Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model

Yoshua Bengio and Jean-Sébastien Senécal

Dept. IRO, Université de Montréal

C.P. 6128, Montreal, Qc, H3C 3J7, Canada

{*bengioy*}*@iro.umontreal.ca*

http://www.iro.umontreal.ca/∼lisa

September 24, 2007

Abstract

Previous work on statistical language modeling has shown that it is possible to train a feed-forward neural network to approximate probabilities over sequences of words, resulting in significant error reduction when compared to standard baseline models based on n-grams. However, training the neural network model with the maximum likelihood criterion requires computations proportional to the number of words in the vocabulary. We introduce adaptive importance sampling as a way to accelerate training of the model. The idea is to use an adaptive n-gram model to track the conditional distributions produced by the neural network. We show that a very significant speed-up can be obtained on standard problems.

1 Introduction

Statistical machine learning often involves a difficult compromise between computational costs and statistical qualities of the model. From a statistical perspective a major challenge is the high dimensionality of the data and the curse of dimensionality, and this is a very serious issue in statistical language modeling. In recent years, a particular class of neural-network based statistical language models has been proposed (Bengio, Ducharme, Vincent, & Jauvin, 2003) to help deal with high dimensionality and data sparsity, and it has been successfully applied and extended for speech recognition by smoothing n-gram language models (Schwenk & Gauvain, 2002; Schwenk, 2004) and stochastic grammars (Xu, Emami, & Jelinek, 2003). These models have a parametrization that remains compact when the size of the vocabulary increases and when the number of words of context increases. They rely on learning a continuous-valued representation for words and word subsequences that could help to efficiently represent context. Even though the computations required for training and probability prediction only scale linearly with vocabulary size and context size, these computations are much greater than those required with classical statistical language models such as n-grams (Manning & Schütze, 1999), because of the need to normalize conditional probabilities overall all the words in the vocabulary, at each prediction step. To make these neural network approaches more applicable, it is thus important to find ways to speed up these algorithms. This paper proposes a novel method to decrease training time, based on *adaptive importance sampling*, that has allowed speeding up training by a factor of 150 in the experiments reported here[1]. The idea of adaptive

[1] We reported a negligible change in perplexity with respect to the standard method

importance sampling is to train a model from which learning and sampling are very efficient (n-gram based) to track the neural network. During training of the neural network, when a word is presented in its context, instead of increasing its conditional likelihood and decreasing the conditional likelihood of all other words, it is sufficient to decrease the conditional likelihood of a few *negative examples*. These negative examples are sampled from the efficient n-gram based model that tracks the relative conditional probabilities of the neural network language model. Statistical language modeling focuses on trying to estimate the underlying distribution $P(w_1^T)$ that generated an observed sequence of words $w_1, \ldots, w_T$ (in this paper, we refer to the sub-sequence $w_i, \ldots, w_j$ as w_i^j, for simplicity).

The distribution can be represented by the conditional probability of the next word given all the previous ones:

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

In order to reduce the difficulty of the modeling problem, one usually compresses the information brought by the last words by considering only the last $n-1$ words, thus yielding the approximation

$$P(w_1^T) \approx \prod_{t=1}^{T} P(w_t | w_{t-n+1}^{t-1}). \tag{2}$$

The conditional probabilities $P(w_t | w_{t-n+1}^{t-1})$ can be easily modeled by considering sub-sequences of length n, usually referred to as *windows*, and computing the estimated joint probabilities $P(w_{t-n+1}^{t})$ and $P(w_{t-n+1}^{t-1})$; the model's conditional probabilities can then be computed as

$$P(w_t | w_{t-n+1}^{t-1}) = \frac{P(w_{t-n+1}^{t})}{P(w_{t-n+1}^{t-1})}. \tag{3}$$

Successful traditional approaches, called *n-grams* (Jelinek & Mercer, 1980; Katz, 1987; Kneser & Ney, 1995), are based on simple counting of frequency of appearance of the various windows of words in a training corpus, i.e. on the empirical conditional probabilities $P_n(w_t|w_{t-n+1}^{t-1}) = \frac{|w_{t-n+1}^t|}{|w_{t-n+1}^{t-1}|}$ where $|w_i^j|$ is just the frequency of subsequence w_i^j in the training corpus. The main problem with P_n is that it quickly overfits as n becomes large, and also quickly requires storing almost all of the data set in memory. In order to smooth P_n, it is usually combined with lower-order n-grams ($P_{n'}$ with $n' < n$) in order to redistribute some of the probability mass to sequences of n words that have never been seen in the data but whose n'-long suffix has been seen (Jelinek & Mercer, 1980; Katz, 1987).

1.1 Fighting the Curse of Dimensionality with Word Similarity

The problem faced by n-grams is just a special case of the *curse of dimensionality*. Word vocabularies being usually large, i.e. in the order of ten to hundred thousand words, modeling the joint distribution of, say, 10 consecutive words potentially requires 10^{40} to 10^{50} free parameters. Since these models do not take advantage of the similarity between words (but see class-based models, discussed and compared in (Bengio et al., 2003)), we believe that they tend to redistribute probability mass too blindly, mostly to sentences with a very low probability. This can be seen by looking at text generated by such models, which is often non-sensical except for short sentences.

A way to approach that problem, first proposed in Bengio, Ducharme, and Vincent (2001) (but see Bengio et al. (2003) for more details), and inspired by previous work on symbolic representation with neural networks, such as Hinton (1986) and Xu and Rudnicky (2000), is to map the words in vocabulary $\mathcal{V}$ into a *feature*

space $\mathbb{R}^m$ in which the notion of similarity between words corresponds to the Euclidean distance between their feature vectors. The learning algorithm finds a mapping $\mathbf{C}$ from the *discrete* set of words in $\mathcal{V}$ to a *continuous* semantic space[2]. In the proposed approach, this mapping is achieved by simply assigning a *feature vector* $\mathbf{C}_w \in \mathbb{R}^m$ to each word w of the vocabulary. The vectors are considered as parameters of the model and are thus learned during training, by gradient descent. The idea of exploiting a continuous representation for words was succesfully exploited in Bellegarda (1997) in the context of an n-gram based model. That of a vector-space representation for symbols in the context of neural networks was also used in terms of a parameter sharing layer (Riis & Krogh, 1996; Jensen & Riis, 2000).

