Trans-dimensional Random Fields for Language Modeling

Bin Wang[1], **Zhijian Ou**[1], **Zhiqiang Tan**[2]
[1]Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
[2]Department of Statistics, Rutgers University, Piscataway, NJ 08854, USA
wangbin12@mails.tsinghua.edu.cn, ozj@tsinghua.edu.cn, ztan@stat.rutgers.edu

Abstract

Language modeling (LM) involves determining the joint probability of words in a sentence. The conditional approach is dominant, representing the joint probability in terms of conditionals. Examples include n-gram LMs and neural network LMs. An alternative approach, called the random field (RF) approach, is used in whole-sentence maximum entropy (WSME) LMs. Although the RF approach has potential benefits, the empirical results of previous WSME models are not satisfactory. In this paper, we revisit the RF approach for language modeling, with a number of innovations. We propose a trans-dimensional RF (TDRF) model and develop a training algorithm using joint stochastic approximation and trans-dimensional mixture sampling. We perform speech recognition experiments on Wall Street Journal data, and find that our TDRF models lead to performances as good as the recurrent neural network LMs but are computationally more efficient in computing sentence probability.

1 Introduction

Language modeling is crucial for a variety of computational linguistic applications, such as speech recognition, machine translation, handwriting recognition, information retrieval and so on. It involves determining the joint probability $p(x)$ of a sentence x, which can be denoted as a pair $x = (l, x^l)$, where l is the length and $x^l = (x_1, \ldots, x_l)$ is a sequence of l words.

Currently, the dominant approach is conditional modeling, which decomposes the joint probability of x^l into a product of conditional probabilities [1] by using the chain rule,

$$p(x_1, \ldots, x_l) = \prod_{i=1}^{l} p(x_i | x_1, \ldots, x_{i-1}). \quad (1)$$

To avoid degenerate representation of the conditionals, the history of x_i, denoted as $h_i = (x_1, \cdots, x_{i-1})$, is reduced to equivalence classes through a mapping $\phi(h_i)$ with the assumption

$$p(x_i|h_i) \approx p(x_i|\phi(h_i)). \quad (2)$$

Language modeling in this conditional approach consists of finding suitable mappings $\phi(h_i)$ and effective methods to estimate $p(x_i|\phi(h_i))$. A classic example is the traditional n-gram LMs with $\phi(h_i) = (x_{i-n+1}, \ldots, x_{i-1})$. Various smoothing techniques are used for parameter estimation (Chen and Goodman, 1999). Recently, neural network LMs, which have begun to surpass the traditional n-gram LMs, also follow the conditional modeling approach, with $\phi(h_i)$ determined by a neural network (NN), which can be either a feedforward NN (Schwenk, 2007) or a recurrent NN (Mikolov et al., 2011).

Remarkably, an alternative approach is used in whole-sentence maximum entropy (WSME) language modeling (Rosenfeld et al., 2001). Specifically, a WSME model has the form:

$$p(x; \lambda) = \frac{1}{Z} \exp\{\lambda^T f(x)\} \quad (3)$$

Here $f(x)$ is a vector of features, which can be arbitrary computable functions of x, λ is the corresponding parameter vector, and Z is the global normalization constant. Although WSME models have the potential benefits of being able to naturally express sentence-level phenomena and integrate features from a variety of knowledge

[1]And the joint probability of x is modeled as $p(x) = p(x^l)p(\langle EOS\rangle|x_l)$, where $\langle EOS\rangle$ is a special token placed at the end of every sentence. Thus the distribution of the sentence length is implicitly modeled.

sources, their performance results ever reported are not satisfactory (Rosenfeld et al., 2001; Amaya and Benedí, 2001; Ruokolainen et al., 2010).

The WSME model defined in (3) is basically a Markov random field (MRF). A substantial challenge in fitting MRFs is that evaluating the gradient of the log likelihood requires high-dimensional integration and hence is difficult even for moderately sized models (Younes, 1989), let alone the language model (3). The sampling methods previously tried for approximating the gradient are the Gibbs sampling, the Independence Metropolis-Hasting sampling and the importance sampling (Rosenfeld et al., 2001). Simple applications of these methods are hardly able to work efficiently for the complex, high-dimensional distribution such as (3), and hence the WSME models are in fact poorly fitted to the data. This is one of the reasons for the unsatisfactory results of previous WSME models.

In this paper, we propose a new language model, called the trans-dimensional random field (TDRF) model, by explicitly taking account of the empirical distributions of lengths. This formulation subsequently enables us to develop a powerful Markov chain Monte Carlo (MCMC) technique, called trans-dimensional mixture sampling and then propose an effective training algorithm in the framework of stochastic approximation (SA) (Benveniste et al., 1990; Chen, 2002). The SA algorithm involves jointly updating the model parameters and normalization constants, in conjunction with trans-dimensional MCMC sampling. Section 2 and 3 present the model definition and estimation respectively.

Furthermore, we make several additional innovations, as detailed in Section 4, to enable successful training of TDRF models. First, the diagonal elements of hessian matrix are estimated during SA iterations to rescale the gradient, which significantly improves the convergence of the SA algorithm. Second, word classing is introduced to accelerate the sampling operation and also improve the smoothing behavior of the models through sharing statistical strength between similar words. Finally, multiple CPUs are used to parallelize the training of our RF models.

In Section 5, speech recognition experiments are conducted to evaluate our TDRF LMs, compared with the traditional 4-gram LMs and the recurrent neural network LMs (RNNLMs) (Mikolov et al., 2011) which have emerged as a new state-of-art of language modeling. We explore the use of a variety of features based on word and class information in TDRF LMs. In terms of word error rates (WERs) for speech recognition, our TDRF LMs alone can outperform the KN-smoothing 4-gram LM with 9.1% relative reduction, and perform comparably to the RNNLM with a slight 0.5% relative reduction. To our knowledge, this result represents the first strong empirical evidence supporting the power of using the whole-sentence language modeling approach. Our open-source TDRF toolkit is released publicly [2].

