Quick Training of Probabilistic Neural Nets by Importance Sampling

Yoshua Bengio and **Jean-Sébastien Senécal**
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal

Abstract

Our previous work on statistical language modeling introduced the use of probabilistic feedforward neural networks to help dealing with the curse of dimensionality. Training this model by maximum likelihood however requires for each example to perform as many network passes as there are words in the vocabulary. Inspired by the contrastive divergence model, we propose and evaluate sampling-based methods which require network passes only for the observed "positive example" and a few sampled negative example words. A very significant speed-up is obtained with an adaptive importance sampling.

1 Introduction

In our previous work (Bengio, Ducharme and Vincent, 2002), we have shown that the probability of word sequences $w_1, w_2, \ldots w_T$ can be efficiently represented by various artificial neural network architectures, where *efficiently* refers here to the statistical sense, meaning that these models generalize well and have low *perplexity* (the perplexity is the exponential of the negative average log-likelihood). However they are computationally much more expensive than *n-grams*, 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 $\mathbf{R^{30}}$). 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 descent on the logarithm of the perplexity). 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.

1.1 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 sentences obviously *blows up exponentially with the number of words*. This can be avoided with drastic conditional independence assumptions, but these prevent us from capturing some dependencies that we know to matter. This well-known problem plagues n-grams (Katz, 1987; Jelinek and Mercer, 1980), lexicalized stochastic grammars (Charniak, 1999; Collins, 1999; Chelba and Jelinek, 2000), 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 analyzes, 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 prevents any observation sequence from getting a tiny probability, but this is a achieved at a high price, by distributing probability mass over vast volumes of data space.

1.2 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 (c.f. (Hinton, 1986)). The hid-

net: w -> z -> a

den units of an artificial neural network encode information in a very efficient way; as an example, 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 all of them. For binary inputs, a polynomial estimator is therefore essentially equivalent to a table of all variable values combinations. In general, for a polynomial model to capture all dependencies of order d (i.e., involving d-tuples of variables) would require $O(m^d)$ parameters. This is a particular case of the curse of dimensionality. Things are obviously much worse with natural 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 H of hidden units can capture the H most important high-order dependencies between the inputs. The number of free parameters is only $O(Hm)$, yet dependencies of any order (between any number of input variables) can be captured. The **drawback** is that the estimation of the parameters is much more difficult and requires **more computation** because the objective function is not convex and can in fact be quite complex. These ideas have been exploited to learn the probability function of high-dimensional discrete data in (Bengio and Bengio, 2000), where comparisons have been made with polynomial learners and table-based graphical models.

In terms of representation, distributed models (e.g. neural networks, distributed representation graphical models (Saul and Jordan, 1996), or Maximum Entropy models (Berger, Della Pietra and Della Pietra, 1996)) can be exponentially more efficient than "localist" models, allowing to consider many higher-order dependencies. However, they are more difficult to optimize, because analytic or quickly converging algorithms like EM are not applicable, and they may involve expensive computations, like the computation of the partition function in Maximum Entropy models.

1.3 Neural Architecture for Representing High-Dimensional Distributions

The neural network already described in (Bengio, Ducharme and Vincent, 2002) has the basic architecture shown in Figure 1.

Many variants are possible, but we formalize one in particular below. 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 each word w_{t-i} in $(w_t, w_{t-1}, w_{t-2}, \ldots, w_{t-n})$ to a lower-dimensional continuous subspace z_i:

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

where C_j is the j-th column of the **word features** matrix of free parameters. The resulting vector z (the concatenation of the projections z_i) 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 (hidden layer weights) and a is a vector of hidden units activations. Finally the output is a scalar energy function

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

where b is a vector of free parameters (called biases), and U (direct input to output layer weights) and V (hidden to output layer weights) are matrices of free parameters with one column U_i or V_i per word.

