# Efficient Subsampling for Training Complex Language Models

**Puyang Xu**
puyangxu@jhu.edu

**Asela Gunawardana**[#]
aselag@microsoft.com

**Sanjeev Khudanpur**
khudanpur@jhu.edu

Department of Electrical and Computer Engineering
Center for Language and Speech Processing
Johns Hopkins University
Baltimore, MD 21218, USA

[#]Microsoft Research
Redmond, WA 98052, USA

## Abstract

We propose an efficient way to train maximum entropy language models (MELM) and neural network language models (NNLM). The advantage of the proposed method comes from a more robust and efficient subsampling technique. The original multi-class language modeling problem is transformed into a set of binary problems where each binary classifier predicts whether or not a particular word will occur. We show that the binarized model is as powerful as the standard model and allows us to aggressively subsample negative training examples without sacrificing predictive performance. Empirical results show that we can train MELM and NNLM at $1\% \sim 5\%$ of the standard complexity with no loss in performance.

## 1 Introduction

Language models (LM) assign probabilities to sequences of words. They are widely used in many natural language processing applications. The probability of a sequence can be modeled as a product of local probabilities, as shown in (1), where $w_i$ is the $i^{th}$ word, and $h_i$ is the word history preceding $w_i$.

$$P(w_1, w_2, ..., w_l) = \prod_{i=1}^{l} P(w_i|h_i) \qquad (1)$$

Therefore the task of language modeling reduces to estimating a set of conditional distributions $\{P(w|h)\}$. The $n$-gram LM is a dominant way to parametrize $P(w|h)$, where it is assumed that $w$ only depends on the previous $n-1$ words. More complex models have also been proposed–MELM (Rosenfeld, 1996) and NNLM (Bengio et al., 2003) are two examples.

Modeling $P(w|h)$ can be seen as a multi-class classification problem. Given the history, we have to choose a word in the vocabulary, which can easily be a few hundred thousand words in size. For complex models such as MELM and NNLM, this poses a computational challenge for learning, because the resulting objective functions are expensive to normalize. In contrast, $n$-gram LMs do not suffer from this computational challenge. In the web era, language modelers have access to virtually unlimited amounts of data, while the computing power available to process this data is limited. Therefore, despite the demonstrated effectiveness of complex LMs, the $n$-gram is still the predominant approach for most real world applications.

Subsampling is a simple solution to get around the constraint of computing resources. For the purpose of language modeling, it amounts to taking only part of the text corpus to train the LM. For complex models such as NNLM, it has been shown that subsampling can speed up training greatly, at the cost of some degradation in predictive performance (Schwenk, 2007), allowing for trade-off between computational cost and LM quality.

Our contribution is a novel way to train complex LMs such as MELM and NNLM which allows much more aggressive subsampling without incurring as high a cost in predictive performance. The key to our approach is reducing the multi-class LM problem into a set of *binary* problems. Instead of training a $V$-class classifier, where $V$ is the size of



*Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1128–1136,
Edinburgh, Scotland, UK, July 27–31, 2011. 

the vocabulary, we train $V$ binary classifiers, each one of which performs a *one-against-all* classification. The $V$ trained binary probabilities are then renormalized to obtain a valid distribution over the $V$ words. Subsampling here can be done in the *negative* examples. Since the majority of training examples are negative for each of the binary classifiers, we can achieve substantial computational saving by only keeping subsets of them. We will show that the binarized LM is as powerful as its multi-class counterpart, while being able to sustain much more aggressive subsampling. For certain types of LMs such as MELM, there are more benefits–the binarization leads to a set of completely independent classifiers to train, which allows easy parallelization and significantly lowers the memory requirement.

Similar one-against-all approaches are often used in the machine learning community, especially by SVM (support vector machine) practitioners to solve multi-class problems (Rifkin and Klautau, 2004; Allwein et al., 2000). The goal of this paper is to show that a similar technique can also be used for language modeling and that it enables us to subsample data much more efficiently. We show that the proposed approach is useful when the dominant modeling constraint is computing power as opposed to training data.

The rest of the paper is organized as follows. In section 2, we describe our binarization and subsampling techniques for language models with MELM and NNLM as two specific examples. Experimental results are presented in Section 3, followed by discussion in Section 4.