1.2 An Energy-based Neural Network for Language Modeling

Many variants of this neural network language model exist, as presented in Bengio et al. (2003). Here we formalize a particular one, on which the proposed resampling method will be applied, but the same idea can be extended to other variants, such as those used in Schwenk and Gauvain (2002), Schwenk (2004), Xu et al. (2003). The probabilistic neural network architecture is illustrated in figure 1. This looks like a time-delay neural network (Lang & Hinton, 1988) since it has local shared weights (the matrix of feature vectors $\mathbf{C}$) in its first layer.

The output of the neural network depends on the next word w_t and the history h_t which is a shorthand notation for w_{t-n+1}^{t-1} as follows. In the *features layer*, one

[2]There has been some concern about whether the learned space is actually that continuous; in some way, it might still act as a discrete space if it is largely constituted of small neighborhoods within which variations don't affect the result at all. That question is still an open issue.

maps each word symbol w_{t-i} in string w_{t-n+1}^{t} to a d-dimensional vector $\mathbf{z}_i$ (with $d \ll |\mathcal{V}|$). In the framework considered here, the target (next) word is mapped to a different feature space than the context (last) words, i.e. a different set of feature vectors is used for the next word, whereas the context words share the same feature vector map [3]:

$$\begin{aligned} \mathbf{z}_i &= \mathbf{C}_{w_{t-i}}, \quad i = 1, \ldots, n-1, \\ \mathbf{z}_0 &= \mathbf{D}_{w_t}, \\ \mathbf{z} &= (\mathbf{z}_0, \ldots, \mathbf{z}_{n-1}) \end{aligned} \tag{4}$$

where $\mathbf{C}_j$ is the j-th column of the *word features* $d \times |\mathcal{V}|$ matrix $\mathbf{C}$ of free parameters for the context words and $\mathbf{D}_j$ is the j-th column of the word features $d \times |\mathcal{V}|$ matrix $\mathbf{D}$ for the next word w_t. The resulting dn-vector $\mathbf{z}$ (the concatenation of the n projections $\mathbf{z}_i$) is the input vector for the next layer, the hidden layer:

$$\mathbf{a} = \tanh(\mathbf{d} + \mathbf{W}\mathbf{z}) \tag{5}$$

where $\mathbf{d}$ is a vector of h free parameters (hidden units biases), $\mathbf{W}$ is a $h \times dn$ matrix of free parameters (hidden layer weights) and $\mathbf{a}$ is a vector of h hidden units activations. Finally the output is a scalar energy function

$$\mathcal{E}(w_t, h_t) = b_{w_t} + \mathbf{V}_{w_t} \cdot \mathbf{a} \tag{6}$$

where $\mathbf{b}$ is a vector of $|\mathcal{V}|$ free parameters (called biases), $\mathbf{V}$ (hidden to output layer weights) is a $h \times |\mathcal{V}|$ matrix of free parameters with one column $\mathbf{V}_i$ per word. Note that w_t and the history (context) $h_t = w_{t-n+1}^{t-1}$ are used in computing $\mathbf{a}$.

[3] We chose to use a different set of features for the next word in order to add more parameters to the model; preliminary experiments also showed that it yielded better results than sharing the feature vectors.

To obtain the joint probability of (w_t, h_t), we normalize the exponentiated energy $e^{-\mathcal{E}(w_t,h_t)}$ by dividing it by the normalizing function $Z = \sum_{(w,h)\in\mathcal{V}^n} e^{-\mathcal{E}(w,h)}$:

$$P(w_t, h_t) = \frac{e^{-\mathcal{E}(w_t,h_t)}}{Z}. \tag{7}$$

The normalization Z is extremely hard to compute, since it requires a number of passes (computations of $\mathcal{E}(\cdot)$) exponential in the number of context words. However, the conditional probability $P(w_t|h_t)$ is easier to normalize:

$$P(w_t|h_t) = \frac{e^{-\mathcal{E}(w_t,h_t)}}{Z(h_t)} \tag{8}$$

where $Z(h_t) = \sum_{w\in\mathcal{V}} e^{-\mathcal{E}(w,h_t)}$ is tractable, though still hard to compute because the number of vocabulary words $|\mathcal{V}|$ is usually large and the activation $\mathcal{E}(\cdot)$ requires many multiplications and additions (approximately the number of hidden units times $d+1$, for each value of w).

The above architecture can be seen as an *energy-based model* (Teh, Welling, Osindero, & Hinton, 2003):

$$P(X = x) = \frac{e^{-\mathcal{E}(x)}}{Z} \tag{9}$$

where $\mathcal{E}(\cdot)$ is a parametrized *energy* function which is low for plausible configurations of x, and high for improbable ones, and where $Z = \sum_{x\in\mathcal{X}} e^{-\mathcal{E}(x)}$ (or $Z = \int_{x\in\mathcal{X}} e^{-\mathcal{E}(x)}dx$ in the continuous case), is called the *partition function*. In the case that interests us, the partition function depends on the context h_t, as seen in (8). X will therefore represent a possible next-word, i.e. $\mathcal{X}$=$\mathcal{V}$, and probabilities are conditional on the context h_t.