To obtain conditional probabilities, we normalize the exponentiated energies:

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

The above neural architecture can be viewed as a special case of *energy-based probability models* of the form

$$P(Y = y) = \frac{e^{-\mathcal{E}(y)}}{\sum_{y'} e^{-\mathcal{E}(y')}} \tag{3}$$

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, denoted Z below). In general, the partition function is extremely expensive to compute (because it may involve an exponential number of terms). However, in the case of the above language model, when the model is used to compute the conditional probability of the next word, the computation of the partition function is feasible (only grows linearly with the vocabulary size) but is still quite expensive.

$$P(w_t = i \mid context) = e^{-\mathcal{E}(i,w_{t-1},\ldots,w_{t-n})} / \sum_j e^{-\mathcal{E}(j,w_{t-1},\ldots,w_{t-n})}$$
$$= \text{softmax}(-\mathcal{E}(.,w_{t-1},\ldots,w_{t-n}))$$

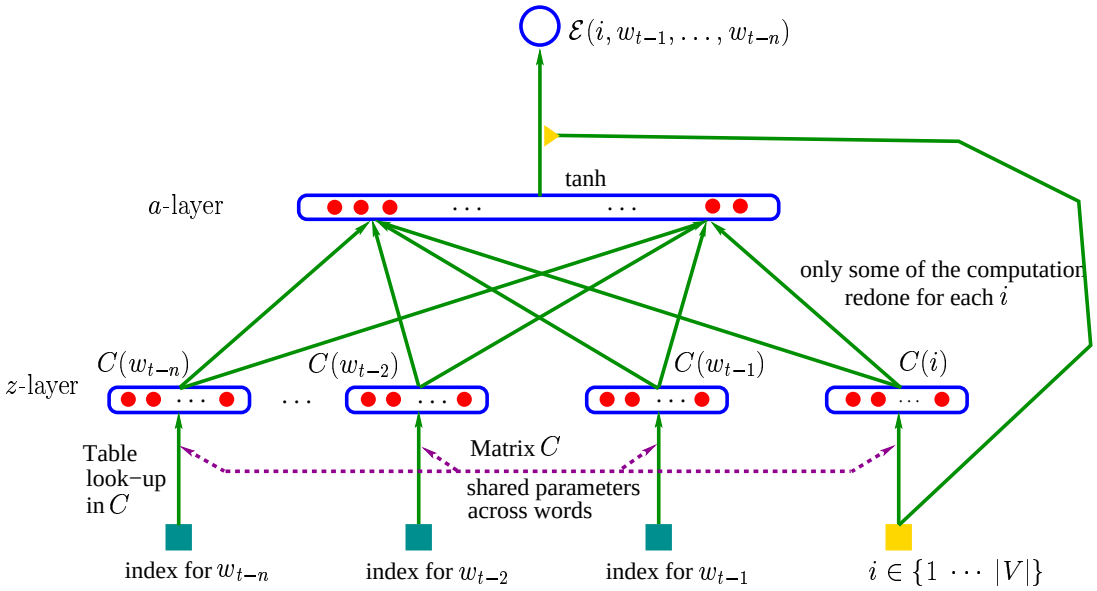

Figure 1: Architecture of the neural language model. The first layer has is linear, with local connections and temporally shared parameters (the matrix C whose columns are word features).

Hinton has proposed the **contrastive divergence** (Hinton, 2002) approach to approximate the gradient of the log-likelihood with respect to these parameters, in the unsupervised case, to make learning computationally feasible. 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* $Y = y$ (the observed value) and *negative reinforcement for every* y'*, weighted by* $P(Y = y')$, as follows (by differentiating equation 3):

$$\frac{\partial \log P(y)}{\partial \theta} = -\frac{\partial \mathcal{E}(y)}{\partial \theta} + \sum_{y'} P(y') \frac{\partial \mathcal{E}(y')}{\partial \theta} \qquad (4)$$

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 (Hinton, 2002)). The chain is started at y so that it would converge to the stationary intractable distribution $P(y')$. Unfortunately, this technique relies on the particular form of the energy function in the case of products of experts, which lends itself naturally to Gibbs sampling (using the activities of the hidden units as one of the random variables, and the network input as the other one, see equation 5 below).

