Hierarchical Probabilistic Neural Network Language Model

Frederic Morin
Dept. IRO, Université de Montréal
P.O. Box 6128, Succ. Centre-Ville,
Montreal, H3C 3J7, Qc, Canada
morinf@iro.umontreal.ca

Yoshua Bengio
Dept. IRO, Université de Montréal
P.O. Box 6128, Succ. Centre-Ville,
Montreal, H3C 3J7, Qc, Canada
Yoshua.Bengio@umontreal.ca

Abstract

In recent years, variants of a neural network architecture for statistical language modeling have been proposed and successfully applied, e.g. in the language modeling component of speech recognizers. The main advantage of these architectures is that they learn an embedding for words (or other symbols) in a continuous space that helps to smooth the language model and provide good generalization even when the number of training examples is insufficient. However, these models are extremely slow in comparison to the more commonly used n-gram models, both for training and recognition. As an alternative to an importance sampling method proposed to speed-up training, we introduce a hierarchical decomposition of the conditional probabilities that yields a speed-up of about 200 both during training and recognition. The hierarchical decomposition is a binary hierarchical clustering constrained by the prior knowledge extracted from the WordNet semantic hierarchy.

1 INTRODUCTION

The curse of dimensionality hits hard statistical language models because the number of possible combinations of n words from a dictionnary (e.g. of 50,000 words) is immensely larger than all the text potentially available, at least for $n > 2$. The problem comes down to transfering probability mass from the tiny fraction of observed cases to all the other combinations. From the point of view of machine learning, it is interesting to consider the different principles at work in obtaining such generalization. The most fundamental principle, used explicitly in non-parametric models, is that of similarity: if two objects are similar they should have a similar probability. Unfortunately, using a knowledge-free notion of similarity does not work well in high-dimensional spaces such as sequences of words. In the case of statistical language models, the most successful generalization principle (and corresponding notion of similarity) is also a very simple one, and it is used in interpolated and back-off n-gram models (**?**; **?**): sequences that share shorter subsequences are similar and should share probability mass.

However, these methods are based on exact matching of subsequences, whereas it is obvious that two word sequences may not match and yet be very close to each other semantically. Taking this into account, another principle that has been shown to be very successful (in combination with the first one) is based on a notion of similarity between individual words: two word sequences are said to be "similar" if their corresponding words are "similar". Similarity between words is usually defined using **word classes** (**?**; **?**). These word classes correspond to a partition of the set of words in such a way that words in the same class share statistical properties in the context of their use, and this partition can be obtained with various clustering algorithms. This is a discrete all-or-nothing notion of similarity. Another way to define similarity between words is based on assigning each word to a continuous-valued set of features, and comparing words based on this feature vector. This idea has already been exploited in information retrieval (**?**; **?**) using a singular value decomposition of a matrix of occurrences, indexed by words in one dimension and by documents in the other.

This idea has also been exploited in (**?**; **?**): a neural network architecture is defined in which the first layer maps word symbols to their continuous representation as feature vectors, and the rest of the neural network is conventional and used to construct conditional probabilities of the next word given the previous ones. This model is described in detail in Section **??**. The idea is to exploit the smoothness of the neural network to make sure that sequences of words that are similar according to this learned metric will be assigned a similar probability. Note that both the feature vectors and the part of the model that computes probabilities from them are estimated jointly, by regularized maximum

likelihood. This type of model is also related to the popular maximum entropy models (**?**) since the latter correspond to a neural network with no hidden units (the unnormalized log-probabilities are linear functions of the input indicators of presence of words).

This neural network approach has been shown to generalize well in comparison to interpolated n-gram models and class-based n-grams (**?**; **?**; **?**; **?**; **?**), both in terms of perplexity and in terms of classification error when used in a speech recognition system (**?**; **?**; **?**). In (**?**; **?**), it is shown how the model can be used to directly improve speech recognition performance. In (**?**), the approach is generalized to form the various conditional probability functions required in a stochastic parsing model called the Structured Language Model, and experiments also show that speech recognition performance can be improved over state-of-the-art alternatives. However, a major weakness of this approach is the very long training time as well as the large amount of computations required to compute probabilities, e.g. at the time of doing speech recognition (or any other application of the model). Note that such models could be used in other applications of statistical language modeling, such as automatic translation and information retrieval, but improving speed is important to make such applications possible.