2 Model Definition

Throughout, we denote [3] by $x^l = (x_1, \dots, x_l)$ a sentence (i.e., word sequence) of length l ranging from 1 to m. Each element of x^l corresponds to a single word. For $l = 1, \dots, m$, we assume that sentences of length l are distributed from an exponential family model:

$$p_l(x^l; \lambda) = \frac{1}{Z_l(\lambda)} e^{\lambda^T f(x^l)}, \tag{4}$$

where $f(x^l) = (f_1(x^l), f_2(x^l), \dots, f_d(x^l))^T$ is the feature vector and $\lambda = (\lambda_1, \lambda_2, \dots, \lambda_d)^T$ is the corresponding parameter vector, and $Z_l(\lambda)$ is the normalization constant:

$$Z_l(\lambda) = \sum_{x^l} e^{\lambda^T f(x^l)} \tag{5}$$

Moreover, we assume that length l is associated with probability π_l for $l = 1, \dots, m$. Therefore, the pair (l, x^l) is jointly distributed as

$$p(l, x^l; \lambda) = \pi_l \, p_l(x^l; \lambda). \tag{6}$$

We provide several comments on the above model definition. First, by making explicit the role of lengths in model definition, it is clear that the model in (6) is a mixture of random fields on sentences of different lengths (namely on subspaces of different dimensions), and hence will be called a trans-dimensional random field (TDRF). Different from the WSME model (3), a crucial aspect of the TDRF model (6) is that the mixture weights π_l can be set to the empirical length probabilities in the training data. The WSME

[2]http://oa.ee.tsinghua.edu.cn/~ouzhijian/software.htm

[3]We add sup or subscript l, e.g. in $x^l, p_l()$, to make clear that the variables and distributions depend on length l.

model (3) is essentially also a mixture of RFs, but the mixture weights implied are proportional to the normalizing constants $Z_l(\lambda)$:

$$p(l, x^l; \lambda) = \frac{Z_l(\lambda)}{Z(\lambda)} \frac{1}{Z_l(\lambda)} e^{\lambda^T f(x^l)}, \qquad (7)$$

where $Z(\lambda) = \sum_{l=1}^{m} Z_l(\lambda)$.

A motivation for proposing (6) is that it is very difficult to sample from (3), namely (7), as a mixture distribution with unknown weights which typically differ from each other by orders of magnitudes, e.g. 10^{40} or more in our experiments. Setting mixture weights to the known, empirical length probabilities enables us to develop a very effective learning algorithm, as introduced in Section 3. Basically, the empirical weights serve as a control device to improve sampling from multiple distributions (Liang et al., 2007; Tan, 2015) .

Second, it can be shown that if we incorporate the length features [4] in the vector of features $f(x)$ in (3), then the distribution $p(x; \lambda)$ in (3) under the maximum entropy (ME) principle will take the form of (6) and the probabilities $(\pi_1, \ldots, \pi_m)$ in (6) implied by the parameters for the length features are exactly the empirical length probabilities.

Third, a feature $f_i(x^l)$, $1 \le i \le d$, can be any computable function of the sentence x^l, such as n-grams. In our current experiments, the features $f_i(x^l)$ and their corresponding parameters λ_i are defined to be position-independent and length-independent. For example, $f_i(x^l) = \sum_k f_i(x^l, k)$, where $f_i(x^l, k)$ is a binary function of x^l evaluated at position k. As a result, the feature $f_i(x^l)$ takes values in the non-negative integers.

3 Model Estimation

We develop a stochastic approximation algorithm using Markov chain Monte Carlo to estimate the parameters λ and the normalization constants $Z_1(\lambda), ..., Z_m(\lambda)$ (Benveniste et al., 1990; Chen, 2002). The core algorithms newly designed in this paper are the joint SA for simultaneously estimating parameters and normalizing constants (Section 3.2) and trans-dimensional mixture sampling (Section 3.3) which is used as Step I of the joint SA. The most relevant previous works that we borrowed from are (Gu and Zhu, 2001) on SA for fitting a single RF, (Tan, 2015) on sampling and estimating normalizing constants from multiple RFs of the same dimension, and (Green, 1995) on trans-dimensional MCMC.

[4]The length feature corresponding to length l is a binary feature that takes one if the sentence x is of length l, and otherwise takes zero.

3.1 Maximum likelihood estimation

Suppose that the training dataset consists of n_l sentences of length l for $l = 1, \ldots, m$. First, the maximum likelihood estimate of the length probability π_l is easily shown to be n_l/n, where $n = \sum_{l=1}^{m} n_l$. By abuse of notation, we set $\pi_l = n_l/n$ hereafter. Next, the log-likelihood of λ given the empirical length probabilities is

$$L(\lambda) = \frac{1}{n} \sum_{l=1}^{m} \sum_{x^l \in D_l} \log p_l(x^l; \lambda), \qquad (8)$$

where D_l is the collection of sentences of length l in the training set. By setting to 0 the derivative of (8) with respect to λ, we obtain that the maximum likelihood estimate of λ is determined by the following equation:

$$\frac{\partial L(\lambda)}{\partial \lambda} = \tilde{p}[f] - p_\lambda[f] = 0, \qquad (9)$$

where $\tilde{p}[f]$ is the expectation of the feature vector f with respect to the empirical distribution:

$$\tilde{p}[f] = \frac{1}{n} \sum_{l=1}^{m} \sum_{x^l \in D_l} f(x^l), \qquad (10)$$

and $p_\lambda[f]$ is the expectation of f with respect to the joint distribution (6) with $\pi_l = n_l/n$:

$$p_\lambda[f] = \sum_{l=1}^{m} \frac{n_l}{n} p_{\lambda,l}[f], \qquad (11)$$

and $p_{\lambda,l}[f] = \sum_{x^l} f(x^l) p_l(x^l; \lambda)$. Eq.(9) has the form of equating empirical expectations $\tilde{p}[f]$ with theoretical expectations $p_\lambda[f]$, as similarly found in maximum likelihood estimation of single random field models.