Based on the premise that we have at our disposal a good proposal distribution Q (an approximation of P) from which it is easy to sample, we can take advantage of sampling to approximate the gradient. In this paper, we propose several new variants of that key idea.

1.4 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:

$$P(y) = \frac{\prod_i P_i(y)}{Z} \qquad (5)$$

where $P_i(y)$ are individual expert models, and Z is 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" of the form

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

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 out in different instances, and they suggest that products of experts can yield significant improvements in terms of out-of-sample likelihood. For example, Foster's experiments (Foster, 2002) confront head-to-head a normalized product of probabilities (implemented by a Maximum Entropy model) with a weighted sum of probabilities. In this case the application is to statistical translation; one would like to estimate

$$P(\textit{next translated word}|\textit{previous translated words}, \textit{source sentence})$$

in the process of building

$$P(\textit{translated sentence}|\textit{source sentence}).$$

With an additive approach, one builds a mixture of $P(\textit{next translated word}|\textit{previous translated words})$ and $P(\textit{next translated word}|\textit{source sentence})$. 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 **conjunction**: 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 convincing set of comparative results is provided by Charniak'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 – a mixture – and a Maximum Entropy scheme – a product – in order to combine several sources of information (conditioning events to predict the rule probability). The performance measure is 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 substantial with respect to the 12% error mark.

Why do we observe such improvements? We conjecture that in high-dimensional spaces, a mixture is generally too wasteful of the 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 the probability mass**. If each expert only has part of the information (e.g. looking only at certain aspects or certain variables), then it is more sensible 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 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 (there are a few computations that do not need to be re-done, corresponding to the part of Wz that depends only on h_t, see equations 1 and 2).

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

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

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

$$\frac{\partial \log P(w_t|h_t)}{\partial \theta} = -\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta} + \sum_{w'} P(w'|h_t) \frac{\partial \mathcal{E}(w',h_t)}{\partial \theta} \quad (6)$$

Because of the large number of words M in the vocabulary, there clearly is a huge potential for greater computational speed 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, might yield training convergence about as fast (noting that the instantaneous 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 thus the following: for each training example (w_t, h_t), run Algorithm 1 with N samples, where w

Algorithm 1 Monte-Carlo approximation of gradient

Add positive contribution: $\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta}$
{ N is the number of random samples }
for $k \leftarrow 1$ **to** N **do**
  Sample negative example $w' \sim P(\cdot|h_t)$ (*)
  Add negative contribution: $-\frac{1}{N}\frac{\partial \mathcal{E}(w',h_t)}{\partial \theta}$
Update parameters θ from sum of above contributions

The maximum speed-up of the above 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, and (Hinton, 2002) found $N = 1$ to work well for the contrastive divergence method, this is appealing. Unfortunately, we can't actually perform a (cheap) ordinary Monte-Carlo approximation of $\sum_{w'} P(w'|h_t)\frac{\partial \mathcal{E}(w',h_t)}{\partial \theta}$ because we don't know how to cheaply sample from $P(w'|h_t)$ (step (*) above): doing it exactly 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, as long as $P > 0 \Rightarrow Q > 0$. P << Q

2.1 Independent Metropolis-Hastings

A Monte-Carlo Markov Chain (MCMC) converging to $P(\cdot|h_t)$ 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

Add positive contribution: $\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta}$
$w \leftarrow w_t$
$k \leftarrow 0$
while $k < N_c + N_s$ **do**
  Sample $w' \sim Q(\cdot|h_t)$
  $r \leftarrow \min(1, \frac{e^{-\mathcal{E}(w',h_t)}}{e^{-\mathcal{E}(w,h_t)}}\frac{Q(w|h_t)}{Q(w'|h_t)})$
  with probability r **do**
    $k \leftarrow k + 1$
    $w \leftarrow w'$
    if $k \geq N_c$ **then**
      Add negative contribution: $-\frac{1}{N_s}\frac{\partial \mathcal{E}(w,h_t)}{\partial \theta}$

The first N_c samples are used to get the Markov chain to converge close enough to P, while the last N_s samples are used to form the gradient average. Note that When we choose N_c small (e.g. 0), we do not 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 much 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 be closer in spirit to the contrastive divergence algorithm. But the way to efficiently perform such a sample is not clear yet.