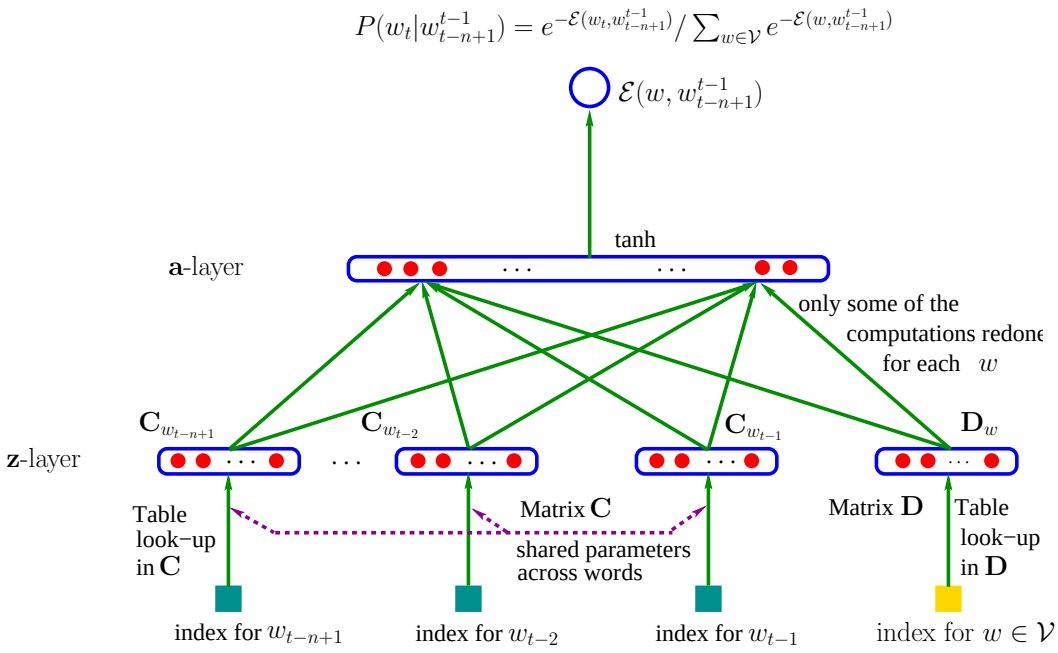


Figure 1: Architecture of the neural language model. The first layer has is linear, with local connections and temporally shared parameters (the matrix C and D whose columns are word features).

The main step in a gradient-based approach to train such models involves computing the gradient of the log-likelihood $\log P(X = x)$ with respect to parameters θ. The gradient can be decomposed in *two parts*: *positive reinforcement* for the observed value $X = x$ and *negative reinforcement* for every x', weighted by $P(X = x')$, as follows (by differentiating the negative logarithm of (9) with respect to θ):

$$\nabla_\theta \left(-\log P(x)\right) = \nabla_\theta \left(\mathcal{E}(x)\right) - \sum_{x' \in \mathcal{X}} P(x') \nabla_\theta \left(\mathcal{E}(x')\right). \tag{10}$$

Clearly, the difficulty here is to compute the negative reinforcement when $|\mathcal{X}|$ is large (as is the case in a language modeling application). However, as is easily seen as a fundamental property of exponential family models (Brown, 1986), the negative part of the gradient is nothing more than the average

$$E_P \left[\nabla_\theta \left(\mathcal{E}(X)\right)\right]. \tag{11}$$

In Hinton and Sejnowski (1986), it is proposed to estimate this average with a Gibbs sampling method, using a Markov Chain Monte-Carlo process. This technique relies on the particular form of the energy function in the case of the Boltzmann machine, which lends itself naturally to Gibbs sampling. The idea of applying sampling techniques to speed-up language models, e.g., of the exponential family is not new. See for example Rosenfeld, Chen, and Zhu (2001).

Note that the energy-based architecture and the products of experts formulation can be seen as extensions of the very successful **Maximum Entropy** models (Berger, Della Pietra, & Della Pietra, 1996), but where the basis functions (or "features", here the hidden units activations) are learned by penalized maximum likelihood at the same time as the convex combination parameters, instead of being learned in an outer loop, with greedy feature subset selection methods.

2 Approximation of the Log-Likelihood Gradient by Biased Importance Sampling

If one could sample from $P(\cdot)$, a simple way to estimate (11) would consist in sampling m points $x_1, \ldots, x_m$ from the network's distribution $P(\cdot)$ and to approximate (11) by the average

$$\frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \left(\mathcal{E}(x_i) \right). \tag{12}$$

This method, known as *classical Monte-Carlo*, yields Algorithm 1 for estimating the gradient of the log-likelihood (10). The maximum speed-up that could be achieved with such a procedure would be $|\mathcal{X}|/m$. In the case of the language modeling application we are considering, that means a potential for a huge speed-up, since $|\mathcal{X}|$ is typically in the tens of thousands and m could be quite small; in fact.

Algorithm 1 Classical Monte-Carlo Approximation of the Gradient

$\nabla_\theta \left(-\log P(x) \right) \leftarrow \nabla_\theta \left(\mathcal{E}(x) \right)$ {Add positive contribution}

for $k \leftarrow 1$ **to** m **do** {Estimate negative contribution}

$x' \sim P(\cdot)$ {Sample negative example}

$\nabla_\theta \left(-\log P(x) \right) \leftarrow \nabla_\theta \left(-\log P(x) \right) - \frac{1}{m} \nabla_\theta \left(\mathcal{E}(x') \right)$ {Add negative contribution}

end for

However, this method requires to sample from distribution $P(\cdot)$, which we can't do without having to compute $P(x)$ explicitly. That means we have to compute the partition function Z, which is still hard because we have to compute $\mathcal{E}(x)$ for each $x \in \mathcal{X}$.

Fortunately, in many applications, such as language modeling, we can use an

alternative, *proposal* distribution Q from which it is cheap to sample. In the case of language modeling, for instance, we can use n-gram models. There exist several Monte-Carlo algorithms that can take advantage of such a distribution to give an estimate of (11).

2.1 Classical Importance Sampling

One well-known statistical method that can make use of a proposal distribution Q in order to approximate the average $E_P[\nabla_\theta(\mathcal{E}(X))]$ is based on a simple observation. In the discrete case

$$E_P[\nabla_\theta(\mathcal{E}(x))] = \sum_{x\in\mathcal{X}} P(x)\nabla_\theta(\mathcal{E}(x)) = \sum_{x\in\mathcal{X}} Q(x)\frac{P(x)}{Q(x)}\nabla_\theta(\mathcal{E}(x)) = E_Q\left[\frac{P(X)}{Q(X)}\nabla_\theta(\mathcal{E}(X))\right]. \tag{13}$$

Thus, if we take m independent samples $y_1, \ldots, y_m$ from Q and apply classical Monte-Carlo to estimate $E_Q\left[\frac{P(X)}{Q(X)}\nabla_\theta(\mathcal{E}(X))\right]$, we obtain the following estimator known as *importance sampling* (Robert & Casella, 2000):

$$\frac{1}{m}\sum_{i=1}^{m}\frac{P(y_i)}{Q(y_i)}\nabla_\theta(\mathcal{E}(y_i)). \tag{14}$$

Clearly, that does not solve the problem: although we do not need to sample from P anymore, the $P(y_i)$'s still need to be computed, which cannot be done without explicitly computing the partition function. Back to square one.