3.2 Joint stochastic approximation

Training random field models is challenging due to numerical intractability of the normalizing constants $Z_l(\lambda)$ and expectations $p_{\lambda,l}[f]$. We propose a novel SA algorithm for estimating the parameters λ by (9) and, simultaneously, estimating the log ratios of normalization constants:

$$\zeta_l^*(\lambda) = \log \frac{Z_l(\lambda)}{Z_1(\lambda)}, \quad l = 1, \ldots, m \qquad (12)$$

Algorithm 1 Joint stochastic approximation

```
Input: training set
 1: set initial values $\lambda^{(0)} = (0,\dots,0)^T$ and
    $\zeta^{(0)} = \zeta^*(\lambda^{(0)}) - \zeta_1^*(\lambda^{(0)})$
 2: for $t = 1, 2, \dots, t_{max}$ do
 3:     set $B^{(t)} = \emptyset$
 4:     set $(L^{(t,0)}, X^{(t,0)}) = (L^{(t-1,K)}, X^{(t-1,K)})$
        Step I: MCMC sampling
 5:     for $k = 1 \to K$ do
 6:         sampling (See Algorithm 3)
    $(L^{(t,k)}, X^{(t,k)}) = SAMPLE(L^{(t,k-1)}, X^{(t,k-1)})$
 7:         set $B^{(t)} = B^{(t)} \cup \{(L^{(t,k)}, X^{(t,k)})\}$
 8:     end for
        Step II: SA updating
 9:     Compute $\lambda^{(t)}$ based on (14)
10:     Compute $\zeta^{(t)}$ based on (15) and (16)
11: end for
```

where $Z_1(\lambda)$ is chosen as the reference value and can be calculated exactly. The algorithm can be obtained by combining the standard SA algorithm for training single random fields (Gu and Zhu, 2001) and a trans-dimensional extension of the self-adjusted mixture sampling algorithm (Tan, 2015).

Specifically, consider the following joint distribution of the pair (l, x^l):

$$p(l, x^l; \lambda, \zeta) \propto \frac{\pi_l}{e^{\zeta_l}} e^{\lambda^T f(x^l)}, \tag{13}$$

where π_l is set to n_l/n for $l = 1, \dots, m$, but $\zeta = (\zeta_1, \dots, \zeta_m)^T$ with $\zeta_1 = 0$ are hypothesized values of the truth $\zeta^*(\lambda) = (\zeta_1^*(\lambda), \dots, \zeta_m^*(\lambda))^T$ with $\zeta_1^*(\lambda) = 0$. The distribution $p(l, x^l; \lambda, \zeta)$ reduces to $p(l, x^l; \lambda)$ in (6) if ζ were identical to $\zeta^*(\lambda)$. In general, $p(l, x^l; \lambda, \zeta)$ differs from $p(l, x^l; \lambda)$ in that the marginal probability of length l is not necessarily π_l.

The joint SA algorithm, whose pseudo-code is shown in Algorithm 1, consists of two steps at each time t as follows.

Step I: MCMC sampling. Generate a sample set $B^{(t)}$ with $p(l, x^l; \lambda^{(t-1)}, \zeta^{(t-1)})$ as the stationary distribution (see Section 3.3).

Step II: SA updating. Compute

$$\lambda^{(t)} = \lambda^{(t-1)} + \gamma_\lambda \left\{ \tilde{p}[f] - \frac{\sum_{(l,x^l)\in B^{(t)}} f(x^l)}{K} \right\} \tag{14}$$

where γ_λ is a learning rate of λ; compute

$$\zeta^{(t-\frac{1}{2})} = \zeta^{(t)} + \gamma_\zeta \left\{ \frac{\delta_1(B^{(t)})}{\pi_1}, \dots, \frac{\delta_m(B^{(t)})}{\pi_m} \right\} \tag{15}$$

$$\zeta^{(t)} = \zeta^{(t-\frac{1}{2})} - \zeta_1^{(t-\frac{1}{2})} \tag{16}$$

where γ_ζ is a learning rate of ζ, and $\delta_l(B^{(t)})$ is the relative frequency of length l appearing in $B^{(t)}$:

$$\delta_l(B^{(t)}) = \frac{\sum_{(j,x^j)\in B^{(t)}} 1(j = l)}{K}. \tag{17}$$

The rationale in (15) is to adjust ζ based on how the relative frequencies of lengths $\delta_l(B^{(t)})$ are compared with the desired length probabilities π_l. Intuitively, if the relative frequency of some length l in the sample set $B^{(t)}$ is greater (or respectively smaller) than the desired length probability π_l, then the hypothesized value $\zeta_l^{(t-1)}$ is an underestimate (or overestimate) of $\zeta_l^*(\lambda^{(t-1)})$ and hence should be increased (or decreased).

Following Gu & Zhu (2001) and Tan (2015), we set the learning rates in two stages:

$$\gamma_\lambda = \begin{cases} t^{-\beta_\lambda} & \text{if } t \le t_0 \\ \frac{1}{t - t_0 + t_0^{\beta_\lambda}} & \text{if } t > t_0 \end{cases} \tag{18}$$

$$\gamma_\zeta = \begin{cases} (0.1t)^{-\beta_\zeta} & \text{if } t \le t_0 \\ \frac{1}{0.1(t-t_0) + (0.1t_0)^{\beta_\zeta}} & \text{if } t > t_0 \end{cases} \tag{19}$$

where $0.5 < \beta_\lambda, \beta_\zeta < 1$. In the first stage ($t \le t_0$), a slow-decaying rate of $t^{-\beta}$ is used to introduce large adjustments. This forces the estimates $\lambda^{(t)}$ and $\zeta^{(t)}$ to fall reasonably fast into the true values. In the second stage ($t > t_0$), a fast-decaying rate of t^{-1} is used. The iteration number t is multiplied by 0.1 in (19), to make the the learning rate of ζ decay more slowly than λ. Commonly, t_0 is selected to ensure there is no more significant adjustment observed in the first stage.