2.2 IS

It is used when one cannot sample from P but has a proposal distribution Q. Whereas the ordinary Monte-Carlo estimator of $E_P[g(Y)] = \sum_{y'} P(y')g(y')$ is

$$\frac{1}{N}\sum_{y' \sim P} g(y')$$

IS ==>

$$\frac{1}{N}\sum_{y' \sim Q} g(y')\frac{P(y')}{Q(y')}.$$

It is easy to show that the above is an unbiased estimator of $E_P[g(Y)]$. Strictly speaking, we cannot use ordinary unbiased importance sampling since we don't want to compute $P(y')$ 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.

Z is itself an average (with uniform distribution $1/M$):

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

IS ==>

$$\hat{Z}(h_t) = \frac{M}{N}\sum_{w' \sim Q(\cdot|h_t)} \frac{e^{-\mathcal{E}(w',h_t)}}{MQ(w'|h_t)}$$

Fact. $\sum_{w:W} F(w) \approx 1/N \sum_{w \sim q} F(w)/q(w)$, $q(w)>0$

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

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

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

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

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

Note that since $E[A/B] \neq E[A]/E[B]$, this is a biased estimator (the more so for small N). [1] The procedure is summarized in Algorithm 3.

Algorithm 3 Importance Sampling Gradient Approximation

Add positive contribution: $\frac{\partial \mathcal{E}(w_t,h_t)}{\partial \theta}$
vector $a \leftarrow \mathbf{0}$
$b \leftarrow 0$
repeat N **times**
  Sample $w' \sim Q(\cdot|h_t)$
  $r \leftarrow \frac{e^{-\mathcal{E}(w',h_t)}}{Q(w'|h_t)}$
  $a \leftarrow a + r \frac{\partial \mathcal{E}(w',h_t)}{\partial \theta}$
  $b \leftarrow b + r$
Add negative contribution: $-\frac{a}{b}$

3 Adapting the Sample Size

Our preliminary experiments with Algorithm 3 showed that whereas a small sample size was appropriate in the initial training epochs, a larger sample size was necessary later on to avoid divergence (increases in perplexity). This is probably because more precision is required in the gradient as training progresses, although we have yet to determine if this is due to a too large bias or a too large variance of the gradient estimator (increasing the sample size decreases both).

This has led to an improved algorithm (Algorithm 4), which is the one that gave us the result reported in the next section, and in which the number of samples is gradually increased according to a diagnostic. It uses Algorithm 3 in the inner loop.

[1] However, the fact that we use the same sample for estimating both the numerator and the denominator certainly does help reduce the bias, because the value of each $e^{-\mathcal{E}(w',h_t)}/Q(w'|h_t)$ will affect both of them in a similar way (i.e. small values will reduce both the numerator and the denominator while large values will increase both).

Algorithm 4 Adaptive Sample Size Algorithm

{ N_0 is the starting number of random samples }
$N \leftarrow N_0$
Compute exact perplexity C_{last} on a tiny data set
$e \leftarrow 1$
while $e \leq n$ **do** { n is the number of epochs }
  $\theta_{last} \leftarrow \theta$ { Save parameters }
  foreach training pair (w_t, h_t) **do**
    Compute the gradient estimator
      using Algorithm 3 with N samples
      and update parameters θ accordingly
  Compute exact perplexity C on the tiny data set
  if $C > C_{last}$ **then**
    $N \leftarrow 2N$ { Increase the number of samples }
    $\theta \leftarrow \theta_{last}$ { Revert to last epoch's version }
  else
    $C_{last} \leftarrow C$
    $e \leftarrow e + 1$

