter updates for each arc have the following form:

$$P(Y = y|X = x) \leftarrow \frac{N(Y = y, X = x)}{\sum_y N(Y = y, X = x)}.$$

Note that the above algorithm introduces some technical difficulties having to do with the sparseness of the matrices (hence of the posteriors) and the fact that some parameters are estimated on one data set and others are estimated on another. For example, some value m of M may not be reachable on the validation set (i.e. the posteriors $P(M = m|W = w_t, H = h_t)$ are 0 for all examples of the validation set), but may be reachable on the training set. Thus we may have some non-zero probability mass in $P(M = m|H)$ on the training set but the m column $P(F|M = m)$ of the $M \to F$ matrix may be empty (because no $M = m$ value has been "seen" on the validation set). This yields to incorrect probability computation ($\sum_f P(F = f|H) < 1$) and the same kind of problem yields to unfeasible re-estimation as well. One type of solution is to explicitly smooth the probability matrices, but this may unfortunately yield to unsparse parameter matrices (which would make the whole procedure extremely inefficient). The other solution (since these problems are rare) is to remove $M = m$ from the set of allowed values for M in the $P(M|E)$ matrix. Once such a thing this is done, note that one may have to propagate this "pruning" back in the chain. For example, removing some values of M removes the corresponding rows of $P(M|E)$, which may create "empty" columns in that matrix. These empty columns correspond to values of E for which we are now unable to make a prediction of M. These values of E should then be removed from $P(E|Z)$, i.e. the corresponding rows of $P(E|Z)$ should be removed, which may yield to empty columns in $P(E|Z)$, etc...

Note also that the $P(M|E)$ matrix is **sparse but much less** so than an ordinary $P(W|H)$ bigram matrix, because the smoothing process is spreading out mass to many neighbors: to each observed (W, H) correspond many (M, E) pairs compatible with the ontology. Thus a *"trigram" version of the above is not feasible* (at least with current typical computer memory). In the next section we study a way to incorporate dependencies beyond the previous word, using a stochastic switching process to control from which predecessor to base the dependence.

3.4 Multi-Step Markov Model

To extend the previous idea to high-order HMMs, while *allowing influences beyond the previous word*, yet *without the combinatorial explosion* that would come by building full tables of the joint dependencies of several previous categories, we introduce a **multi-step switching Markov process**. The basic idea is generate the high level category C_t from **one** of its predecessors C_{t-i}, according to a hidden control variable A_{t-i}. Therefore we have a mixture of bigrams of the form $P(C_t|C_{t-i})$, but no explicit higher order dependencies (beyond the pairs of categories). The hidden control variable sequence is the $\{A_t\}$ process, with $A_t \in \{1, \ldots, H\}$, where H here is the maximum horizon over which we are willing to consider direct dependencies. This random sequence generates an "output" sequence $\{C_t\}$ (e.g. the sequence of high-level categories). The generative process is formalized through a generative algorithm:

```
$t$=1
sample $C_1$ from prior $P(C_1)$
while $t < T$:
    sample $A_t \sim P(A_t)$
    sample $C_{t+A_t} \sim P(C_{t+A_t}|C_t, \Delta = A_t)$
    for $i \in [1, A_t - 1]$
        $A_{t+i} = -1$
        sample $C_{t+i} \sim P(C_{t+i}|C_{t+i-1}, \Delta = 1)$
    $t = t + A_t$
```

Note that it is not difficult to perform inference on this simple graphical model, and thus to apply the EM algorithm. However, our real objective is to generate a word sequence, through senses and categories, following the pattern introduced in section 3.3.

3.5 Ontology Smoothed Multi-HMM

To achieve that objective, we propose the dynamical graphical model with the following sequences of random variables:

- $\{W_t\}$ is the observed word sequence.
- $\{S_t\}$ is the sequence of senses S_t of W_t.
- $\{M_t\}$ is the sequence of "cousin" senses of S_t (both share the ontological ancestor C_t).
- $\{C_t\}$ is the high-level semantic category C_t that is an ancestor of S_t and M_t, and from which C_{t+i} will be indirectly emitted (through M_{t+i}), where the particular choice of i is determined by the $\{A_t\}$ sequence.
- $\{A_t\}$ is the sequence of switching jump decisions for the multi-step Markov model (when $A_t = i$, then C_t is chosen as the activated parent for C_{t+i}).

Figure 3 illustrates the interactions between the above variables.