## 2 Approximating Language Models with Binary Classifiers

Suppose we have an LM that can be written in the form

$$P(w|h) = \frac{\exp a_w(h;\theta)}{\sum_{w'} \exp a_{w'}(h;\theta)}, \quad (2)$$

where $a_w(h;\theta)$ is a parametrized history representation for word $w$.

Given a training corpus of word history pairs with empirical distribution $\tilde{P}(h,w)$, the regularized log likelihood of the training set can be written as

$$\mathcal{L} = \sum_h \tilde{P}(h) \sum_w \tilde{P}(w|h) \log P(w|h) - r(\theta), \quad (3)$$

where $r(\theta)$ is the regularizing function over the parameters.

Assuming that $r(\theta)$ can be written as a sum over per-word regularizers, namely $r(\theta) = \sum_w r_w(\theta)$, we can take the gradient of the log likelihood w.r.t $\theta$ to show that the regularized MLE for the LM satisfies

$$\sum_h \tilde{P}(h) \sum_w P(w|h) \nabla_\theta a_w(h;\theta)$$
$$= \sum_{h,w} \tilde{P}(w,h) \nabla_\theta a_w(h;\theta) - \sum_w \nabla_\theta r_w(\theta). \quad (4)$$

For each word $w$, we can define a binary classifier that predicts whether the next word is $w$ by

$$P_b(w|h) = \frac{\exp a_w(h;\theta)}{1 + \exp a_w(h;\theta)}. \quad (5)$$

The regularized training set log likelihood for all the binary classifiers is given by

$$\mathcal{L}_b = \sum_w \sum_h \tilde{P}(h) \Big[ \tilde{P}(w|h) \log P_b(w|h)$$
$$+ \tilde{P}(\bar{w}|h) \log P_b(\bar{w}|h) \Big] - \sum_w r_w(\theta), \quad (6)$$

where $P_b(\bar{w}|h) = 1 - P_b(w|h)$ is the probability of $w$ not occurring. Here we assume the same structure of the regularizer $r(\theta)$.

The regularized MLE for the binary classifiers satisfies

$$\sum_h \tilde{P}(h) \sum_w P_b(w|h) \nabla_\theta a_w(h;\theta)$$
$$= \sum_{h,w} \tilde{P}(w,h) \nabla_\theta a_w(h;\theta) - \sum_w \nabla_\theta r_w(\theta). \quad (7)$$

Notice the right hand sides of (4) and (7) are the same. Thus, taking $P'(w|h) = P_b(w|h)$ from ML trained binary classifiers gives an LM that meets the MLE constraints for language models. Therefore, if $\sum_w P_b(w|h) = 1$, ML training for the language model is equivalent to ML training of the binary classifiers and using the probabilities given by the classifiers as our LM probabilities.

Note that in practice, the probabilities given by the binary classifiers are not guaranteed to sum up to one. For tasks such as measuring perplexity,

these probabilities have to be normalized explicitly. Our hope is that for large enough data sets and rich enough history representation $a_w(h;\theta)$, we will get $\sum_w P_b(w|h) \approx 1$ so that renormalizing the classifiers to get

$$P'(w|h) = \frac{P_b(w|h)}{\sum_{w' \in V} P_b(w'|h)} \tag{8}$$

will not change the MLE constraint too much.

### 2.1 Stratified Sampling

We note that iterative estimation of the LM shown in (2) in general requires enumerating over the $T$ training cases in the training set and computing the denominator of (2) for each case at a cost of $O(V)$. Thus, each iteration of training takes $O(VT)$ in general. The complexity of estimating each of the $V$ binary classifiers is $O(T)$ per iteration, also giving $O(VT)$ per iteration in total.

However, as mentioned earlier, we are able to maximally subsample negative examples for each classifier. Thus the classifier for $w$ is trained using the $C(w)$ positive examples and a proportion $\alpha$ of the $T - C(w)$ negative examples. The total number of training examples for all $V$ classifiers is then $(1-\alpha)T + \alpha VT$. For large $V$, we choose $\alpha >> \frac{1}{1+V}$ so that this is approximately $\alpha VT$. Thus, our complexity for estimating all $V$ classifiers is $O(\alpha VT)$.