2.2 Biased Importance Sampling

Fortunately, there is a way to estimate (11) without sampling from P nor having to compute the partition function. The proposed estimator is a biased version of classical importance sampling (Kong, Liu, & Wong, 1994). It can be used when $P(x)$ can be computed explicitly up to a multiplicative constant: in the case of

energy-based models, this is clearly the case since $P(x) = Z^{-1}e^{-\mathcal{E}(x)}$. The idea is to use $\frac{1}{W}w(y_i)$ to weight the $\nabla_\theta\left(\mathcal{E}(y_i)\right)$, with $w(x) = \frac{e^{-\mathcal{E}(x)}}{Q(x)}$ and $W = \sum_{j=1}^{m} w(y_i)$, thus yielding the estimator (Liu, 2001)

$$\frac{1}{W}\sum_{i=1}^{m} w(y_i)\nabla_\theta\left(\mathcal{E}(y_i)\right). \tag{15}$$

Though this estimator is biased, its bias decreases as m increases. It can be shown to converge to the true average (11) as $m \to \infty$ [4].

The advantage of using this estimator over classical importance sampling is that we need not compute the partition function: we just need to compute the energy function for the sampled points.

Algorithm 2 Biased Importance Sampling Approximation of the Gradient

$\nabla_\theta\left(-\log P(x)\right) \leftarrow \nabla_\theta\left(\mathcal{E}(x)\right)$ {Add positive contribution}

vector g $\leftarrow \mathbf{0}$

$W \leftarrow 0$

for $k \leftarrow 1$ **to** m **do** {Estimate negative contribution}

$y' \sim Q(\cdot)$ {Sample negative example}

$w \leftarrow \frac{e^{-\mathcal{E}(y')}}{Q(y')}$

$\mathbf{g} \leftarrow \mathbf{g} + w\nabla_\theta\left(\mathcal{E}(y')\right)$

$W \leftarrow W + w$

end for

$\nabla_\theta\left(-\log P(x)\right) \leftarrow \nabla_\theta\left(-\log P(x)\right) - \frac{1}{W}\mathbf{g}$ {Add negative contributions}

[4] However, this does not guarantee that the variance of the estimator remains bounded. We have not dealt with the problem yet, but maybe some insights could be found in Luis and Leslie (2000)

3 Adapting the Sample Size

Preliminary experiments with Algorithm 2 using the unigram distribution showed that whereas a small sample size was appropriate in the initial training epochs, a larger sample size was necessary later to avoid divergence (increases in training error). This may be explained by a too large bias – because the network's distribution diverges from that of the unigram, as training progresses – and/or by a too large variance in the gradient estimator.

In Bengio and Senécal (2003), we presented an improved version of Algorithm 2 that makes use of a diagnostic, called *effective sample size* (Kong, 1992; Kong et al., 1994). For a sample $y_1, \ldots, y_m \sim Q$, the effective sample size is:

$$ESS = \frac{(\sum_{j=1}^{m} w(y_j))^2}{\sum_{j=1}^{m} w^2(y_j)}. \tag{16}$$

Basically, this measure approximates the number of samples from the target distribution P that would have yielded, with classical Monte-Carlo, the same variance as the one yielded by the biased IS estimator with sample $y_1, \ldots, y_m$.

We can use this measure to diagnose whether we have sampled enough points. In order to do that, we fix a baseline sample size l. This baseline is the number of samples we would sample in a classical Monte-Carlo scheme, were we able to do it. We then sample points from Q by "blocks" of size $m_b \geq 1$ until the effective sample size becomes larger than the target l. If the number of samples becomes too large, we switch back to a full back-propagation (i.e. we compute the true negative gradient). This happens when the IS approximation is not accurate enough and we might as well use the exact gradient. This safeguarding condition ensures that convergence does not get slower than the one obtained when computing the exact gradient.

4 Adapting the Proposal Distribution

The method was used with a simple unigram proposal distribution to yield significant speed-up on the Brown corpus (Bengio & Senécal, 2003). However, the required number of samples was found to increase quite drastically as training progresses. This is because the unigram distribution stays fixed while the network's distribution changes over time and becomes more and more complex, thus diverging from the unigram.

An early idea we tried was to start with a unigram distribution and switching to an interpolated bigram, and then to an interpolated trigram during training. After each epoch, we compared the model's perplexity with that of, namely, the unigram, the bigram and the trigram. For example, once the model's perplexity would become lower than that of the interpolated bigram, we would switch to the interpolated bigram as our proposal distribution. Once the perplexity would become lower than the interpolated trigram's, we would switch to the interpolated trigram.

This procedure provided even poorer results than just using a simple unigram, requiring larger samples to get a good approximation. We think that this is due to the fact that, as was pointed out in Goodman (2003), the bigram and trigram have distributions that are much different from neural network language models. Bengio et al. (2003) also showed that learning a simple linear interpolation with a smoothed trigram helps achieve an even lower perplexity, confirming that the two

models give quite different distributions.

Clearly, using a proposal distribution that stays "close" to the target distribution would yield even greater speed-ups, as we would need less samples to approximate the gradient. We propose to use a n-gram model that is *adapted* during training to fit to the target (neural network) distribution P [5]. In order to do that, we propose to *redistribute the probability mass* of the sampled points in the n-gram to track P. This will be achieved with k-gram tables Q_k which will be estimated with the goal of matching the order k conditional probabilities of samples from our model P when the history context is sampled from the empirical distribution. The tables corresponding to different orders k will be interpolated in the usual way to form a predictive model, from which it is easy to sample. Hence we try to combine ease of sampling (Q_k is a k-gram model, which can be sampled from very quickly) with approximating of the model distribution (represented implicitly by the neural network).

define the *adaptive n-gram*:

$$Q(w_t|h_t) = \sum_{k=1}^{n} \alpha_k(h_t) Q_k(w_t|w_{t-k+1}^{t-1}) \tag{17}$$

where the Q_k are the sub-models that we wish to estimate, and $\alpha_k(h_t)$ is a mixture function such that $\sum_{k=1}^{n} \alpha_k(h_t) = 1$. Usually, for obvious reasons of memory constraints, the probabilities given by a n-gram will be non-null only for those sequences that are observed. Mixing with lower-order models allows to give some probability mass to unseen word sequences.