The objective of this paper is thus to propose a much faster variant of the neural probabilistic language model. It is based on an idea that could in principle deliver close to exponential speed-up with respect to the number of words in the vocabulary. Indeed the computations required during training and during probability prediction are a small constant plus a factor linearly proportional to the number of words $|V|$ in the vocabulary V. The approach proposed here can yield a speed-up of order $O(\frac{|V|}{\log |V|})$ for the second term. It follows up on a proposal made in (**?**) to rewrite a probability function based on a partition of the set of words. The basic idea is to form a hierarchical description of a word as a sequence of $O(\log |V|)$ decisions, and to learn to take these probabilistic decisions instead of directly predicting each word's probability. Another important idea of this paper is to reuse the same model (i.e. the same parameters) for all those decisions (otherwise a very large number of models would be required and the whole model would not fit in computer memory), using a special symbolic input that characterizes the nodes in the tree of the hierarchical decomposition. Finally, we use prior knowledge in the WordNet lexical reference system to help define the hierarchy of word classes.

2 PROBABILISTIC NEURAL LANGUAGE MODEL

The objective is to estimate the joint probability of sequences of words and we do it through the estimation of the conditional probability of the next word (the **target word**) given a few previous words (the **context**):

$$P(w_1, \ldots, w_l) = \prod_t P(w_t | w_{t-1}, \ldots, w_{t-n+1}),$$

where w_t is the word at position t in a text and $w_t \in V$, the vocabulary. The conditional probability is estimated by a normalized function $f(w_t, w_{t-1}, \ldots, w_{t-n+1})$, with $\sum_v f(v, w_{t-1}, \ldots, w_{t-n+1}) = 1$.

In (**?**; **?**) this conditional probability function is represented by a neural network with a particular structure. Its most important characteristic is that each input of this function (a word symbol) is first embedded into a Euclidean space (by learning to associate a real-valued "feature vector" to each word). The set of feature vectors for all the words in V is part of the set of parameters of the model, estimated by maximizing the empirical log-likelihood (minus weight decay regularization). The idea of associating each symbol with a distributed continuous representation is not new and was advocated since the early days of neural networks (**?**; **?**). The idea of using neural networks for language modeling is not new either (**?**, e.g.)]Miikkulainen91,Xu+Rudnicky-2000, and is similar to proposals of character-based text compression using neural networks to predict the probability of the next character (**?**).

There are two variants of the model in (**?**; **?**): one with $|V|$ outputs with softmax normalization (and the target word w_t is not mapped to a feature vector, only the context words), and one with a single output which represents the unnormalized probability for w_t given the context words. Both variants gave similar performance in the experiments reported in (**?**; **?**). We will start from the second variant here, which can be formalized as follows, using the Boltzmann distribution form, following (**?**):

$$f(w_t, w_{t-1}, \ldots, w_{t-n+1}) = \frac{e^{-g(w_t, w_{t-1}, \ldots, w_{t-n+1})}}{\sum_v e^{-g(v, w_{t-1}, \ldots, w_{t-n+1})}}$$

where $g(v, w_{t-1}, \ldots, w_{t-n+1})$ is a learned function that can be interpreted as an energy, which is low when the tuple $(v, w_{t-1}, \ldots, w_{t-n+1})$ is "plausible".