The resulting training set for each binary classifier is a stratified sample (Neyman, 1934), and our estimate needs to be calibrated to account for this. Since the training set subsamples negative examples by $\alpha$, the resulting classifier will have a likelihood ratio

$$\frac{P_b(w|h)}{1 - P_b(w|h)} = \exp a_w(h;\theta) \tag{9}$$

that is overestimated by a factor of $\frac{1}{\alpha}$. This can be corrected by simply adding $\log \alpha$ to the bias (unigram) weight of the classifier.

### 2.2 Maximum Entropy LM

MELM is an effective alternative to the standard $n$-gram LM. It provides a flexible framework to incorporate different knowledge sources in the form of feature constraints. Specifically, MELM takes the form of (2), for word $w$ following history $h$, we have the following probability definition,

$$P(w|h) = \frac{\exp \sum_i \theta_i f_i(h,w)}{\sum_{w' \in V} \exp \sum_i \theta_i f_i(h,w')}. \tag{10}$$

$f_i$ is the $i^{th}$ feature function defined over the word-history pair, $\theta_i$ is the feature weight associated with $f_i$. By defining general features, we have a natural framework to go beyond $n$-grams and capture more complex dependencies that exist in language. Previous research has shown the benefit of including various kinds of syntactic and semantic information into the LM (Khudanpur and Wu, 2000). However, despite providing a promising avenue for language modeling, MELM are computationally expensive to estimate. The bottleneck lies in the denominator of (10).

To estimate $\theta_i$s, gradient based methods can be used. The derivative of the likelihood function $\mathcal{L}$ w.r.t $\theta_i$ has a simple form, namely

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \sum_k f_i(w_k, h_k) - \sum_k \sum_{w' \in V} P(w'|h) f_i(w', h_k), \tag{11}$$

where $k$ is the index of word-history pair in the training corpus. The first term in the derivative is the observed feature count in the training corpus, the second term is the expected feature count according to the model. In order to obtain $P(w'|h)$ in the second term, we need to compute the normalizer, which involves a very expensive summation over the entire vocabulary. As described earlier, the complexity for each iteration of training is at $O(VT)$, where $T$ is the size of training corpus.

For feature sets that can be expressed hierarchically, for example $n$-gram feature set, where higher order $n$-grams imply lower order $n$-grams, Wu and Khudanpur (2000) exploit the structure of the normalizer, and precompute components that can be shared by different histories. For arbitrary feature sets, however, it may not be possible to establish the required hierarchical relations and the normalizer still needs to be computed explicitly. Goodman (2001) changes the original LM into a class-based LM, where each one of the two-step predictions only involves a much smaller summation in the normalizer. In addition, MELM estimation can be parallelized, with expected count computation done

separately for different parts of the training data and merged together at the end of each iteration. For models with massive parametrizations, this merge step can be expensive due to communication costs.

Obviously, a different way to expedite MELM training is to simply train on less data. We propose a way to do this without incurring a significant loss of modeling power, by reframing the problem in terms of binary classification. As mentioned above, we build $V$ binary classifiers of the form in (5) to model the distribution over the $V$ words. The binary classifiers use the same features as the MELM of (10), and are given by:

$$P_b(w|h) = \frac{\exp \sum_i \theta_i f_i(h, w)}{1 + \exp \sum_i \theta_i f_i(h, w)}. \quad (12)$$

We assume the features are partitioned over the vocabulary, so that each feature $f_i$ has an associated $w$ such that $f_i(h, w') = 0$ for all $w' \neq w$. Therefore, the corresponding $\theta_i$ affects only the binary classifier for $w$. This gives an important advantage in terms of parallelization–we have a set of binary classifiers with no feature sharing, and can be trained separately on different machines. The parallelized computations are completely independent and do not require the tedious communication between machines. Memory-wise, since the computations are independent, each word trainer only have to store features that are associated with the word, so the memory requirement for each individual worker is significantly reduced.

### 2.3 Neural Network LM

Neural Network Language Models (NNLM) have gained a lot of interest since their introduction (Bengio et al., 2003). While in standard language modeling, words are treated as discrete symbols, NNLM map them into a continuous space and learn their representations automatically. It is often believed that NNLM can generalize better to sequences that are not seen in the training data. However, despite having been shown to outperform standard $n$-gram LM (Schwenk, 2007), NNLM are computationally expensive to train.