Let $\mathcal{W}$ be the set of m words sampled from Q. The number m is chosen by using the effective sample size approximation (see Section 3). Let $\bar{q}_k = \sum_{w \in \mathcal{W}} Q_k(w|w_{t-k+1}^{t-1})$

[5] A similar approach was proposed in Cheng and Druzdzel (2000) for Bayesian networks.

be the total probability mass of the sampled points in k-gram Q_k and $\bar{p} = \sum_{w \in \mathcal{W}} e^{-\mathcal{E}(w,h_t)}$ the unnormalized probability mass of these points in P. Let $\tilde{P}(w|h_t) = \frac{e^{-\mathcal{E}(w,h_t)}}{\bar{p}}$ for each $w \in \mathcal{W}$. For each k and for each $w \in \mathcal{W}$, the values in Q_k are updated as follows:

$$Q_k(w|w_{t-k+1}^{t-1}) \leftarrow (1-\lambda) Q_k(w|w_{t-k+1}^{t-1}) + \lambda \bar{q}_k \tilde{P}(w|h_t) \tag{18}$$

where λ is a kind of "learning rate" which needs to be set empirically: it adds an extra hyper-parameter to the whole system, which can be selected by comparing the evolution of perplexity on a small dataset for different values of λ. Note that

$$\sum_{w \in \mathcal{W}} \bar{q}_k \tilde{P}(w|h_t) = \bar{q}_k = \sum_{w \in \mathcal{W}} Q_k(w|w_{t-k+1}^{t-1})$$

so the update only redistributes probability mass within $\mathcal{W}$. Since the other probabilities in Q_k are not changed, this shows that $\sum_w Q_k(w|w_{t-k+1}^{t-1}) = 1$ after the update.

preserve the summation

Note that we would like to match the model probability $Q_k(w|w_{t-k+1}^{t-1})$ with the target probability $p_w = \frac{e^{-\mathcal{E}(w,h_t)}}{\sum_{w \in \mathcal{V}} e^{-\mathcal{E}(w,h_t)}}$, but we can only compute the conditional probabilities within the sample $\mathcal{W}$, so we try to match $\frac{Q_k(w|w_{t-k+1}^{t-1})}{\bar{q}_k}$ with $\frac{p_w}{\sum_{w \in \mathcal{W}} p_w} = \tilde{P}(w|h_t)$. The proposed update rule actually moves $\bar{q}_k$ around and tries to equate $Q_k(w|w_{t-k+1}^{t-1})$ with $\bar{q}_k \tilde{P}(w|h_t)$. More precisely, equation 18 can be seen as gradient descent in a quadratic criterion whose corresponding first order conditions give rise to conditional probabilities that are in average correct (conditional on $h_t = w_{t-n+1}^{t-1}$ and on the sampled set $\mathcal{W}$). This criterion is the expected value (over contexts h_t and sample sets $\mathcal{W}$) of $Q_k(w|w_{t-k+1}^{t-1})^2 - \bar{q}_k^2 \tilde{P}(w|h_t)$. The first-order condition is that the average over h_t and $\mathcal{W}$ of $Q_k(w|w_{t-k+1}^{t-1})$ equals the average of the renormalized target probability, $\bar{q}_k \tilde{P}(w|h_t)$. Performing a stochastic

gradient step with learning rate λ gives exactly rise to equation 18. Hence if the learning rate λ is reduced appropriately, one would get convergence to the global minimum of this criterion.

The parameters of functions $\alpha_k(\cdot)$ are updated so as to minimize the Kullback-Leibler divergence $\sum_{w \in \mathcal{W}} \tilde{P}(w|h_t) \log \frac{\tilde{P}(w|h_t)}{Q(w|h_t)}$ by gradient descent.

We describe here the method we used to train the α_k's in the case of a bigram interpolated with a unigram, i.e. $n = 2$ above. Rather than having the α_k's be a function of all possible histories h_t, we instead clustered the h_t's into equivalent classes and the α_k's were a function only of the class of h_t. Specifically, in our experiments, the α_k's were a function of the frequency of the last word w_{t-1}. The words were thus first clustered in C frequency bins $\mathcal{B}_c, c = 1, \ldots, C$. Those bins were built so as to *group words with similar frequencies in the same bin while keeping the bins balanced*[6]. (Algorithm 3)

Then , an "energy" value $a(c)$ was assigned for $c = 1, \ldots, C$. We set $\alpha_1(h_t) = \sigma(a(h_t))$ and $\alpha_2(h_t) = 1 - \alpha_1(h_t)$ where $\sigma(z) = 1/(1+e^{-z})$

and $a(h_t) = a(w_{t-1}) = a(c)$, c being the class (bin) of w_{t-1}. The energy $a(h_t)$ is thus updated with the following rule, trying again to match our target distribution $\tilde{P}(\cdot|h_t)$:

$$a(h_t) \leftarrow a(h_t) - \eta \alpha_1(h_t)\alpha_2(h_t) \sum_{w \in \mathcal{W}} \tilde{P}(w|h_t) \frac{Q(w|h_t)}{Q_2(w|w_{t-1}) - Q_1(w)} \quad (19)$$

where η is a learning rate.

[6] By *balanced*, we mean that the sum of word frequencies does not vary a lot between two bins. That is, let $|w|$ be the frequency of word w in the training set, then we wish that $\forall i, j, \sum_{w \in \mathcal{B}_i} |w| \approx \sum_{w \in \mathcal{B}_j} |w|$.

