Learning Trans-dimensional Random Fields with Applications to Language Modeling

Bin Wang, *Student Member, IEEE,* Zhijian Ou, *Senior Member, IEEE,* and Zhiqiang Tan

Abstract—To describe trans-dimensional observations in sample spaces of different dimensions, we propose a probabilistic model, called the trans-dimensional random field (TRF) by explicitly mixing a collection of random fields. In the framework of stochastic approximation (SA), we develop an effective training algorithm, called augmented SA, which jointly estimates the model parameters and normalizing constants while using trans-dimensional mixture sampling to generate observations of different dimensions. Furthermore, we introduce several statistical and computational techniques to improve the convergence of the training algorithm and reduce computational cost, which together enable us to successfully train TRF models on large datasets. The new model and training algorithm are thoroughly evaluated in a number of experiments. The word morphology experiment provides a benchmark test to study the convergence of the training algorithm and to compare with other algorithms, because log-likelihoods and gradients can be exactly calculated in this experiment. For language modeling, our experiments demonstrate the superiority of the TRF approach in being computationally more efficient in computing data probabilities by avoiding local normalization and being able to flexibly integrate a richer set of features, when compared with n-gram models and neural network models.

Index Terms—Language modeling, Random field, Stochastic approximation, Trans-dimensional sampling, Undirected graphical modeling.

1 Introduction

GRAPHICAL models [1], [2] have emerged as a general framework for describing and applying probabilistic models in building intelligent systems, and can be broadly classified into two classes. In the directed graphical models (also known as Bayesian networks), the joint distribution is factorized into a product of local conditional density functions, whereas in the undirected graphical models (also known as Markov random fields or Markov networks) the joint density function is defined to be proportional to the product of local potential functions. In certain applications, namely *fixed-dimensional* settings, the probabilistic model specifies a joint distribution over a fixed set of random variables, for example, in the modeling of fixed-size images. However, in many other applications, the probabilistic model relates to a much more complex sample space, which will be called the *trans-dimensional* setting: the observations can be of different dimensions. A familiar case is temporal modeling of sequential data, where each observation is a sequence of a random length. It is desirable to build a probabilistic model applicable to sequences of different lengths.

Currently in the trans-dimensional setting, the directed graphical modeling approach, e.g. through hidden Markov models or, more generally, dynamic Bayesian networks, is dominant particularly for sequential data. In contrast, the undirected graphical modeling approach is rarely used in the trans-dimensional setting. This is presumably because fitting undirected models is more challenging than fitting directed models [2]. In general, calculating the log-likelihood and its gradient is analytically intractable even for fixed-dimensional undirected models, because this involves evaluating the normalizing constant (also called the partition function in physics) and, respectively, the expectation with respect to the model distribution. It should be noted that the normalizing constant and model expectation can be exactly calculated or efficiently approximated under some limited circumstances, mostly in low tree-width[1] random fields (e.g. chain-structured) with moderately sized state spaces[2]. The widely used conditional random fields (CRFs) [3] satisfy these conditions, by only modeling the conditional distribution over labels given the observations. As a result, CRFs deal with trans-dimensional data but with a significantly reduced sample space of labels. Indeed, currently much of random field learning is pursued for CRFs. But CRFs can only be used for discriminative tasks, e.g. segmenting and labeling of natural language sentences or images.

In this paper, we are interested in building random field (RF) models for trans-dimensional observations. The fields can have greater tree-widths (e.g. 5) and larger sized state spaces (e.g. 10^4), so that they can capture higher-order interactions inherent in the trans-dimensional data. Roughly speaking, the main advantages of undirected modeling over directed modeling are: (1) undirected modeling is more natural for certain domains such as relational data, where fixing the directions of edges is awkward in a graphical model, and (2) the undirected representation provides greater flexibility and potentially more powerful modeling capacity in avoiding local normalization and acyclicity requirements. Eliminating these requirements allows us to easily encode a much richer set of patterns/features which characterize the studied phenomenon.