Like for the smoothed bigram model of section 3.3, some parameters are smoothing parameters and should be learned on a separate data set (validation set): $P(C_t|M_t)$. They choose the level of an ontological ancestor (i.e. how to spread probability mass through that ontological ancestor).

Again inference in this model is reasonably cheap (so EM can be used), essentially increasing the computational cost of the smoothed bigram of section 3.3 by a factor H (the maximum horizon).

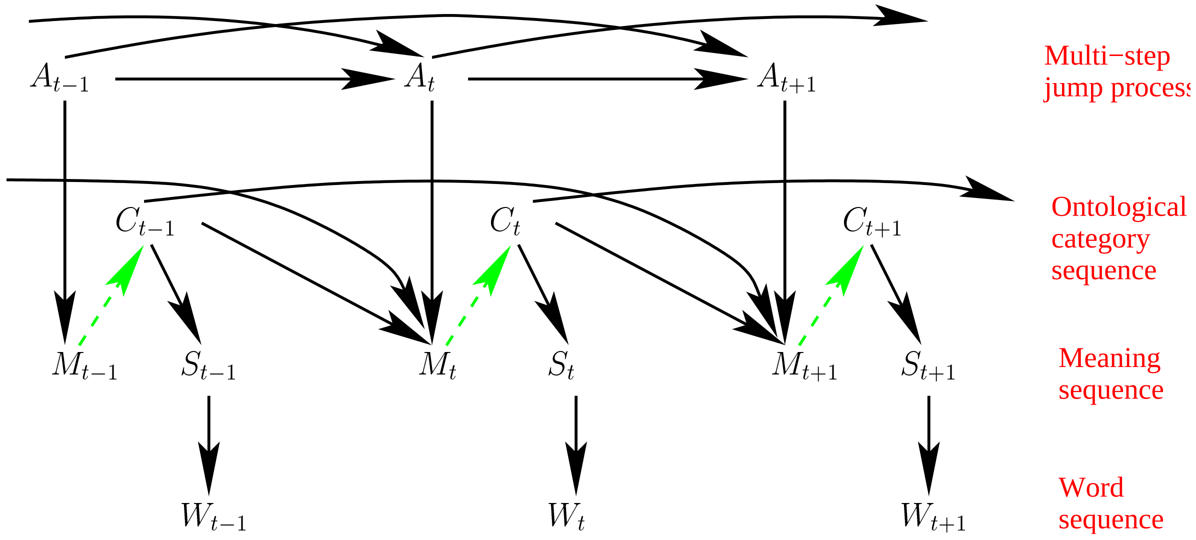


Figure 3: An graphical model illustration of the multi-step HMM. Note that because A_t is a switching variable, depending on its value, only one of the C_{t-i} parents of M_t is actually conditioning M_t. The dashed arrows represent smoothing parameters, learned on a validation set.

3.6 Plugging WordNet in the Neural Network

As mentionned in section 3.3, the neural network model of section 2.4 might be confused by polysemic words (i.e. most words), trying to infer an internal representation (the word features) that are simultaneously right for multiple senses of the word.

One way to deal with this would be to feed the neural network with **senses** rather than **words**. An external inference process would be estimating the probability of the sense sequence, and could be used to compute posteriors over the allowed senses (that can then be used to train the neural network by generalized EM, e.g. as in Input/Output HMMs (Bengio and Frasconi, 1996)).

In addition of using WordNet to constrain the possible senses associated with a word, as suggested at the end of section 3.2, we can take advatange of WordNet to provide further generalization bias to the neural network, by using a somewhat distributed code for the senses, that incorporates the ontological structure. The raw input of the regular neural network is really a one-hot code for the word (one bit is 1, the other 0's). Instead consider for each input sense a bit vector of length equal to the number of WordNet categories (which include senses). The entry corresponding to the input sense would be set to 1, along with the entries corresponding to the sense's ancestors in the ontology.

Since senses are not observed, we can insert the neural network in a probabilistic model of the possible word senses:

$$P(W_1^T, S_1^T) = \prod_t P(W_t|S_t)P(S_t|S_{t-1}, \ldots, S_{t-n})$$

where $P(S_t|S_{t-1}, \ldots, S_{t-n})$ is the neural network and $P(W_t|S_t)$ can be more simply represented by a table (it is a very sparse matrix because of WordNet). The number of sense sequences to consider to compute the complete likelihood grows exponentially with n (but slowly because the number of senses associated with each word is small, e.g. typically less than a handful). Instead one could use an approximation of the likelihood based on its largest terms, e.g. using **beam-search** to compute probabilities only for the most likely sense sequences.