3.3 Trans-dimensional mixture sampling

We describe a trans-dimensional mixture sampling algorithm to simulate from the joint distribution $p(l, x^l; \lambda, \zeta)$, which is used with $(\lambda, \zeta) = (\lambda^{(t-1)}, \zeta^{(t-1)})$ at time t for MCMC sampling in the joint SA algorithm. The name "mixture sampling" reflects the fact that $p(l, x^l; \lambda, \zeta)$ represents a labeled mixture, because l is a label indicating that x^l is associated with the distribution $p_l(x^l; \zeta)$. With fixed (λ, ζ), this sampling algorithm can be seen as formally equivalent to reversible jump MCMC (Green, 1995), which was originally proposed for Bayes model determination.

The trans-dimensional mixture sampling algorithm consists of two steps at each time t: local jump between lengths and Markov move of sentences for a given length. In the following, we denote by $L^{(t-1)}$ and $X^{(t-1)}$ the length and sequence before sampling, but use the short notation (λ, ζ) for $(\lambda^{(t-1)}, \zeta^{(t-1)})$.

Step I: Local jump. The Metropolis-Hastings method is used in this step to sample the length. Assuming $L^{(t-1)} = k$, first we draw a new length $j \sim \Gamma(k, \cdot)$. The jump distribution $\Gamma(k, l)$ is defined to be uniform at the neighborhood of k :

$$\Gamma(k,l) = \begin{cases} \frac{1}{3}, \text{if } k \in [2, m-1], l \in [k-1, k+1] \\ \frac{1}{2}, \text{if } k = 1, l \in [1,2] \text{ or } k = m, l \in [m-1, m] \\ 0, \text{otherwise} \end{cases} \tag{20}$$

where m is the maximum length. Eq.(20) restricts the difference between j and k to be no more than one. If $j = k$, we retain the sequence and perform the next step directly, i.e. set $L^{(t)} = k$ and $X^{(t)} = X^{(t-1)}$. If $j = k+1$ or $j = k-1$, the two cases are processed differently.

If $j = k + 1$, we first draw an element (i.e., word) Y from a proposal distribution: $Y \sim g_{k+1}(y|X^{(t-1)})$. Then we set $L^{(t)} = j\ (= k+1)$ and $X^{(t)} = \{X^{(t-1)}, Y\}$ with probability

$$\min\left\{1, \frac{\Gamma(j,k)}{\Gamma(k,j)} \frac{p(j, \{X^{(t-1)}, Y\}; \lambda, \zeta)}{p(k, X^{(t-1)}; \lambda, \zeta) g_{k+1}(Y|X^{(t-1)})}\right\} \tag{21}$$

where $\{X^{(t-1)}, Y\}$ denotes a sequence of length $k+1$ whose first k elements are $X^{(t-1)}$ and the last element is Y.

If $j = k - 1$, we set $L^{(t)} = j\ (= k-1)$ and $X^{(t)} = X^{(t-1)}_{1:j}$ with probability

$$\min\left\{1, \frac{\Gamma(j,k)}{\Gamma(k,j)} \frac{p(j, X^{(t-1)}_{1:j}; \lambda, \zeta) g_k(X^{(t-1)}_k | X^{(t-1)}_{1:j})}{p(k, X^{(t-1)}; \lambda, \zeta)}\right\} \tag{22}$$

where $X^{(t-1)}_{1:j}$ is the first j elements of $X^{(t-1)}$ and $X^{(t-1)}_k$ is the kth element of $X^{(t-1)}$.

In (21) and (22), $g_{k+1}(y|x^k)$ can be flexibly specified as a proper density function in y. In our application, we find the following choice works reasonably well:

$$g_{k+1}(y|x^k) = \frac{p(k+1, \{x^k, y\}; \lambda, \zeta)}{\sum_w p(k+1, \{x^k, w\}; \lambda, \zeta)}. \tag{23}$$

Step II: Markov move. After the step of local jump, we obtain

$$X^{(t)} = \begin{cases} X^{(t-1)} & \text{if } L^{(t)} = k \\ \{X^{(t-1)}, Y\} & \text{if } L^{(t)} = k+1 \\ X^{(t-1)}_{1:k-1} & \text{if } L^{(t)} = k-1 \end{cases} \tag{24}$$

Then we perform Gibbs sampling on $X^{(t)}$, from the first element to the last element (Algorithm 2)

Algorithm 2 Markov Move

```
1: for $i = 1 \to L^{(t)}$ do
2:     draw $W \sim p(L^{(t)}, \{X^{(t)}_{1:i-1}, w, X^{(t)}_{i+1:L^{(t)}}\}; \lambda, \zeta)$
3:     set $X^{(t)}_i = W$
4: end for
```

4 Algorithm Optimization and Acceleration

The joint SA algorithm may still suffer from slow convergence, especially when λ is high-dimensional. We introduce several techniques for improving the convergence of the algorithm and reducing computational cost.

4.1 Improving SA recursion

We propose two techniques to effectively improve the convergence of SA recursion.

The first technique is to incorporate Hessian information, similarly as in related works on s-tochastic approximation (Gu and Zhu, 2001) and stochastic gradient descent algorithms (Byrd et al., 2014). But we only use the diagonal elements of the Hessian matrix to re-scale the gradient, due to high-dimensionality of λ.

Taking the second derivatives of $L(\lambda)$ yields

$$H_i = -\frac{d^2 L(\lambda)}{d\lambda_i^2} = p[f_i^2] - \sum_{l=1}^{m} \pi_l (p_l[f_i])^2 \tag{25}$$

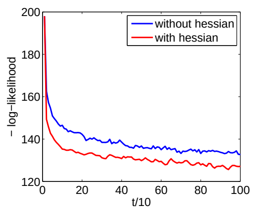
where H_i denotes the ith diagonal element of Hessian matrix. At time t, before updating the parameter λ (Step II in Section 3.2), we compute

$$H_i^{(t-\frac{1}{2})} = \frac{1}{K} \sum_{(l,x^l) \in B^{(t)}} f_i(x^l)^2 - \sum_{l=1}^{m} \pi_l (\bar{p}_l[f_i])^2, \tag{26}$$

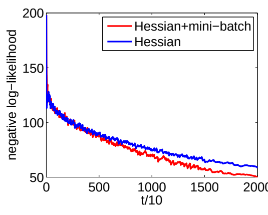
$$H_i^{(t)} = H_i^{(t-1)} + \gamma_H (H_i^{(t-\frac{1}{2})} - H_i^{(t-1)}), \tag{27}$$

where $\bar{p}_l[f_i] = |B_l^{(t)}|^{-1} \sum_{(l,x^l) \in B_l^{(t)}} f_i(x^l)$, and $B_l^{(t)}$ is the subset, of size $|B_l^{(t)}|$, containing all sentences of length l in $B^{(t)}$.