Figure 1 shows the standard feed-forward NNLM architecture. Starting from the left part of the figure, each word of the $n-1$ words history is mapped to a

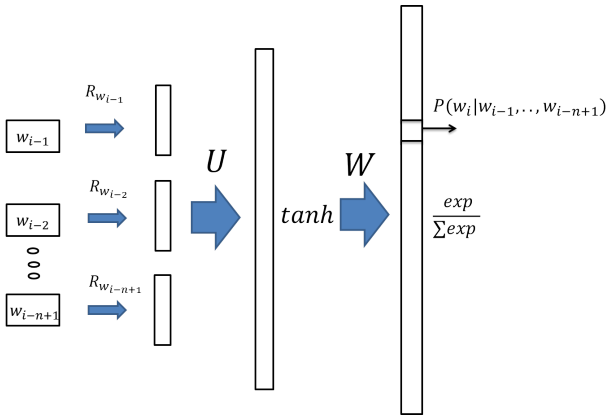


Figure 1: Feed-forward NNLM

continuous vector and concatenated. Through a nonlinear hidden layer, the neural network constructs a multinomial distribution at the output layer. Denoting the concatenated $d$-dimensional word representations $\mathbf{r}$, we have the following probability definition:

$$P(w_i = k|w_{i-1}, ..., w_{i-n+1}) = \frac{e^{a_k}}{\sum_m e^{a_m}}, \quad (13)$$

$$a_k = b_k + \sum_{l=1}^{h} W_{kl} \tanh(c_l + \sum_{j=1}^{(n-1)d} U_{lj} r_j), \quad (14)$$

where $h$ denotes the hidden layer size, $\mathbf{b}$ and $\mathbf{c}$ are the bias vectors for the output nodes and hidden nodes respectively. Note that NNLM also has the form of (2).

Stochastic gradient descent is often used to maximize the training data likelihood under such a model. The gradient can be computed using the back-propagation method. To analyze the complexity, computing an $n$-gram conditional probability requires approximately

$$O((n-1)dh + h + Vh + V) \quad (15)$$

operations, where $V$ is the size of the vocabulary. The four terms in the complexity correspond to computing the hidden layer, applying the nonlinearity, computing the output layer and normalization, respectively. The error propagation stage can be analyzed similarly and takes about the same number of operations. For typical values as used in our experiments, namely $n = 3$, $d = 50$, $h = 200$,

$V = 10000$, the majority of the complexity per iteration comes from the term $hV$. For large scale tasks, it may be impractical to train an NNLM.

A lot of previous research has focused on speeding up NNLM training. It usually aims at removing the computational dependency on $V$. Schwenk (2007) used a short list of frequent words such that a large number of out-of-list words are taken care of by a back-off LM. To reduce the gradient computation introduced by the normalizer, Bengio and Senecal (2008) proposed a different kind of importance sampling technique. A recent work (Mikolov et al., 2011) applied Goodman's class MELM trick (2001) to NNLM, in order to avoid the gigantic normalization. A similar technique has been introduced even earlier which took the idea of factorizing output layer to the extreme (Morin, 2005) by replacing the $V$-way prediction by a tree-style hierarchical prediction. The authors show a theoretical complexity reduction from $O(V)$ to $(\log V)$, but the technique requires a careful clustering which may not be easily attainable in practice.

Subsampling has also been proposed to accelerate NNLM training (Schwenk, 2007). The idea is to select random subsets of the training data in each epoch of stochastic gradient descent. After some epochs, it is very likely that all of the training examples have been seen by the model. We will show that our binary classifier representation leads to a more robust and promising subsampling strategy.

As with MELM, we notice that the parameters of (14) can be interpreted as also defining a set of $V$ per-word binary classifiers

$$P_b(w_i = k|w_{i-1}, ..., w_{i-n+1}) = \frac{e^{a_k}}{1 + e^{a_k}}, \quad (16)$$

but with a common hidden layer representation. As in MELM, we will train the classifiers, and renormalize them to obtain an NNLM over the $V$ words.