We are currently experimenting with a variant of Algorithm 4 in which the diagnostic does not require the exact computation of the perplexity on a tiny data set. Instead one measures the "effective sample size" S of the importance sampling estimator:

$$S = \frac{(\sum_{j=1}^{N} r_j)^2}{\sum_{j=1}^{N} r_j^2}$$

where r_j is the importance sampling ratio for the j-th sample, $r_j = P(w'_j)/Q(w'_j)$, which is estimated here with $r_j \approx \frac{e^{-\mathcal{E}(w'_j,h_t)}/Q(w'_j|h_t)}{\sum_{w' \in \mathcal{J}} e^{-\mathcal{E}(w',h_t)}/Q(w'|h_t)}$, where $\mathcal{J}$ is the set of N words sampled from $Q(\cdot|h_t)$ and w'_j is the j-th sampled word in $\mathcal{J}$. The idea is to choose N so as to make sure that the effective sample size is always greater than a minimum value N_0.

4 Experimental Results

4.1 Metropolis-Hastings

Experiments with the Metropolis-Hastings algorithm (Algorithm 2) have yielded very poor results. At first, the training perplexity decreases; but as soon as the network's distribution starts to diverge too much from the proposal, perplexity increases. Sampling more to wait for convergence (i.e. increasing N_c) does not seem to help, at least for reasonable values of N_c (e.g. 100, 500, 1000), nor does increasing N_s. Sampling from another proposal distribution (e.g. interpolated trigram instead of unigram) yielded even worse results.

We believe that the explanation lies in the MCMC being too far from convergence, but we have yet to verify this experimentally. The results of these experiments also suggested a very strong sensitivity to the proposal distribution.

4.2 Is

The importance sampling algorithm has been much more successful. We provide results of an experiment in which we compare a model trained with the exact and complete gradient (i.e. iterating through all the M words in the vocabulary) and a model trained with importance sampling (Algorithm 4).

Both models were trained on the Brown corpus, with the same data preprocessing as in (Bengio, Ducharme and Vincent, 2002). The most important step is the merging of rare words into a single token, to yield $M = 16383$ words in the vocabulary. We used the same 800,000 training examples and the test set of 181,041 examples (the validation set was not used to perform early stopping, instead a fixed number of iterations were run).

The two models had 80 hidden units (dimension of vector a of hidden unit activations in equation 2), 30 features per word (number of rows of matrix C), and the output weights V are controlled by the target word w_t. On the other hand the U matrix (direct connections from features layer to output layer) was not used (i.e. set to 0), since it did not improve performance (see the results on various architectures described in (Bengio, Ducharme and Vincent, 2002)). The number of words of context was $n = 3$. For Algorithm 4 the initial sampling size is $N_0 = 100$. The unigram is used proposal distribution. Preliminary experiments with the higher order n-grams did not help, nor did adaptive mixtures of unigram and interpolated trigram (so as to match the neural network's perplexity).

The tiny data set used in Algorithm 4 to check perplexity was a set of 1000 examples randomly taken from the training set. This estimated training perplexity was recalculated after each batch of 200,000 examples.

We ran both algorithms for 8 full epochs (full passes over all of the 800,000 examples of the training set). The train and test perplexities were approximately the same for both models: about 214 on the training set and 278 on the test set.

However, we achieved a **15-fold speed-up** with importance sampling, bringing the training time from several days down to a few hours. Note that the deleted interpolation trigram gives a perplexity of 336 on that same test data, and that further significant improvements in perplexity can be obtained by simple averaging of the trigram and the neural network (e.g. down to 265 in (Bengio, Ducharme and Vincent, 2002)).

Only one problem remains: the minimum sampling size needed to avoid divergence starts very small (100) and rapidly increases. At the end of training, we need to sample over 3000 examples. Future work should determine if this algorithm scales to much larger training sets.

5 Future Work

Better results (i.e. requiring a smaller sampling size) might be obtained with a more appropriate proposal distribution. Surprisingly, the unigram worked much better than the interpolated bigram or interpolated trigram as a proposal distribution in our experiments. At the beginning of training, it is natural to sample from the unigram, the network's distribution being very close to it; however, contrary to our expectations, we discovered that switching to a bigram or trigram later, even smoothly, actually worsens the training, requiring a much larger sampling size than for the unigram to lower the perplexity.