3.7 Neural Lexicalized Stochastic Grammars

Stochastic grammars (or probabilistic grammars) allow representing recursive and long-term structure, through a syntactic parse tree of each sentence. The hidden variable is the parse tree itself, which hierarchically groups consecutive words into phrases and assigns each word and each of these phrases a syntactic role (a tag or grammar non-terminal symbol). For example a probabistic grammar can be obtained by assigning a conditional probability to each generative rule of a context free grammar, whose terminal symbols are the words.

Unfortunately, by not taking into account of most of the semantic dependencies, these grammars do not yield very good perplexity. In the last decade, several results have shown that much better performance can be obtained with **lexicalized stochastic grammars**, such as (Charniak, 1999; Collins, 1997; Collins, 1999; Chelba and Jelinek, 2000; Ratnaparkhi, 1999), in which each node of the parse tree is also associated with a word (known as the **head word**) from the phrase that it dominates. By conditionning the rules on the head words, much richer models are possible. For example, the Structured Language Model (Chelba and Jelinek, 2000) contains the trigram as a special case.

The above model can yield significantly improved perplexity in comparison to a state-of-the-art trigram model, both in terms of perplexity and in terms of speech recognition word error rate (Chelba and Jelinek, 2000). This model generates the sentence and a binary parse tree from the left to the right, by incrementally adding words and possibly making the last two adjacent trees children of a new root node and choosing which head word to percolate. The main components of the model are the following conditional probabilities:

- $P(\text{next word}|\text{previous heads and their tags})$
- $P(\text{next tag}|\text{next word, previous tags})$
- $P(\text{parsing action}|\text{last head words})$

Inference (and thus parsing) is difficult) $O(n^6)$ (Jelinek and Chelba, 1999) for a sequence of length n, therefore in practice one uses a beam search and the N-best solutions for parsing and training.

One of the limitations of the above model is that we would like to have many conditionning variables in the above components, but even with the above variables one has to use smoothing strategies (back-off or deleted interpolation). This is a typical case where we want to model the distribution between many discrete random variables but we are facing the curse of dimensionality.

What we propose here is to use a neural network model similar to the one introduced in section 3.6, i.e. with senses in input, in order to represent the above tables (possibly with more conditioning variables than in the current Structured Language Model) in a *statistically* (not *computationaly*) more efficient way. The general features of the proposed model are thus the following:

- Each of the above 3 types of conditional probabilities is represented by a neural network (in some sense this is like the use of neural networks in Input/Output HMMs (Bengio and Frasconi, 1996)).
- The grammar generates **senses** rather than **words**.
- The sense inputs of the neural network incorporates the high-level category bits, as suggested in section 3.6.
- Parameter sharing of the first layer "features" is done across inputs of the same "type", e.g. tags and senses each have their matrix of features.

Obviously, for the proposals of the last two sections to become computationally realistic, we first have to make more progress on the algorithms to speed-up training and computations in the "neural" language model. Indeed, in these two approaches, the neural network computations are within an outer loop of "hidden variable search" (e.g. in the space of sense sequences, or in the space of parse trees).

4 Conclusions

This paper has argued for distributed representations to represent high-dimensional dependencies in statistical language modeling, and presented new learning techniques for arbitrary energy-based models based on importance sampling, new graphical models to take advantage of WordNet's synsets and ontology, and extensions of the neural network model to take advantage of our knowledge of polysemy, ontology, and grammatical structure.In particular, the Monte-Carlo sampling techniques that we have presented avoid the costly exact gradient computation that involves computations proportional to the vocabulary size each time an example is presented, because of the contribution of the partition function of the probability model.

Many ideas have been presented but most of the work remains to be done in order to test them and refine them to make them work, and probably face unforeseen challenges. Overall, the central message of this paper is that the main role of machine learning in statistical language modeling is in dealing with the curse of dimensionality. To face that challenge, many computational questions have been raised and this paper has presented some working solutions, and proposals for new directions of improvement.

Acknowledgments

The author would like to thank Geoffrey Hinton, Jean-Sébastien Senecal and Christian Jauvin for useful discussions and feedback. This research was made possible by funding from the NSERC granting agency, the MITACS and IRIS networks, and the Canada Research Chairs.