In order to train the classifiers, we need to compute all $V$ output nodes and propagate the errors back. Since the hidden layer is shared, the classifiers are not independent, and the computations can not be easily parallelized to multiple machines. However, subsampling can be done differently for each classifier. Each training instance serves as a positive example for one classifier and as a negative example for only a fraction $\alpha$ of the others. The rest of the nodes are not computed and do not produce error signal for the hidden representation. We calibrate the classifiers after subsampled training as described above for MELM.

It is straightforward to show that the dominating term $Vh$ in the complexity is reduced to $\alpha Vh$. We want to point out that compared with MELM, subsampling the negatives here does not always reduce the complexity proportionally. In cases where the vocabulary is very small, as shown in (15), computing the hidden layer can no longer be ignored. Nonetheless, real world applications such as speech recognition, usually involves a vocabulary of considerable size, therefore, subsampling in the binary setting can still achieve substantial speedup for NNLM.

## 3 Experimental Results

### 3.1 MELM

We evaluate the proposed technique on two datasets of different sizes. Our first dataset is obtained from Penn Treebank. Section 00-20 are used for training(972K tokens), section 21-22 are the validation set(77K), section 23-24(86K) are the test set. The vocabulary size of the experiment is $10,000$. This is one of the standard setups on which many researchers have reported perplexity results on (Mikolov et al., 2011).

The binary MELM is trained using stochastic gradient descent, no explicit regularization is performed (Zhang, 2004). The learning rate starts at $0.1$ and is halved every time the perplexity on the validation set stops decreasing. It usually takes around 20 iterations before no significant improvement can be obtained on the validation set. The training stops at that time.

We compare perplexity with both the standard interpolated Kneser-Ney trigram model and the standard MELM. The MELM is $L^2$ regularized and estimated using a variant of generalized iterative scaling, the regularizer is tuned on the validation data. To demonstrate the effectiveness of our subsampling approach, we compare the subsampled versions of the binary MELM and the standard MELM. In order to obtain valid perplexities, the binary LMs are first renormalized explicitly according to equation (8) for each test history.

| Model | PPL |
|---|---|
| KN Trigram | 153.0 |
| Standard MELM, Feat-I | 154.2 |
| Binary MELM, Feat-I | 153.7 |
| Standard MELM, Feat-II | 140.2 |
| Binary MELM, Feat-II | 141.1 |

Table 1: Binary MELM vs. Standard MELM

We consider two kinds of feature sets: Feat-$I$ contains only $n$-gram features, namely unigram, bigram and trigram features, with no count cutoff, the total number of features is $0.9M$. Feat-$II$ is augmented with skip-1 bigrams and skip-1 trigrams (Goodman, 2001), as well as word trigger features as described in (Rosenfeld, 1996). The total number of features in this set is $1.9M$. Note that the speedup trick described in (Wu and Khudanpur, 2000) can be used for feat-$I$, but not feat-$II$.

Table 1 shows the perplexity results when no subsampling is performed. With only $n$-gram features, the binary MELM is able to match both standard MELM and the Kneser-Ney model. We can also see that by adding features that are known to be able to improve the standard MELM, we can get the same improvement in the binary setting.

Figure 2 shows the comparisons of the two types of MELM when the training data are subsampled. The standard MELM with $n$-gram features suffers drastically as we sample more aggressively. In contrast, the binary $n$-gram MELM(Feat-$I$) does not appear to be hurt by aggressive subsampling, even when $99\%$ of the negative examples are discarded. The robustness also holds for Feat-$II$ where more complicated features are added into the model. This suggests a very efficient way of training MELM– with only $1\%$ of the computational cost, we are able to train an LM as powerful as the standard MELM.

We further test our approach on a second dataset which comes from Wall Street Journal corpus. It contains 26M training tokens and a test set of 22K tokens. We also have a held-out validation set to tune parameters. This set of experiments is intended to demonstrate that the binary subsampling technique is useful on a large text corpus where training a standard MELM is not practical, and gives a better LM than the commonly used Kneser-Ney baseline.