Algorithm 3 Building Balanced Frequency Bins

(1) $c \leftarrow 1$ index of bins

(2) $\mathcal{B}_1 \leftarrow \emptyset$

(3) $n_b \leftarrow 0$

(4) $\bar{n_b} \leftarrow \frac{1}{C} \sum_{w \in \mathcal{V}} |w|$ {Target frequency sum per bin}

(5) $\mathcal{V}' \leftarrow \mathcal{V}$

(6) **while** $\mathcal{V}' \neq \emptyset$ **do**

(7) $w_{max} \leftarrow \text{argmax}_{w \in \mathcal{V}'} |w|$ {w_{max} is the next maximum frequency word}

(8) $\mathcal{V}' \leftarrow \mathcal{V}' \setminus \{w_{max}\}$

(9) $\mathcal{B}_c \leftarrow \mathcal{B}_c \cup \{w_{max}\}$ Bins

(10) $n_b \leftarrow n_b + |w_{max}|$

(11) **if** $n_b > \bar{n_b}$ **then**

(12) $n_b \leftarrow 0$

(13) $c \leftarrow c + 1$ next bin

(14) **end if**

(15) **end while**

Algorithm 4 Adaptive Algorithm

(1) $\nabla_\theta(-\log P(x)) \leftarrow \nabla_\theta(\mathcal{E}(x))$ {Add positive contribution}

(2) **vector** $\mathbf{g} \leftarrow \mathbf{0}$, $\mathcal{W} \leftarrow \emptyset$

(3) $W \leftarrow 0, S \leftarrow 0$, $ESS \leftarrow 0$, $m \leftarrow m_b$

(4) **while** $ESS < n$ **and** $m \leq threshold$ **do**

(5) **for** $k \leftarrow 1$ **to** m_b **do** {Estimate negative contribution}

(6) $y' \sim Q(\cdot)$ {Sample negative example}

(7) $w \leftarrow \frac{e^{-\mathcal{E}(y')}}{Q(y')}$

(8) $\mathbf{g} \leftarrow \mathbf{g} + w\nabla_\theta(\mathcal{E}(y'))$

(9) $W \leftarrow W + w, S \leftarrow S + w^2$ to compute ESS

(10) $\mathcal{W} \leftarrow \mathcal{W} \bigcup \{y'\}$

(11) **end for**

(12) $ESS \leftarrow \frac{W^2}{S}$

(13) $m \leftarrow m + m_b$

(14) **end while**

(15) **if** $m > threshold$ **then**

(16) {Compute the gradient on all words}

(17) $\mathcal{W} \leftarrow \mathcal{V}$

(18) $\nabla_\theta(-\log P(x)) \leftarrow \nabla_\theta(-\log P(x)) - \sum_{x \in \mathcal{X}} P(x)\nabla_\theta(\mathcal{E}(x))$

(19) **else** biased IS

(20) $\nabla_\theta(-\log P(x)) \leftarrow \nabla_\theta(-\log P(x)) - \frac{1}{W}\mathbf{g}$ {Add negative contributions}

(21) **end if**

(22) {Update proposal distribution according to Eq. (18) and Eq. (19)}

(23) $updateProposal(Q, \mathcal{W}, \lambda, \eta)$

5 Experimental Results

We ran some experiments on the Brown corpus, with different configurations. The Brown corpus consists of 1,181,041 words from various American English documents. The corpus was divided in train (800K words), validation (200K words) and test (the remaining $\approx$ 180K words) sets. The vocabulary was truncated by mapping all "rare" words (words that appear 3 times or less in the corpus) into a single special word. The resulting vocabulary contains 14,847 words.

On this dataset, a simple interpolated trigram, serving as our baseline, achieves a perplexity of 253.8 on the test set. The weights of the interpolation are obtained by maximizing likelihood on the validation set. Better results can be achieved with a Kneser-Ney back-off trigram, but it has been shown in Bengio et al. (2003) that a neural network converges to a lower perplexity on Brown, and that the neural network can be interpolated with the trigram for even larger perplexity reductions. Kneser-Ney is simple but more sophisticated and better performing smoothing techniques have been proposed: see for example Rosenfeld (2000), Goodman (2001), Wang and Harper (2002).

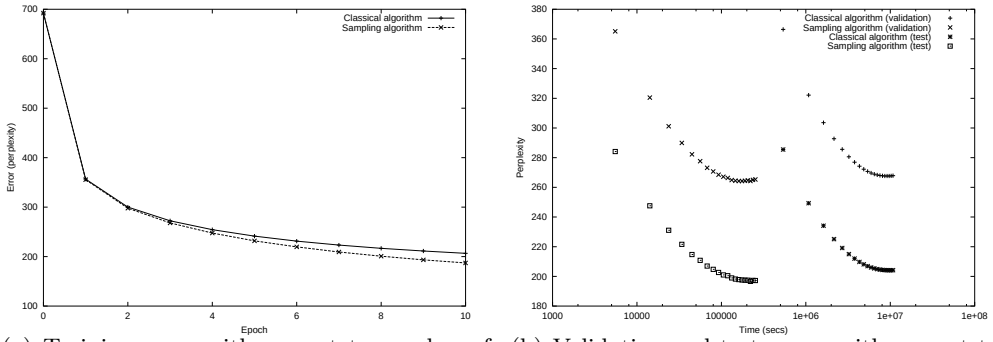
In all settings, we used 30 word features for both context and target words, and 80 hidden neurons. The number of context words was 3. The learning rate for the neural network was gradually decreased using $\frac{3\times10^{-3}}{1+10^{-8}t}$, with t the number of updates performed since the beginning of training. We used a weight decay of 10^{-4} to regularize the parameters. The output biases b_{w_t} in (6) were manually initialized so that the neural network's initial distribution is equal to the unigram (see Bengio et al. (2003) for details). This setting is the same as that of the neural network that achieved the best results on Brown, as described in Bengio

et al. (2003). In this setting, a classical neural network – one that doesn't make a sampling approximation of the gradient – converges to a perplexity of 204 in test, after 18 training epochs. In the adaptive bigram algorithm, the parameters λ in (18) and η in (19) were both set to $1e^{-3}$, as preliminary experiments had showed it to be a fair value for the model to converge. The $a(h_t)$ were initially set to $\sigma^{-1}(0.9)$ so that $\alpha_1(h_t) = 0.9$ and $\alpha_2(h_t) = 0.1$ for all h_t; this way, at the start of training, the target (neural network) and the proposal distribution are close to each other (both are close to the unigram).

Figure 2(a) plots the training error at every epoch for the network trained without sampling and a network trained by importance sampling, using an adaptive bigram with a target effective sample size of 50. The number of frequency bins used for the mixing variables was 10. It shows that the convergence of both networks is similar. The same holds for validation and test errors, as is shown in figure 2(b). In this figure, the errors are plotted with respect to computation time on a Pentium 4 2 GHz. As can be seen, the network trained with the sampling approximation converges before the network trained classically even completes one full epoch.