The second technique is to introduce the "mini-batch" on the training set. At each iteration, a subset $D^{(t)}$ of K sentences are randomly selected from the training set. Then the gradient is approximated with the overall empirical expectation $\tilde{p}[f]$ being replaced by the empirical expectation over the subset $D^{(t)}$. This technique is reminiscent of stochastic gradient descent using a random sub-sample of training data to achieve fast convergence



(a)



(b)

Figure 1: Examples of convergence curves on training set after introducing hessian and training set mini-batching.

of optimization algorithms (Bousquet and Bottou, 2008).

By combining the two techniques, we revise the updating equation (14) of λ to

$$\lambda_i^{(t)} = \lambda_i^{(t-1)} + \frac{\gamma_\lambda}{\max(H_i^{(t)}, h)} \times \left\{ \frac{\sum_{(l,x^l)\in D^{(t)}} f_i(x^l)}{K} - \frac{\sum_{(l,x^l)\in B^{(t)}} f_i(x^l)}{K} \right\} \tag{28}$$

where $0 < h < 1$ is a threshold to avoid $H_i^{(t)}$ being too small or even zero. Moreover, a constant t_c is added to the denominator of (18), to avoid too large adjustment of λ, i.e.

$$\gamma_\lambda = \begin{cases} \frac{1}{t_c + t^{\beta_\lambda}} & \text{if } t \le t_0, \\ \frac{1}{t_c + t - t_0 + t_0^{\beta_\lambda}} & \text{if } t > t_0. \end{cases} \tag{29}$$

Fig.1(a) shows the result after introducing hessian estimation, and Fig.1(b) shows the effect of training set mini-batching.

4.2 Sampling acceleration

For MCMC sampling in Section 3.3, the Gibbs sampling operation of drawing $X_i^{(t)}$ (Step 2 in Algorithms 2) involves calculating the probabilities of all the possible elements in position i. This is computationally costly, because the vocabulary size $|\mathcal{V}|$ is usually 10 thousands or more in practice. As a result, the Gibbs sampling operation presents a bottleneck limiting the efficiency of sampling algorithms.

We propose a novel method of using class information to effectively reduce the computational cost of Gibbs sampling. Suppose that each word in vocabulary $\mathcal{V}$ is assigned to a single class. If the total class number is $|\mathcal{C}|$, then there are, on average, $|\mathcal{V}|/|\mathcal{C}|$ words in each class. With the class information, we can first draw the class of $X_i^{(t)}$, denoted by $c_i^{(t)}$, and then draw a word belonging to class $c_i^{(t)}$. The computational cost is reduced from $|\mathcal{V}|$ to $|\mathcal{C}| + |\mathcal{V}|/|\mathcal{C}|$ on average.

Algorithm 3 Class-based MCMC sampling

```
1:  function SAMPLE($(L^{(t-1)}, X^{(t-1)})$)
2:      set $k = L^{(t-1)}$
3:      init $(L^{(t)}, X^{(t)}) = (k, X^{(t-1)})$
        Step I: Local jump
4:      generate $j \sim \Gamma(k, \cdot)$ (Eq.(20))
5:      if $j = k + 1$ then
6:          generate $C \sim Q_{k+1}(c)$
7:          generate $Y \sim \breve{g}_{k+1}(y|X^{(t-1)}, C)$ (Eq.31)
8:          set $L^{(t)} = j$ and $X^{(t)} = \{X^{(t-1)}, Y\}$ with probability (Eq.21) and (Eq.32)
9:      end if
10:     if $j = k - 1$ then
11:         set $L^{(t)} = j$ and $X^{(t)} = X_{1:k-1}^{(t-1)}$ with probability Eq.(22) and (Eq.32)
12:     end if
        Step II: Markov move
13:     for $i = 1 \to L^{(t)}$ do
14:         draw $C \sim Q_i(c)$
15:         set $c_i^{(t)} = C$ with probability (Eq.30)
16:         draw $W \in \mathcal{V}_{c_i^{(t)}}$
17:         set $X_i^{(t)} = W$
18:     end for
19:     return $(L^{(t)}, X^{(t)})$
20: end function
```

The idea of using class information to accelerate training has been proposed in various contexts of language modeling, such as maximum entropy models (Goodman, 2001b) and RNN LMs (Mikolov et al., 2011). However, the realization of this idea is different for training our models.

The pseudo-code of the new sampling method is shown in Algorithm 3. Denote by $\mathcal{V}_c$ the subset of $\mathcal{V}$ containing all the words belonging to class c. In the Markov move step (Step 13 to 18 in Algorithm 3), at each position i, we first generate a class C from a proposal distribution $Q_i(c)$ and then accept C as the new $c_i^{(t)}$ with probability

$$\min\left\{1, \frac{Q_i(c_i^{(t)})}{Q_i(C)} \frac{p_i(C)}{p_i(c_i^{(t)})}\right\} \tag{30}$$

where

$$p_i(c) = \sum_{w\in\mathcal{V}_c} p(L^{(t)}, \{X_{1:i-1}^{(t)}, w, X_{i+1:L^{(t)}}^{(t)}\}; \lambda, \zeta).$$

The probabilities $Q_i(c)$ and $p_i(c)$ depend on $\{X_{1:i-1}^{(t)}, X_{i+1:L^{(t)}}^{(t)}\}$, but this is suppressed in the notation. Then we normalize the probabilities of words belonging to class $c_i^{(t)}$ and draw a word as the new $X_i^{(t)}$ from the class $c_i^{(t)}$.