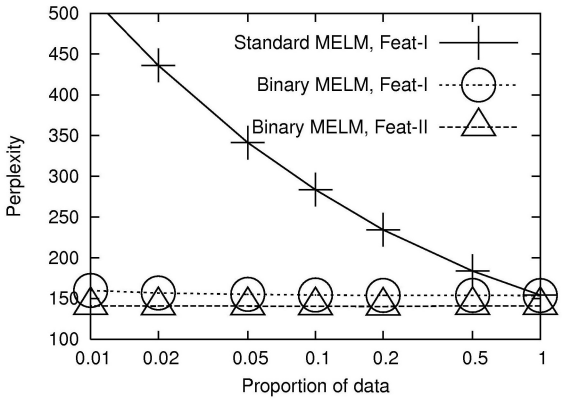


Figure 2: Subsampled Binary MELM vs. Subsampled Standard MELM

| Model | PPL |
|---|---|
| KN Trigram | 117.7 |
| Standard MELM, Trigram | 116.5 |
| Binary MELM, Feat-III, 10% | 110.2 |
| Binary MELM, Feat-III, 5% | 110.8 |
| Binary MELM, Feat-III, 2% | 112.1 |
| Binary MELM, Feat-III, 1% | 112.4 |

Table 2: Binary Subsampled MELM on WSJ

The binary MELM is trained in the same way as described in the previous experiment. Besides unigram, bigram and trigram features, we also added skip-1 bigrams and skip-1 trigrams, this gives us 7.5M features in total. We call this set of features feat-$III$. We were unable to train a standard MELM with feat-$III$ or a binary MELM without subsampling because of the computational cost. However, with our binary subsampling technique, as shown in Table 2, we are able to benefit from skip $n$-gram features with only $5\%$ of the standard MELM complexity. Also the performance does not degrade much as we discard more negative examples.

To show that such improvement in perplexity translates into gains in practical applications, we conducted a set of speech recognition experiments. The task is on Wall Street Journal, the LMs are trained on 37M tokens and are used to rescore the $n$-best list generated by the first pass recognizer with a trigram LM. The details of the experimental setup can be found in (Xu et al., 2009). Our baseline LM is an interpolated Kneser-Ney $4$-gram model.

Note that the size of the vocabulary for the task

| Model | Dev WER | Eval WER |
|---|---|---|
| KN 4-gram | 11.8 | 17.2 |
| Binary MELM, Feat-IV, 5% | 11.0 | 16.7 |
| Binary MELM, Feat-IV, 2% | 11.2 | 16.7 |
| Binary MELM, Feat-IV, 1% | 11.2 | 16.7 |

Table 3: WSJ WER improvement. Binary MELM are interpolated with KN 4-gram

| Model | PPL | | | |
|---|---|---|---|---|
| | 100% | 20% | 10% | 5% |
| Standard NNLM | 154.3 | 239.8 | 297.0 | 360.3 |
| Binary NNLM | - | 152.7 | 160.0 | 176.2 |
| KN trigram | 153.0 | - | - | - |

Table 4: Binary NNLM vs. Standard NNLM. Fixed random subset.

| Model | Interpolated PPL | | | |
|---|---|---|---|---|
| | 100% | 20% | 10% | 5% |
| Standard NNLM | 132.7 | 145.6 | 148.6 | 150.7 |
| Binary NNLM | - | 132.1 | 134.2 | 138.0 |
| KN trigram | 153.0 | - | - | - |

Table 5: Binary NNLM vs. Standard NNLM. Fixed random subset. Interpolated with KN trigram.

is 20K, for the purpose of rescoring, we are only interested in the words that exist in the $n$-best list, therefore, for the binary MELM, we only have to train about 5300 binary classifiers. For comparison, the KN 4-gram also uses the same restricted vocabulary. The features for the binary MELM are $n$-gram features up to $4$-grams plus skip-1 bigrams and skip-1 trigrams. The total number of features is 10M. We call this set of features Feat-IV.

Table 3 demonstrates the word error rate(WER) improvement enabled by our binary subsampling technique. Note that we can achieve $0.5\%$ absolute WER improvement on the test set at only $1\%$ of the standard MELM complexity. More specifically, with only 50 machines, such a reduction in complexity allows us to train a binary MELM with skip $n$-gram features in less than two hours, which is not possible for the standard MELM on 37M words.