References

Bengio, S. and Bengio, Y. (2000a). Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE Transactions on Neural Networks, special issue on Data Mining and Knowledge Discovery*, 11(3):550–557.

Bengio, Y. and Bengio, S. (2000b). Modeling high-dimensional discrete data with multi-layer neural networks. In Solla, S. A., Leen, T. K., and Müller, K.-R., editors, *Advances in Neural Information Processing Systems*, volume 12, pages 400–406. MIT Press.

Bengio, Y., Ducharme, R., and Vincent, P. (2002). A neural probabilistic language model. Technical Report 1198, Dept. IRO, Université de Montréal.

Bengio, Y. and Frasconi, P. (1996). Input/Output HMMs for sequence processing. *IEEE Transactions on Neural Networks*, 7(5):1231–1249.

Berger, A., Della Pietra, S., and Della Pietra, V. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22:39–71.

Charniak, E. (1999). A maximum-entropy-inspired parser. Technical Report CS-99-12, Brown University.

Chelba, C. and Jelinek, F. (2000). Structured language modelin. *Computer, Speech and Language*, 14(4):282–332.

Collins, M. (1997). Three generative, lexicalized models for statistical parsing. In *35th Annual Meeting of the ACL*, pages 16–23, Madrid, Spain.

Collins, M. (1999). *Head-driven statistical models for natural language parsing.* PhD thesis, University of Pennsylvania.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., and R.Harshman (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database.* MIT Press.

Foster, G. (2002). *Text Prediction for Translators.* PhD thesis, Dept. IRO, Université de Montréal.

Gao, J., Goodman, J., and Miao, J. (2001). The use of clustering techniques for asian language modeling. In *Computational Linguistics and Chinese Language Processing*, volume 6(1), pages 27–60.

Genest, C. and Zideck, J. (1986). Combining probability distributions: A critique and an annotated bibliography. *Statistical Science*, 1:114–148.

Goodman, J. (2001). A bit of progress in language modeling. Technical Report MSR-TR-2001-72, Microsoft Research.

Heskes, T. (1998). Bias/variance decompositions for likelihood-based estimators. *Neural Computation*, 10:1425–1433.

Hinton, G. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 1–12, Amherst 1986. Lawrence Erlbaum, Hillsdale.

Hinton, G. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800.

Jaakkola, T. and Jordan, M. (1999). Varitional methods and the qmr-dt database. *Journal of Artificial Intelligence*, 10:291–322.

Jelinek, F. and Chelba, C. (1999). Putting language into language modeling. In *European Conference on Speech Communication and Technology*, volume 1, pages KN1–KN5, Budapest.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In Gelsema, E. S. and Kanal, L. N., editors, *Pattern Recognition in Practice*. North-Holland, Amsterdam.

Jensen, K. and Riis, S. (2000). Self-organizing letter code-book for text-to-phoneme neural network model. In *Proceedings ICSLP*.

Jordan, M. (1998). *Learning in Graphical Models*. Kluwer, Dordrecht, Netherlands.

Jordan, M., Ghahramani, Z., Jaakkola, T., and Saul, L. (1999). An introduction to variational methods in graphical models. In Jordan, M., editor, *Learning in Graphical Models*. MIT Press, Cambridge, MA.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(3):400–401.

Manning, C. and Schutze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

Miikkulainen, R. and Dyer, M. (1991). Natural language processing with modular neural networks and distributed lexicon. *Cognitive Science*, 15:343–399.

Paccanaro, A. and Hinton, G. (2000). Extracting distributed representations of concepts and relations from positive and negative propositions. In *Proceedings of the International Joint Conference on Neural Network, IJCNN'2000*, Como, Italy. IEEE, New York.

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, 341(2):151–176.

Riis, S. and Krogh, A. (1996). Improving protein secondary structure prediction using structured neural networks and multiple sequence profiles. *Journal of Computational Biology*, pages 163–183.

Schmidhuber, J. (1996). Sequential neural text compression. *IEEE Transactions on Neural Networks*, 7(1):142–146.

Schutze, H. (1993). Word space. In Hanson, S. J., Cowan, J. D., and Giles, C. L., editors, *Advances in Neural Information Processing Systems 5*, pages pp. 895–902, San Mateo CA. Morgan Kaufmann.

Weiss, Y. (2000). Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12:1–41.