Let F be an embedding matrix (a parameter) with row F_i the embedding (feature vector) for word i. The above energy function is represented by a first transformation of the input label through the feature vectors F_i, followed by an ordinary feedforward neural network (with a single output and a bias dependent on v):

$$g(v, w_{t-1}, \ldots, w_{t-n+1}) = a'.\tanh(c + Wx + UF'_v) + b_v \quad (1)$$

where x' denotes the transpose of x, $\tanh$ is applied element by element, a, c and b are parameters vectors, W and U are weight matrices (also parameters), and x denotes the concatenation of input feature vectors for context words:

$$x = (F_{w_{t-1}}, \ldots, F_{w_{t-n+1}})'. \quad (2)$$

Let h be the number of hidden units (the number of rows of W) and d the dimension of the embedding (number of columns of F). Computing $f(w_t, w_{t-1}, \ldots, w_{t-n+1})$ can be done in two steps: first compute $c + Wx$ (requires $hd(n-1)$ multiply-add operations), and second, for each $v \in V$, compute UF'_v (hd multiply-add operations) and the value of $g(v, ...)$ (h multiply-add operations). Hence total computation time for computing f is on the order of $(n-1)hd + |V|h(d+1)$. In the experiments reported in (**?**), n is around 5, $|V|$ is around 20000, h is around 100, and d is around 30. This gives around 12000 operations for the first part (independent of $|V|$) and around 60 million operations for the second part (that is linear in $|V|$).

Our *goal in this paper is to drastically reduce the second part*, ideally by replacing the $O(|V|)$ computations by $O(\log |V|)$ computations.

3 HIERARCHICAL DECOMPOSITION CAN PROVIDE EXPONENTIAL SPEED-UP

In (**?**) it is shown how to speed-up a maximum entropy class-based statistical language model by using the following idea. Instead of computing directly $P(Y|X)$ (which involves normalization across all the values that Y can take), one defines a clustering partition for the Y (into the word classes C, such that there is a deterministic function $c(.)$ mapping Y to C), so as to write

$$P(Y = y|X = x) = P(Y = y|C = c(y), X)P(C = c(y)|X = x).$$

This is always true for any function $c(.)$ because $P(Y|X) = \sum_i P(Y, C = i|X) = \sum_i P(Y|C = i, X)P(C = i|X) = P(Y|C = c(Y), X)P(C = c(Y)|X)$ because only one value of C is compatible with the value of Y, the value $C = c(Y)$.

Although any $c(.)$ would yield correct probabilities, generalization could be better for choices of word classes that "make sense", i.e. those for which it easier to learn the $P(C = c(y)|X = x)$.

If Y can take 10000 values and we have 100 classes with 100 words y in each class, then instead of doing normalization over 10000 choices we only need to do two normalizations, each over 100 choices. If computation of conditional probabilities is proportional to the number of choices then the above would reduce computation by a factor 50. This is approximately what is gained according to the measurements reported in (**?**). The same paper suggests that one could introduce more levels to the decomposition and here we push this idea to the limit. Indeed, whereas a one-level decomposition should provide a speed-up on the order of $\frac{|V|}{\sqrt{|V|}} = \sqrt{|V|}$, a hierarchical decomposition represented by a balanced binary tree should provide an exponential speed-up, on the order of $\frac{|V|}{\log_2 |V|}$ (at least for the part of the computation that is linear in the number of choices).

Each word v must be represented by a bit vector $(b_1(v), \ldots b_m(v))$ (where m depends on v). This can be achieved by building a binary hierarchical clustering of words, and a method for doing so is presented in the next section. For example, $b_1(v) = 1$ indicates that v belongs to the top-level group 1 and $b_2(v) = 0$ indicates that it belongs to the sub-group 0 of that top-level group.

The next-word conditional probability can thus be represented and computed as follows:

$$P(v|w_{t-1}, \ldots, w_{t-n+1}) = \prod_{j=1}^{m} P(b_j(v)|b_1(v), \ldots, b_{j-1}(v), w_{t-1}, \ldots, w_{t-n+1})$$

This can be interpreted as a series of binary stochastic decisions associated with nodes of a binary tree. Each node is indexed by a bit vector corresponding to the path from the root to the node (append 1 or 0 according to whether the left or right branch of a decision node is followed). Each leaf corresponds to a word. If the tree is balanced then the maximum length of the bit vector is $\lceil \log_2 |V| \rceil$. Note that we could further reduce computation by looking for an encoding that takes the frequency of words into account, to reduce the average bit length to the unconditional entropy of words. For example with the corpus used in our experiments, $|V| = 10000$ so $\log_2 |V| \approx 13.3$ while the unigram entropy is about 9.16, i.e. a possible additional speed-up of 31% when taking word frequencies into account to better balance the binary tree. The gain would be greater for larger vocabularies, but not a very significant improvement over the major one obtained by using a simple balanced hierarchy.