Obviously, with more machines, the estimation can be even faster, it's also reasonable to expect that with more kinds of features, the improvement can be even larger. We think that the proposed technique opens the door for the utilization of the modeling framework provided by MELM at a scale that has not been possible before.

### 3.2 NNLM

We evaluate our binary subsampling technique on the same Penn Treebank corpus as described for the MELM experiments. Taking random subsets of the training data with the standard model is our primary baseline to compare with. The NNLM we train is a trigram LM with $tanh$ hidden units. The size of word representation and the size of hidden layer are tuned minimally on the validation set(Hidden layer size 200; Representation size 50). We adopt the same learning rate strategy as for training MELM, and the validation set is used to track perplexity performance and adjust learning rate correspondingly. All parameters are initialized randomly with mean $0$ and variance $0.01$. As with binary MELM, binary NNLM are explicitly renormalized to obtain valid perplexities.

In our first experiment, we keep the subsampled data fixed as we did for MELM. For the standard NNLM, it means only a subset of the data is seen by the model and it does not change through epochs; For binary NNLM, it means the subset of negative examples for each binary classifier does not change. Table $4$ shows the perplexity results by NNLM itself and the interpolated results are shown in Table $5$.

We can see that both models exhibit a tendency to deteriorate as we subsample more aggressively. However, the standard NNLM is clearly impacted more severely. With binary NNLM, we are able to retain all the gain after interpolation with only $20\%$ of the negative examples.

Notice that with a fixed random subset, we are not replicating the experiments of Schwenk (Schwenk, 2007) exactly, although it is reasonable to expect both models are able to benefit from seeing different random subsets of the training data. This is verified by results in Table 6 and Table 7.

The standard NNLM benefits quite a lot going from using a fixed random subset to a variable random subset, but still demonstrates a clear tendency to deteriorate as we discard more and more data. On the constrast, the binary NNLM maintains all the performance gain with only $5\%$ of the negative examples and still clearly outperforms its counterpart.

| Model | PPL | | | |
|---|---|---|---|---|
| | 100% | 20% | 10% | 5% |
| Standard NNLM | 154.3 | 157.7 | 172.2 | 186.5 |
| Binary NNLM | - | 151.7 | 150.1 | 152.1 |

Table 6: Binary NNLM vs. Standard NNLM. Variable random subset.

| Model | Interpolate PPL | | | |
|---|---|---|---|---|
| | 100% | 20% | 10% | 5% |
| Standard NNLM | 132.7 | 133.9 | 138.1 | 141.2 |
| Binary NNLM | - | 132.2 | 131.7 | 132.2 |

Table 7: Binary NNLM vs. Standard NNLM. Variable random subset. Interpolated with KN trigram.

## 4 Discussion

For the standard models, the amount of existent patterns fed into training heavily depends on the subsampling rate $\alpha$. For a small $\alpha$, the models will inevitably lose some training patterns given any reasonable number of epochs of training. Taking variable random subsets in each epoch can alleviate this problem to some extent, but still can not solve the fundamental problem. In the binary setting, we are able to do subsampling differently. While the complexity remains the same without subsampling, the majority of the complexity comes from processing negatives examples for each binary classifier. Therefore, we can achieve the same level of speedup as standard subsampling by only subsampling negative examples, and most importantly, it allows us to keep all the existent patterns(positive examples) in the training data. Of course, negative examples are important and even in the binary case, we benefit from including more of them, but since we have so many of them, they might not be as critical as positive examples in determining the distribution.

A similar conclusion can be drawn from Google's work on large LMs (Brants et al., 2007). Not having to properly smooth the LM, they are still able to benefit from large volumes of web text as training data. It is probably more important to have a high $n$-gram coverage than having a precise distribution.

The explanation here might lead us to wonder whether for the multi-class problem, subsampling the terms in the normalizer would achieve the same results. More specifically, instead of summing over all words in the vocabulary, we may choose to only consider $\alpha$ of them. In fact, the short-list approach in (Schwenk, 2007) and the adaptive importance sampling in (Bengio and Senecal, 2008) have exactly this intuition. However, in the multi-class setup, subsampling like this has to be very careful. We have to either have a good estimate of how much probability mass we've thrown away, as in the short-list approach, or have a good estimate of the entire normalizer, as in the importance sampling approach. It is very unlikely that an arbitrary random subsampling will not harm the model. Fortunately, in the binary case, the effect of random subsampling is much easier to analyze. We know exactly how much negative examples we've discarded, and they can be compensated easily in the end.