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 toward the minimum variance estimator.

In the discrete case, it is well known that the minimum variance proposal distribution for importance sampling, trying to estimate the scalar $E_P[g(Y)]$, is

$$\frac{P(y)|g(y)|}{\sum_{y'} P(y')|g(y')|}$$

However, we are averaging 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

$$\frac{P(y)\|g(y)\|}{\sum_{y'} P(y')\|g(y')\|}.$$

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 for the constraint $\sum_{y'} Q(y') = 1$, and solving for the zero derivative with respect to $Q(y)$.

Let us therefore consider how we could approximate a minimum variance proposal distribution, e.g. 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 $\mathcal{W} = \{\text{sampled negative examples at } t\} \cup \{w_t\}$, where context $h_t = (w_{t-1}, w_{t-2}, \ldots)$. Let $\bar{b} = \sum_{w' \in \mathcal{W}} B(w'|w_{t-1})$ be the total B mass of those words and $\bar{y} = \sum_{w' \in \mathcal{W}} e^{-\mathcal{E}(w', h_t)}$ their neural network total unweighted probability. We propose the following *adaptive reweighting of the bigram probabilities* $B(w'|w_{t-1})$ to track $P(w'|w_{t-1})$, as follows:

$$B(w'|w_t) \leftarrow \alpha B(w'|w_t) + (1-\alpha) \left\| \frac{\partial \mathcal{E}(w', h_t)}{\partial \theta} \right\| \frac{e^{-\mathcal{E}(w', h_t)} \bar{b}}{\bar{y}}$$

doing this $\forall w' \in \mathcal{W}$. The idea is to redistribute probability mass between the sampled words so that for B their relative probability within $\mathcal{W}$ agrees with the ideal proposal distribution, i.e. $\frac{B(i|w_t)}{b} \approx \frac{e^{-\mathcal{E}(i,h_t)}}{\bar{y}}$. Here α plays the role of a "learning rate" to average over different h_t sharing the same w_{t-1}, and to keep adapting B as parameters change.

For saving on storage, the entries for very rare (w_t, w_{t-1}) pairs (e.g. not appearing in the data) might not be stored in B. For these one could just back-off to the unigram as the proposal distribution.

6 Conclusion

This paper has argued for distributed representations to model high-dimensional dependencies in statistical language modeling, and presented a new learning technique for arbitrary energy-based models based on importance sampling. This technique avoids the costly exact gradient computation that involves computations proportional to the vocabulary size each time an example is presented.

The basic idea here is that the total gradient is decomposed in two parts: a "positive" contribution due to the observed example, and a "negative" contribution due to all the other examples (because of the partition function), weighted by their probability. A Monte-Carlo scheme approximates this weighted average with only a few samples, thereby resulting in a noisier estimate of the gradient. However, in the context of stochastic gradient descent, which already uses a noisy but unbiased estimator of the gradient, it is possible to take advantage of this method. In experiments on the Brown corpus, the proposed adaptive importance sampling scheme gave a **15-fold speedup** over ordinary gradient computation, yielding a network with as good training and test performance as the full-gradient model, and thus beating the interpolated trigram test performance.

More work is required to further our understanding of the reasons why some sampling schemes work and others don't, how to adapt the number of samples, and how to adapt the proposal distribution, but the proposed algorithm is already very useful.

References

Bengio, S. and Bengio, Y. (2000). 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., Ducharme, R., and Vincent, P. (2002). A neural probabilistic language model. Technical Report 1198, Dept. IRO, Université de Montréal.

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

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

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

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.

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.

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

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.

Saul, L. and Jordan, M. (1996). Exploiting tractable substructures in intractable networks. In Mozer, M., Touretzky, D., and Perrone, M., editors, *Advances in Neural Information Processing Systems 8.* MIT Press, Cambridge, MA.