The "target class" (0 or 1) for each node is obtained directly from the target word in each context, using the bit encoding of that word. Note also that there will be a target

(and gradient propagation) only for the nodes on the path from the root to the leaf associated with the target word. This is the major source of savings during training.

During recognition and testing, there are two main cases to consider: one needs the probability of only one word, e.g. the observed word, (or very few) , or one needs the probabilities of all the words. In the first case (which occurs during testing on a corpus) we still obtain the exponential speed-up. In the second case, we are back to $O(|V|)$ computations (with a constant factor overhead). For the purpose of estimating generalization performance (out-of-sample log-likelihood) only the probability of the observed next word is needed. And in practical applications such as speech recognition, we are only interested in discriminating between a few alternatives, e.g. those that are consistent with the acoustics, and represented in a treillis of possible word sequences.

This speed-up should be contrasted with the one provided by the importance sampling method proposed in (**?**). The latter method is based on the observation that the log-likelihood gradient is the average over the model's distribution for $P(v|context)$ of the energy gradient associated with all the possible next-words v. The idea is to approximate this average by a biased (but asymptotically unbiased) importance sampling scheme. This approach can lead to significant speed-up during training, but because the architecture is unchanged, probability computation during recognition and test still requires $O(|V|)$ computations for each prediction. Instead, the architecture proposed here gives significant speed-up both during training and test / recognition.

4 SHARING PARAMETERS ACROSS THE HIERARCHY

If a separate predictor is used for each of the nodes in the hierarchy, about $2|V|$ predictors will be needed. This represents a huge capacity since each predictor maps from the context words to a single probability. This might create problems in terms of computer memory (not all the models would fit at the same time in memory) as well as overfitting. Therefore we have chosen to build a model in which parameters are shared across the hierarchy. There are clearly many ways to achieve such sharing, and alternatives to the architecture presented here should motivate further study.

Based on our discussion in the introduction, it makes sense to force the word embedding to be shared across all nodes. This is important also because the matrix of word features F is the largest component of the parameter set.

Since each node in the hierarchy presumably has a semantic meaning (being associated with a group of hopefully similar-meaning words) it makes sense to also associate each node with a feature vector. Without loss of generality, we can consider the model to predict $P(b|node, w_{t-1}, \ldots, w_{t-n+1})$ where $node$ corresponds to a sequence of bits specifying a node in the hierarchy and b is the next bit (0 or 1), corresponding to one of the two children of $node$. This can be represented by a model similar to the one described in Section **??** and (**?**; **?**) but with two kinds of symbols in input: the context words and the current node. We allow the embedding parameters for word cluster nodes to be different from those for words. Otherwise the architecture is the same, with the difference that there are only two choices to predict, instead of $|V|$ choices.

More precisely, the specific predictor used in our experiments is the following:

$$P(b = 1|node, w_{t-1}, \ldots, w_{t-n+1}) = \\ \text{sigmoid}(\alpha_{node} + \beta'. \tanh(c + Wx + UN_{node}))$$

where x is the concatenation of context word features as in eq. **??**, $\text{sigmoid}(y) = 1/(1 + exp(-y))$, α_i is a bias parameter playing the same role as b_v in eq. **??**, β is a weight vector playing the same role as a in eq. **??**, c, W, U and F play the same role as in eq. **??**, and N gives feature vector embeddings for nodes in a way similar that F gave feature vector embeddings for next-words in eq. **??**.