Similarly, in the local jump step with $k = L^{(t-1)}$, if the proposal $j = k + 1$ (Step 5 to 9 in Algorithm 3), we first generate $C \sim Q_{k+1}(c)$ and then generate Y from class C by

$$\breve{g}_{k+1}(y|x^k, C) = \frac{p(k+1, \{x^k, y\}; \lambda, \zeta)}{\sum_{w \in \mathcal{V}_C} p(k+1, \{x^k, w\}; \lambda, \zeta)} \quad (31)$$

with $x^k = X^{(t-1)}$. Then we set $L^{(t)} = j$ and $X^{(t)} = \{X^{(t-1)}, Y\}$ with probability as defined in (21), by setting

$$g_{k+1}(y|x^k) = Q_{k+1}(C)\breve{g}_{k+1}(y|x^k, C). \quad (32)$$

If the proposal $j = k - 1$, similarly we use acceptance probability (22) with (32).

In our application, we construct $Q_i(c)$ dynamically as follows. Write x^l for $\{X^{(t-1)}, Y\}$ in Step 8 or for $X^{(t)}$ in Step 11 of Algorithm 3. First, we construct a reduced model $p_l^c(x^l)$, by including only the features that depend on x_i^l through its class and retaining the corresponding parameters in $p_l(x^l; \lambda, \zeta)$. Then we define the distribution

$$Q_i(c) = p_l^c(\{x_{1:i-1}^l, c, x_{i+1:l}^l\}),$$

which can be directly calculated without knowing the value of x_i^l.

4.3 Parallelization of sampling

The sampling operation can be easily parallelized in SA Algorithm 1. At each time t, both the parameters λ and log normalization constants ζ are fixed at $\lambda^{(t-1)}$ and $\zeta^{(t-1)}$. Instead of simulating one Markov Chain, we simulate J Markov Chains on J CPU cores separately. As a result, to generate a sample set $B^{(t)}$ of size K, only K/J sampling steps need to be performed on each CPU core. By parallelization, the sampling operation is completed J times faster than before.

5 Experiments

5.1 PTB perplexity results

In this section, we evaluate the performance of LMs by perplexity (PPL). We use the Wall Street Journal (WSJ) portion of Penn Treebank (PTB). Sections 0-20 are used as the training data (about 930K words), sections 21-22 as the development data (74K) and section 23-24 as the test data (82K). The vocabulary is limited to 10K words, with one special token $\langle UNK \rangle$ denoting words not in the vocabulary. This setting is the same as that used in other studies (Mikolov et al., 2011).

The baseline is a 4-gram LM with modified Kneser-Ney smoothing (Chen and Goodman, 1999), denoted by KN4. We use the RNNLM toolkit[5] to train a RNNLM (Mikolov et al., 2011). The number of hidden units is 250 and other configurations are set by default[6].

Type	Features
w	$(w_{-3}w_{-2}w_{-1}w_0)(w_{-2}w_{-1}w_0)(w_{-1}w_0)(w_0)$
c	$(c_{-3}c_{-2}c_{-1}c_0)(c_{-2}c_{-1}c_0)(c_{-1}c_0)(c_0)$
ws	$(w_{-3}w_0)(w_{-3}w_{-2}w_0)(w_{-3}w_{-1}w_0)(w_{-2}w_0)$
cs	$(c_{-3}c_0)(c_{-3}c_{-2}c_0)(c_{-3}c_{-1}c_0)(c_{-2}c_0)$
wsh	$(w_{-4}w_0)$ $(w_{-5}w_0)$
csh	$(c_{-4}c_0)$ $(c_{-5}c_0)$
cpw	$(c_{-3}c_{-2}c_{-1}w_0)$ $(c_{-2}c_{-1}w_0)(c_{-1}w_0)$

Table 1: Feature definition in TDRF LMs

Word classing has been shown to be useful in conditional ME models (Chen, 2009). For our TDRF models, we consider a variety of features as shown in Table 1, mainly based on word and class information. Each word is deterministically assigned to a single class, by running the automatic clustering algorithm proposed in (Martin et al., 1998) on the training data.

In Table 1, $w_i, c_i, i = 0, -1, \ldots, -5$ denote the word and its class at different position offset i, e.g. w_0, c_0 denotes the current word and its class. We first introduce the classic word/class n-gram features (denoted by "w"/"c") and the word/class skipping n-gram features (denoted by "ws"/"cs") (Goodman, 2001a). Second, to demonstrate that long-span features can be naturally integrated in TDRFs, we introduce higher-order features "wsh"/"csh", by considering two words/classes separated with longer distance. Third, as an example of supporting heterogenous features that combine different information, the crossing features "cpw" (meaning class-predict-word) are introduced. Note that for all the feature types in Table 1, only the features observed in the training data are used.

The joint SA (Algorithm 1) is used to train the TDRF models, with all the acceleration methods described in Section 4 applied. The minibatch size $K = 300$. The learning rates γ_λ and γ_ζ are configured as (29) and (19) respectively with $\beta_\lambda = \beta_\zeta = 0.6$ and $t_c = 3000$. For t_0, it is first initialized to be 10^4. During iterations, we monitor the smoothed log-likelihood (moving average of 1000 iterations) on the PTB development data.

[5] `http://rnnlm.org/`

[6] Minibatch size=10, learning rate=0.1, BPTT steps=5. 17 sweeps are performed before stopping, which takes about 25 hours. No word classing is used, since classing in RNNLMs reduces computation but at cost of accuracy. RNNLMs were experimented with varying numbers of hidden units (100-500). The best result from using 250 hidden units is reported.

models	PPL (± std. dev.)
KN4	142.72
RNN	128.81
TDRF w+c	130.69±1.64

Table 2: The PPLs on the PTB test data. The class number is 200.