Quite interestingly, the network trained by sampling converges to an even lower perplexity than the ordinary one (trained with the exact gradient). After 9 epochs (26 hours), its perplexity over the test set is equivalent to that of the one trained with exact gradient at its overfitting point (18 epochs, 113 days). The sampling approximation thus allowed more than **100-fold speed-up** (104.3, more precisely). Both algorithms were implemented in C++ using shared code that attempted to be efficient.

Surprisingly enough, if we let the sampling-trained model converge, it starts to



(a) Training error with respect to number of epochs

(b) Validation and test errors with respect to CPU time

Figure 2: Comparison of errors between a model trained with the classical algorithm and a model trained by adaptive importance sampling.

overfit at epoch 18 – as for classical training – but with a lower test perplexity of 196.6, a 3.8% improvement. Total improvement in test perplexity with respect to the trigram baseline is 29%.

Apart from the speed-up, the other interesting point to note if we compare the results with those obtained by using a non-adaptive proposal distribution (Bengio & Senécal, 2003), which yielded a speed-up factor of 18, is that the mean number of samples required in order to ensure convergence seems to grow almost linearly with time (as shown in figure 3) whereas the required number of samples with the non-adaptive unigram was growing exponentially.

An important detail is worth mentioning here. Since $|\mathcal{V}|$ is large, we first thought that there was too small a chance to sample the same word w twice from the proposal Q at each step to really worry about it. However, we found out the chance of picking twice the same w to be quite high in practice (with an adaptive bigram proposal distribution). This may be simply due to Zipf's law concentrating mass on a few words. It may also be due to the particular form of our proposal

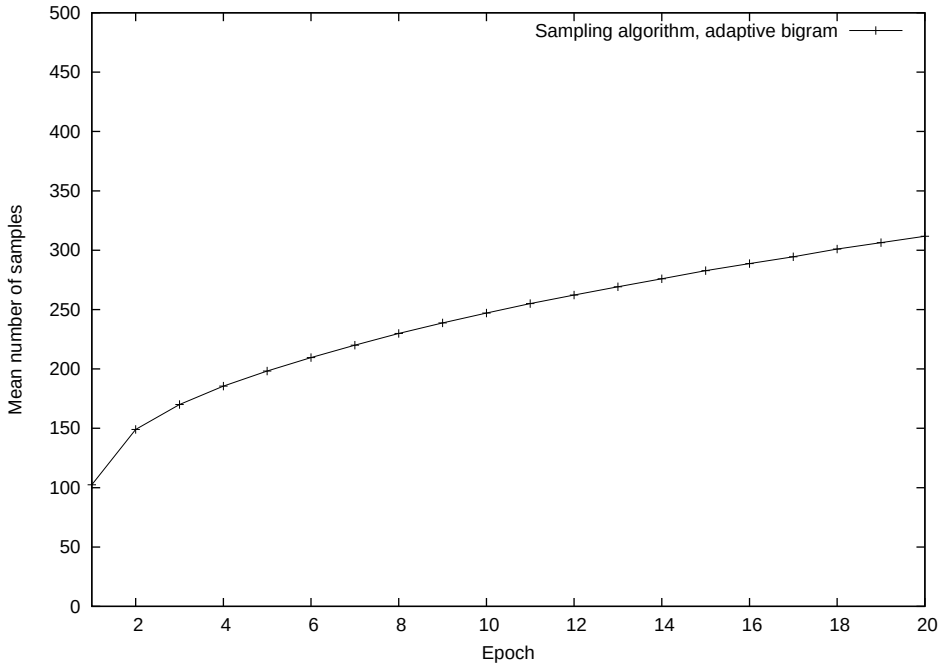


Figure 3: Comparison between the mean number of samples (for each stochastic gradient estimation) and the number of epochs shows an almost linear relation between them.

distribution (17). The bigram part $Q_2(w|w_{t-1})$ of that distribution being non-null for only those words w for which $|w_{t-1}w| > 0$, there are contexts w_{t-1} in which the number of candidate words w for which $Q_2(w|w_{t-1}) > 0$ is small, thus there are actually good chances to pick twice the same word. Knowing this, one can save much computation time by avoiding to compute the energy function $\mathcal{E}(w, h_t)$ many times for the same word w. Instead, the values of the energy functions for sampled words is kept in memory. When a word is first picked, its energy $\mathcal{E}(w, h_t)$ is computed in order to calculate the sampling weights (see Algorithm 4). The value of the sampling weight is kept in memory so that, whenever the same word is picked during the same iteration, all that needs to be done is to use the copied weight, thus saving one full propagation of the energy function. **This trick increases the speed-up from a factor of 100 to a factor of 150**.

Since the adaptive bigram is, supposedly, a close approximation of the neural network's distribution, we thought of evaluating it on the test corpus. Clearly, if the bigram was close enough, it would yield a comfortable perplexity, with the advantage of being a lot quicker. However, experiments showed a larger perplexity than a simple interpolated trigram.

6 Future Work

Previous work (Bengio et al., 2003) used a parallel implementation in order to speed-up training and testing. Although our sampling algorithm works very well on a single machine, we had much trouble making an efficient parallel implementation of it. The reason is that the parallelization has to be done on the hidden layer; thus for each back-propagation, we have to accumulate the gradient with respect

to the feature parameters (the $\mathbf{z}_i$'s) for each processor and then share the gradients. The process of sharing the gradient necessitates huge resources in terms of data transmission, which we have found to take up to 60% of the back-propagation time. One way to deal with the problem is to desynchronize the sharing of the parameters on the feature vectors i.e. allowing the computations to continue while the messages are transmitted. Since the changes in the feature vectors are quite small, this should affect convergence only slightly.

The other problem we face is that of choosing the target effective sample size. Currently, we have to choose it conservatively enough to guarantee convergence. In fact, we could achieve the same convergence by adapting it to the gradient's variance: as training progresses, the gradient is likely to become noisier, thus necessitating a greater number of samples for even the classical Monte-Carlo estimate to yield a good approximation. We could thus save even more computations by targeting a smaller effective sample size at the start of training and increasing it afterwards.