5 USING WORDNET TO BUILD THE HIERARCHICAL DECOMPOSITION

A very important component of the whole model is the choice of the words binary encoding, i.e. of the hierarchical word clustering. In this paper we combine empirical statistics with prior knowledge from the WordNet resource (**?**). Another option would have been to use a purely data-driven hierarchical clustering of words, and there are many other ways in which the WordNet resource could have been used to influence the resulting clustering.

The IS-A taxonomy in WordNet organizes semantic concepts associated with senses in a graph that is almost a tree. For our purposes we need a tree, so we have manually selected a parent for each of the few nodes that have more than one parent. The leaves of the WordNet taxonomy are senses and each word can be associated with more than one sense. Words sharing the same sense are considered to be synonymous (at least in one of their uses). For our purpose we have to choose one of the senses for each word (to make the whole hierarchy one over words) and we selected the most frequent sense. A straightforward extension of the proposed model would keep the semantic ambiguity of each word: each word would be associated with several leaves (senses) of the WordNet hierarchy. This would

require summing over all those leaves (and corresponding paths to the root) when computing next-word probabilities.

Note that the WordNet tree is not binary: each node may have many more than two children (this is particularly a problem for verbs and adjectives, for which WordNet is shallow and incomplete). To transform this hierarchy into a binary tree we perform a data-driven binary hierarchical clustering of the children associated with each node, as illustrated in Figure **??**. The K-means algorithm is used at each step to split each cluster. To compare nodes, we associate each node with the subset of words that it covers. Each word is associated with a TF/IDF (**?**) vector of document/word occurrence counts, where each "document" is a paragraph in the training corpus. Each node is associated with the dimension-wise median of the TF/IDF scores. Each TF/IDF score is the occurrence frequency of the word in the document times the logarithm of the ratio of the total number of documents by the number of documents containing the word.

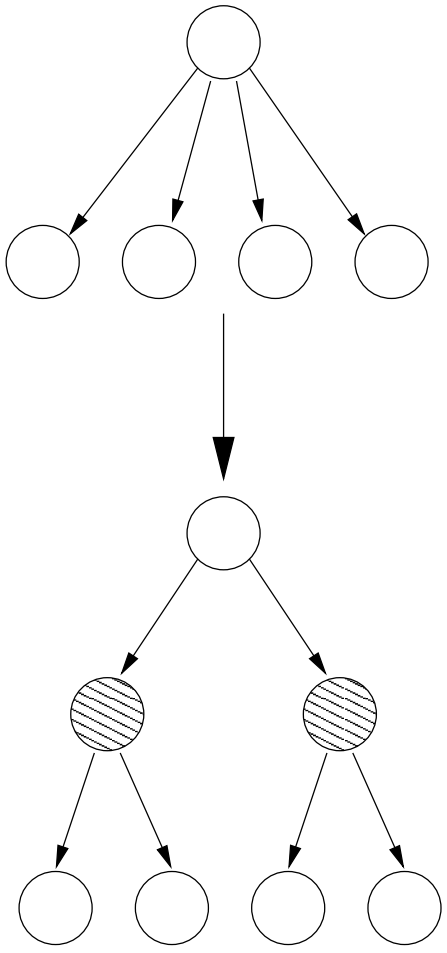

Figure 1: *WordNet's IS-A hierarchy is not a binary tree: most nodes have many children. Binary hierarchical clustering of these children is performed.*

6 COMPARATIVE RESULTS

Experiments were performed to evaluate the speed-up and any change in generalization error. The experiments also compared an alternative speed-up technique (**?**) that is based on importance sampling (but only provides a speed-up during training). The experiments were performed on the Brown corpus, with a reduced vocabulary size of 10,000 words (the most frequent ones). The corpus has 1,105,515 occurrences of words, split into 3 sets: 900,000 for training, 100,000 for validation (model selection), and 105,515 for testing. The validation set was used to select among a small number of choices for the size of the embeddings and the number of hidden units.

architecture	Time per epoch (s)	Time per ex. (ms)	speed-up
original neural net	416 300	462.6	1
importance sampling	6 062	6.73	68.7
hierarchical model	1 609	1.79	258