We set t_0 to the current iteration number once the rising percentage of the smoothed log-likelihoods within 100 iterations is below 20%, and then continue 5000 further iterations before stopping. The configuration of hessian estimation (Section 4.1) is $\gamma_H = \gamma_\lambda$ and $h = 10^{-4}$. L_2 regularization with constant 10^{-5} is used to avoid over-fitting. 8 CPU cores are used to parallelize the algorithm, as described in Section 4.3, and the training of each TDRF model takes less than 20 hours.

The perplexity results on the PTB test data are given in Table 2. As the normalization constants of TDRF models are estimated stochastically, we report the Monte Carlo mean and standard deviation from the last 1000 iterations for each PPL. The TDRF model using the basic "w+c" features performs close to the RNNLM in perplexity. To be compact, results with more features are presented in the following WSJ experiment.

5.2 WSJ speech recognition results

In this section, we continue to use the LMs obtained above (using PTB training and development data), and evaluate their performance measured by WERs in speech recognition, by re-scoring 1000-best lists from WSJ'92 test data (330 sentences). The oracle WER of the 1000-best lists is 3.4%, which are generated from using the Kaldi toolkit[7] with a DNN-based acoustic model.

TDRF LMs using a variety of features and different number of classes are tested. The results are shown in Table 3. Different types of features, like the skipping features, the higher-order features and the crossing features can all be easily supported in TDRF LMs, and the performance is improved to varying degrees. Particularly, the TDRF using the "w+c+ws+cs+cpw" features with class number 200 performs comparable to the RNNLM in both perplexity and WER. Numerically, the relative reduction is 9.1% compared with the KN4 LMs, and 0.5% compared with the RNN LM.

[7]`http://kaldi.sourceforge.net/`

model	WER	PPL (± std. dev.)	#feat
KN4	8.71	295.41	1.6M
RNN	7.96	256.15	5.1M
WSMEs (200c)			
w+c+ws+cs	8.87	$\approx 2.8 \times 10^{12}$	5.2M
w+c+ws+cs+cpw	8.82	$\approx 6.7 \times 10^{12}$	6.4M
TDRFs (100c)			
w+c	8.56	268.25±3.52	2.2M
w+c+ws+cs	8.16	265.81±4.30	4.5M
w+c+ws+cs+cpw	8.05	265.63±7.93	5.6M
w+c+ws+cs+wsh+csh	8.03	276.90±5.00	5.2M
TDRFs (200c)			
w+c	8.46	257.78±3.13	2.5M
w+c+ws+cs	8.05	257.80±4.29	5.2M
w+c+ws+cs+cpw	**7.92**	264.86±8.55	6.4M
w+c+ws+cs+wsh+csh	**7.94**	266.42±7.48	5.9M
TDRFs (500c)			
w+c	8.72	261.02±2.94	2.8M
w+c+ws+cs	8.29	266.34±6.13	5.9M

Table 3: The WERs and PPLs on the WSJ'92 test data. "#feat" denotes the feature number. Different TDRF models with class number 100/200/500 are reported (denoted by "100c"/"200c"/"500c")

5.3 Comparison and discussion

TDRF vs WSME. For comparison, Table 3 also presents the results from our implementation of the WSME model (3), using the same features as in Table 1. This WSME model is the same as in (Rosenfeld, 1997), but different from (Rosenfeld et al., 2001), which uses the traditional n-gram LM as the priori distribution p_0.

For the WSME model (3), we can still use a SA training algorithm, similar to that developed in Section 3.2, to estimate the parameters λ. But in this case, there is no need to introduce ζ_l, because the normalizing constants $Z_l(\lambda)$ are canceled out as seen from (7). Specifically, the learning rate γ_λ and the L_2 regularization are configured the same as in TDRF training. A fixed number of iterations with $t_0 = 5000$ is performed. The total iteration number is 10000, which is similar to the iteration number used in TDRF training.

In order to calculate perplexity, we need to estimate the global normalizing constant $Z(\lambda) = \sum_{l=1}^{m} Z_l(\lambda)$ for the WSME model. Similarly as in (Tan, 2015), we apply the SA algorithm in Section 3.2 to estimate the log normalizing constants ζ, while fixing the parameters λ to be those already estimated from the WSME model and using uniform probabilities $\pi_l \equiv m^{-1}$.

The resulting PPLs of these WSME models are extremely poor. The average test log-likelihoods per sentence for these two WSME models are

-494 and -509 respectively. However, the W-ERs from using the trained WSME models in hypothesis re-ranking are not as poor as would be expected from their PPLs. This appears to indicate that the estimated WSME parameters are not so bad for relative ranking. Moreover, when the estimated λ and ζ are substituted into our TDRF model (6) with the empirical length probabilities π_l, the "corrected" average test log-likelihoods per sentence for these two sets of parameters are improved to be -152 and -119 respectively. The average test log-likelihoods are both -96 for the two corresponding TDRF models in Table 3. This is some evidence for the model deficiency of the WSME distribution as defined in (3), and introducing the empirical length probabilities gives a more reasonable model assumption.

TDRF vs conditional ME. After training, TDRF models are computationally more efficient in computing sentence probability, simply summing up weights for the activated features in the sentence. The conditional ME models (Khudanpur and Wu, 2000; Roark et al., 2004) suffer from the expensive computation of local normalization factors. This computational bottleneck hinders their use in practice (Goodman, 2001b; Rosenfeld et al., 2001). Partly for this reason, although building conditional ME models with sophisticated features as in Table 1 is theoretically possible, such work has not been pursued so far.

TDRF vs RNN. The RNN models suffer from the expensive softmax computation in the output layer [8]. Empirically in our experiments, the average time costs for re-ranking of the 1000-best list for a sentence are 0.16 sec vs 40 sec, based on TDRF and RNN respectively (no GPU used).