Finally, although accelerating the gradient's calculation is an important matter, an accelerated method for estimating the probabilities is left unaddressed by our work. Interesting methods that use the network only for the most frequent words have yielded impressive results (Schwenk & Gauvain, 2002). These authors rely mostly on the idea of restricting the vocabulary predicted by the neural network in order to reduce computation time, and that technique yields savings both during training and testing. On the other hand, what we propose is here only helps during training. If one wants to compute the actual conditional probabilities of the next word given the context using the neural network model, one still has to

compute the sum over all the words in the vocabulary. Other techniques such as those discussed in Schwenk and Gauvain (2002), Schwenk (2004) may be used to speed-up computation during testing.

One issue that should be looked at are the factors that would make the proposed adaptive importance sampling method work better or worse. Based on the experiments we have performed, an important factor is that the approximation of the target model by the n-gram used for sampling should be sufficiently good, otherwise the importance sampling becomes inefficient (that explains why most of the speed-up is obtained initially when the neural network has a model that is not too different from what an n-gram can capture). For this reason, exploring higher order n-grams for the proposal distribution might become necessary for more modeling more complex data. In particular, experiments on much larger datasets are required to evaluate how this method scales.

7 Conclusion

In this paper, we describe a method to efficiently train a probabilistic energy-based neural network. Though the application was to language modeling with a neural network, the method could in fact be used to train arbitrary energy-based models as well.

The method is based on the observation that the gradient of the log-likelihood can be decomposed in two parts: positive and negative contributions. The negative contribution is usually hard to compute because it involves a number of passes through the network equivalent to the size of the vocabulary. Luckily, it can be estimated efficiently by importance sampling.

We had already argued for such a method in Bengio and Senécal (2003), achieving a significant 19-fold speed-up on a standard problem (Brown). Here we show that an even greater speed-up can be obtained by *adapting* the proposal distribution as training progresses so that it stays as close as possible to the network's distribution. We have found that it is possible to do it efficiently by *reusing the sampled words* to re-weight the probabilities given by a n-gram. With the new method, we were able to achieve a 150-fold speed-up on the same problem, with a negligible change in perplexity. Analysis of the required sample size through time also suggests that the algorithm will scale with more difficult problems, since the mean sample size remains approximately proportional to the number of epochs.

Acknowledgements

The authors would like to thank Geoffrey Hinton for fruitful discussions, and the following funding organizations: NSERC, MITACS, and the Canada Research Chairs.

References

Bellegarda, J. (1997). A latent semantic analysis framework for large–span language modeling. In *Proceedings of Eurospeech 97*, pp. 1451–1454 Rhodes, Greece.

Bengio, Y. (2002). New distributed probabilistic language models. Tech. rep. 1215, Dept. IRO, Université de Montréal.

Bengio, Y., Ducharme, R., & Vincent, P. (2001). A neural probabilistic language model. In Leen, T. K., Dietterich, T. G., & Tresp, V. (Eds.), *Advances in*

Neural Information Processing Systems 13, pp. 932–938. MIT Press.

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of Machine Learning Research.

Bengio, Y., & Senécal, J.-S. (2003). Quick training of probabilistic neural nets by sampling. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, Vol. 9 Key West, Florida. AI and Statistics.

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

Brown, L. D. (1986). *Fundamentals of Statistical Exponential Families*, Vol. 9. Inst. of Math. Statist. Lecture Notes Monograph Series.

Cheng, J., & Druzdzel, M. J. (2000). Ais-bn: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks. *Journal of Artificial Intelligence Research*, *13*, 155–188.

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

Goodman, J. (2003). A bit of progress in language modeling – extended version. Tech. rep. MSR-TR-2001-72, Machine Learning and Applied Statistics Group, Microsoft Research.

Hinton, G. E., & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E., & McClelland, J. L. (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. MIT Press, Cambridge, MA.

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

Hinton, G. (1999). Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN99)*, pp. 1–6 Edin burgh, Scotland.

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

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

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.

Kneser, R., & Ney, H. (1995). Improved backing-off for m-gram language modeling. In *International Conference on Acoustics, Speech and Signal Processing*, pp. 181–184.

Kong, A. (1992). A note on importance sampling using standardized weights. Tech. rep. 348, Department of Statistics, University of Chicago.

Kong, A., Liu, J. S., & Wong, W. H. (1994). Sequential imputations and bayesian missing data problems. *Journal of the American Statistical Association*, *89*, 278–288.

Lang, K. J., & Hinton, G. E (1988). The development of the time-delay neural network architecture for speech recognition. Tech. rep. CMU-CS-88-152, Carnegie-Mellon University.

Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*. Springer.

Luis, O., & Leslie, K. (2000). Adaptive importance sampling for estimation in structured domains. In *Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pp. 446–454.

Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press.

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

Robert, C. P., & Casella, G. (2000). *Monte Carlo Statistical Methods*. Springer. Springer texts in statistics.

Rosenfeld, R. (2000). Two decades of statistical language modeling: Where do we go from here?. *Proceedings of the IEEE*, *88*(8).

Rosenfeld, R., Chen, S. F., & Zhu, X. (2001). Whole-sentence exponential language models: A vehicle for linguistic-statistical integration. *Computers Speech and Language*, *15*(1).

Schwenk, H., & Gauvain, J.-L. (2002). Connectionist language modeling for large vocabulary continuous speech recognition. In *International Conference on Acoustics, Speech and Signal Processing*, pp. 765–768 Orlando, Florida.

Schwenk, H. (2004). Efficient training of large neural networks for language modeling. In *IEEE International Joint Conference on Neural Networks (IJCNN)*. to appear July 2004.

Schwenk, H., & Gauvain, J.-L. (2002). Connectionist language modeling for large vocabulary continuous speech recognition. In *Proceedings of ICASSP*, pp. 765–768 Orlando.

Teh, Y.-W., Welling, M., Osindero, S., & Hinton, G. E. (2003). Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, *4*, 1235–1260.

Wang, W., & Harper, M. P. (2002). The superARV language model: investigating the effectiveness of tightly integrating multiple knowledge sources. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pp. 238–247 Morristown, NJ, USA. Association for Computational Linguistics.

Xu, P., Emami, A., & Jelinek, F. (2003). Training connectionist models for the structured language model. In *Empirical Methods in Natural Language Processing, EMNLP'2003*.

Xu, W., & Rudnicky, A. (2000). Can artificial neural network learn language models. In *International Conference on Statistical Language Processing*, pp. M1–13 Beijing, China.