It is worth pointing out that the proposed technique is not restricted to MELM and NNLM. We have done experiments to binarize the class trick sometimes used for language modeling (Goodman, 2001; Mikolov et al., 2011), and it also proves to be useful. We plan to report these results in the future. More generally, for many large-scale multi-class problems, binarization and subsampling can be an effective combination to consider.

## 5 Conclusion

We propose efficient subsampling techniques for training large multi-class classifiers such as maximum entropy language models and neural network language models. The main idea is to replace a multi-way decision by a set of binary decisions. Since most of the training instances in the binary setting are negatives examples, we can achieve substantial speedup by subsampling only the negatives. We show by extensive experiments that this is more robust than subsampling subsets of training data for the original multi-class classifier. The proposed method can be very useful for building large language models and solving more general multi-class problems.

## Acknowledgments

This work is partially supported by National Science Foundation Grant No 0963898, the DARPA GALE Program and JHU/HLTCOE.

## References


Allwein, Erin, Robert Schapire, Yoram Singer and Pack Kaelbling. 2000. Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, 1:113-141.

Bengio, Yoshua, Rejean Ducharme and Pascal Vincent 2003. A neural probabilistic language model *Journal of Machine Learning research*, 3:1137–1155.

Bengio, Yoshua and J. Senecal 2008. Adaptive importance sampling to accelerate training of a neural probabilistic language model *IEEE Transaction on Neural Network*, Apr. 2008.

Berger, Adam, Stephen A. Della Pietra and Vicent J. Della Pietra 1996. A Maximum Entropy approach to Natural Language Processing. *Computational Linguistics*, 1996, 22:39-71.

Brants, Thorsten, Ashok C. Popat, Peng Xu, Frank J. Och and Jeffrey Dean 2007. Large language models in machine translation. *In Proceedings of 2007 Conference on Empirical Methods in Natural Language Processing*, 858–867.

Goodman, Joshua 2001. Classes for Fast Maximum Entropy Training. *Proceedings of 2001 IEEE International Conference on Acoustics, Speech and Signal Processing*.

Goodman, Joshua 2001. A bit of Progress in Language Modeling. *Computer Speech and Language*, 403-434.

Khudanpur, Sanjeev and Jun Wu 2000. Maximum Entropy Techniques for Exploiting Syntactic, Semantic and Collocational Dependencies in Language Modeling. *Computer Speech and Language*, 14(4):355-372.

Mikolov, Tomas, Stefan Kombrink, Lukas Burget, Jan "Honza" Cernocky and Sanjeev Khudanpur 2011. Extensions of recurrent neural network language model. *Proceedings of 2011 IEEE International Conference on Acoustics, Speech and Signal Processing*.

Morin, Frederic 2005. Hierarchical probabilistic neural network language model. *AISTATS'05*, pp. 246-252.

Neyman, Jerzy 1934. On the Two Different Aspects of the Representative Method: The Method of Stratified Sampling and the Method of Purposive Selection. *Journal of the Royal Statistical Society*, 97(4):558-625.

Rifkin, Ryan and Aldebaro Klautau 2004. In Defense of One-Vs-All Classification. *Journal of Machine Learning Research*.

Rosenfeld, Roni. 1996. A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, 10:187–228.

Schwenk, Holger 2007. Continuous space language model. *Computer Speech and Language*, 21(3):492-518.

Wu, Jun and Sanjeev Khudanpur. 2000. Efficient training methods for maximum entropy language modeling. *Proceedings of the 6th International Conference on Spoken Language Technologies*, pp. 114–117.

Xu, Puyang, Damianos Karakos and Sanjeev Khudanpur. 2009. Self-supervised discriminative training of statistical language models. *Proceedings of 2009 IEEE Automatic Speech Recognition and Understanding Workshop*.

Zhang, Tong 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. *Proceedings of 2004 International Conference on Machine Learnings*.