6 Related Work

While there has been extensive research on conditional LMs, there has been little work on the whole-sentence LMs, mainly in (Rosenfeld et al., 2001; Amaya and Benedí, 2001; Ruokolainen et al., 2010). Although the whole-sentence approach has potential benefits, the empirical results of previous WSME models are not satisfactory, almost the same as traditional n-gram models. After incorporating lexical and syntactic information, a mere relative improvement of 1% and 0.4% respectively in perplexity and in WER is reported for the resulting WSEM (Rosenfeld et al., 2001). Subsequent studies of using WSEMs with grammatical features, as in (Amaya and Benedí, 2001) and (Ruokolainen et al., 2010), report perplexity improvement above 10% but no WER improvement when using WSEMs alone.

Most RF modeling has been restricted to fixed-dimensional spaces [9]. Despite recent progress, fitting RFs of moderate or large dimensions remains to be challenging (Koller and Friedman, 2009; Mizrahi et al., 2013). In particular, the work of (Pietra et al., 1997) is inspiring to us, but the improved iterative scaling (IIS) method for parameter estimation and the Gibbs sampler are not suitable for even moderately sized models. Our TDRF model, together with the joint SA algorithm and trans-dimensional mixture sampling, are brand new and lead to encouraging results for language modeling.

7 Conclusion

In summary, we have made the following contributions, which enable us to successfully train T-DRF models and obtain encouraging performance improvement.

- The new TDRF model and the joint SA training algorithm, which simultaneously updates the model parameters and normalizing constants while using trans-dimensional mixture sampling.
- Several additional innovations including accelerating SA iterations by using Hessian information, introducing word classing to accelerate the sampling operation and improve the smoothing behavior of the models, and parallelization of sampling.

In this work, we mainly explore the use of features based on word and class information. Future work with other knowledge sources and larger-scale experiments is needed to fully exploit the advantage of TDRFs to integrate richer features.

8 Acknowledgments

This work is supported by Toshiba Corporation, National Natural Science Foundation of China (NSFC) via grant 61473168, and Tsinghua Initiative. We thank the anonymous reviewers for helpful comments on this paper.

[8] This deficiency could be partly alleviated with some speed-up methods, e.g. using word clustering (Mikolov, 2012) or noise contrastive estimation (Mnih and Kavukcuoglu, 2013).

[9] Using local fixed-dimensional RFs in sequential models was once explored, e.g. temporal restricted Boltzmann machine (TRBM) (Sutskever and Hinton, 2007).

References

Fredy Amaya and José Miguel Benedí. 2001. Improvement of a whole sentence maximum entropy language model using grammatical features. In *Association for Computational Linguistics (ACL).*

Albert Benveniste, Michel Métivier, and Pierre Priouret. 1990. *Adaptive algorithms and stochastic approximations*. New York: Springer.

Olivier Bousquet and Leon Bottou. 2008. The tradeoffs of large scale learning. In *NIPS*, pages 161–168.

Richard H Byrd, SL Hansen, Jorge Nocedal, and Yoram Singer. 2014. A stochastic quasi-newton method for large-scale optimization. *arXiv preprint arXiv:1401.7020.*

Stanley F. Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13:359–394.

Hanfu Chen. 2002. *Stochastic approximation and its applications*. Springer Science & Business Media.

Stanley F. Chen. 2009. Shrinking exponential language models. In *Proc. of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics.*

Joshua Goodman. 2001a. A bit of progress in language modeling. *Computer Speech & Language*, 15:403–434.

Joshua Goodman. 2001b. Classes for fast maximum entropy training. In *Proc. of International Conference on Acoustics, Speech, and Signal Processing (ICASSP).*

Peter J. Green. 1995. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82:711–732.

Ming Gao Gu and Hong-Tu Zhu. 2001. Maximum likelihood estimation for spatial models by markov chain monte carlo stochastic approximation. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63:339–355.

Sanjeev Khudanpur and Jun Wu. 2000. Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling. *Computer Speech & Language*, 14:355–372.

Daphne Koller and Nir Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Faming Liang, Chuanhai Liu, and Raymond J Carroll. 2007. Stochastic approximation in monte carlo computation. *Journal of the American Statistical Association*, 102(477):305–320.

Sven Martin, Jörg Liermann, and Hermann Ney. 1998. Algorithms for bigram and trigram word clustering. *Speech Communication*, 24:19–37.

Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan H Cernocky, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *Proc. of International Conference on Acoustics, Speech and Signal Processing (ICASSP).*

Tomáš Mikolov. 2012. Statistical language models based on neural networks. *Ph.D. thesis, Brno University of Technology.*

Yariv Dror Mizrahi, Misha Denil, and Nando de Freitas. 2013. Linear and parallel learning of markov random fields. *arXiv preprint arXiv:1308.6342.*

Andriy Mnih and Koray Kavukcuoglu. 2013. Learning word embeddings efficiently with noise-contrastive estimation. In *Neural Information Processing Systems (NIPS).*

Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. 1997. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:380–393.

Brian Roark, Murat Saraclar, Michael Collins, and Mark Johnson. 2004. Discriminative language modeling with conditional random fields and the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL)*, page 47.

Ronald Rosenfeld, Stanley F. Chen, and Xiaojin Zhu. 2001. Whole-sentence exponential language models: a vehicle for linguistic-statistical integration. *Computer Speech & Language*, 15:55–73.

Ronald Rosenfeld. 1997. A whole sentence maximum entropy language model. In *Proc. of Automatic Speech Recognition and Understanding (ASRU).*

Teemu Ruokolainen, Tanel Alumäe, and Marcus Dobrinkat. 2010. Using dependency grammar features in whole sentence maximum entropy language model for speech recognition. In *Baltic HLT.*

Holger Schwenk. 2007. Continuous space language models. *Computer Speech & Language*, 21:492–518.

Ilya Sutskever and Geoffrey E Hinton. 2007. Learning multilevel distributed representations for high-dimensional sequences. In *International Conference on Artificial Intelligence and Statistics (AISTATS).*

Zhiqiang Tan. 2015. Optimally adjusted mixture sampling and locally weighted histogram. In *Technical Report, Department of Statistics, Rutgers University.*

Laurent Younes. 1989. Parametric inference for imperfectly observed gibbsian fields. *Probability theory and related fields*, 82:625–645.