Table 1: *Training time per epoch (going once through all the training examples) and per example. The original neural net is as described in sec.* **??**. *The importance sampling algorithm (***?***) trains the same model faster. The hierarchical model is the one proposed here, and it yields a speed-up not only during training but for probability predictions as well (see the next table).*

architecture	Time per example (ms)	speed-up
original neural net	270.7	1
importance sampling	221.3	1.22
hierarchical model	1.4	193

Table 2: *Test time per example for the different algorithms. See Table 1's caption. It is at test time that the hierarchical model's advantage becomes clear in comparison to the importance sampling technique, since the latter only brings a speed-up during training.*

The results in terms of raw computations (time to process one example), either during training or during test are shown respectively in Tables **??** and **??**. The computations were performed on Athlon processors with a 1.2 GHz clock. The speed-up during training is by a factor greater than 250 and during test by a factor close to 200. These are impressive but less than the $|V|/\log_2|V| \approx 750$ that could be expected if there was no overhead and no constant term in the computational cost.

It is also important to verify that learning still works and that the model generalizes well. As usual in statistical language modeling this is measured by the model's

	Validation perplexity	Test perplexity
trigram	299.4	268.7
class-based	276.4	249.1
original neural net	213.2	195.3
importance sampling	209.4	192.6
hierarchical model	241.6	220.7

Table 3: *Test perplexity for the different architectures and for an interpolated trigram. The hierarchical model performed a bit worse than the original neural network, but is still better than the baseline interpolated trigram and the class-based model.*

perplexity on the test data, which is the exponential of the average negative log-likehood on that data set. Training is performed over about 20 to 30 epochs according to validation set perplexity (early stopping). Table **??** shows the comparative generalization performance of the different architectures, along with that of an interpolated trigram and a class-based n-gram (same procedures as in (**?**), which follow respectively (**?**) and (**?**; **?**; **?**)). The validation set was used to choose the order of the n-gram and the number of word classes for the class-based models. We used the implementation of these algorithms in the SRI Language Modeling toolkit, described by (**?**) and in `www.speech.sri.com/projects/srilm/`. Note that better performance should be obtainable with some of the tricks in (**?**). Combining the neural network with a trigram should also decrease its perplexity, as already shown in (**?**).

As shown in Table **??**, the hierarchical model does not generalize as well as the original neural network, but the difference is not very large and still represents an improvement over the benchmark n-gram models. Given the very large speed-up, it is certainly worth investigating variations of the hierarchical model proposed here (in particular how to define the hierarchy) for which generalization could be better. Note also that the speed-up would be greater for larger vocabularies (e.g. 50,000 is not uncommon in speech recognition systems).

7 CONCLUSION AND FUTURE WORK

This paper proposes a novel architecture for speeding-up neural networks with a huge number of output classes and shows its usefulness in the context of statistical language modeling (which is a component of speech recognition and automatic translation systems). This work pushes to the limit a suggestion of (**?**) but also introduces the idea of sharing the same model for all nodes of the decomposition, which is more practical when the number of nodes is very large (tens of thousands here). The implementation and the experiments show that a very significant speed-up of around 200-fold can be achieved, with only a little degradation in generalization performance.

From a linguistic point of view, one of the weaknesses of the above model is that it considers word clusters as deterministic functions of the word, but uses the nodes in WordNet's taxonomy to help define those clusters. However, WordNet provides word sense ambiguity information which could be used for linguistically more accurate modeling. The hierarchy would be a sense hierarchy instead of a word hiearchy, and each word would be associated with a number of senses (those allowed for that word in WordNet). In computing probabilities, this would involve summing over several paths from the root, corresponding to the different possible senses of the word. As a side effect, this could provide a word sense disambiguation model, and it could be trained both on sense-tagged supervised data and on unlabeled ordinary text. Since the average number of senses per word is small (less than a handful), the loss in speed would correspondingly be small.

Acknowledgments

The authors would like to thank the following funding organizations for support: NSERC, MITACS, IRIS, and the Canada Research Chairs.