As remarked in [2], it can be problematic to construct a template-based RF model with a certain dimension, and apply

- *Bin Wang and Zhijian Ou are with the Department of Electronic Engineering, Tsinghua university, Beijing, China. E-mail: ozj@tsinghua.edu.cn*
- *Zhiqiang Tan is with the Department of Statistics, Rutgers University, NJ, USA. Email: ztan@stat.rutgers.edu*

1. The tree-width of a graph is defined as the minimum width over all possible tree decompositions of the graph, which measures roughly how close the graph is to a tree. The tree-width of a tree is 1.

2. The state space of a multivariate model is the set of all possible values for each coordinate of a multivariate observation.

it to data of a very different dimension. A pioneering work in building RF models for trans-dimensional observations is [4], but the proposed Improved Iterative Scaling (IIS) method for parameter estimation and the Gibbs sampler are not suitable for even moderately sized RF models. Inducing RF models of English word spellings, being sequences of characters, is studied in [4] without numerical evaluation of performances. It is interesting to note that the feature induction and IIS algorithms proposed in [4] are used mostly in later CRF studies. Another previous work, which is more directly related to our study of RF modeling of trans-dimensional data, is whole-sentence maximum entropy (WSME) language modeling [5]. Language modeling (LM) is crucial for a variety of computational linguistic applications, such as speech recognition, machine translation, handwriting recognition, information retrieval and so on. Except the WSME LMs, most LMs follow the directed modeling approach, e.g. n-gram LMs [6], neural network (NN) LMs [7], [8], [9] and conditional maximum entropy LMs [10], [11], [12]. Although WSME models have the potential benefits of being able to naturally express sentence-level phenomena and integrate features from a variety of knowledge sources, their performance results ever reported are not satisfactory [5], [13], [14] (see Section 2.2). The learning algorithm used is Generalized Iterative Scaling (GIS) [15] and the sampling methods for approximating model expectations are Gibbs sampling, independence Metropolis-Hasting sampling and importance sampling [5]. Simple applications of these methods are hardly able to work efficiently in a trans-dimensional setting especially with large-size state space[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.

To describe trans-dimensional observations, we propose a new probabilistic model, called the trans-dimensional random field (TRF) model, by explicitly mixing a collection of RFs in sample spaces of different dimensions. With this model formulation, we develop an effective training algorithm in the framework of stochastic approximation (SA) [16], [17], [18], to jointly estimate the model parameters and normalizing constants. The training algorithm, which is called augmented SA (AugSA) algorithm, involves two steps, sampling observations and then updating estimates of model parameters and normalizing constants, at each iteration. For the sampling step, we also develop a powerful Markov chain Monte Carlo (MCMC) technique, called trans-dimensional mixture sampling (TransMS). Furthermore, we introduce several statistical and computational techniques, including two-stage configuration of learning rates, use of empirical variances to rescale SA updates, proper specification of dimension probabilities for mixture sampling, a modeling strategy to deal with rare high-dimensional observations (e.g. rare long sentences), use of class information and multiple CPUs to reduce the computational cost, regularization and stopping decision. All these techniques together enable successful training of our TRF models on large datasets.

The TRF modeling approach and the proposed AugSA and TransMS algorithms are thoroughly evaluated in three experiments. The first is the word morphology experiment as studied in [4]. Due to the small size of the state space in this experiment, parameter learning can be performed with exactly calculated log-likelihoods and gradients. This experiment serves as a valid benchmark to study the convergence of Monte Carlo training algorithms with various configurations. The next two experiments are about language modeling, which is our motivating application. We show that TRF LMs significantly outperform n-gram LMs [6], and perform better than recurrent neural network (RNN) LMs [8], [19] and close to Long Short Term Memory (LSTM) LMs [9], [20] but being much faster in computing sentence probabilities. Moreover, interpolated TRF and LSTM LMs produces the lowest word error rate (WER) in speech recognition. We examine the scalability of TRF LMs by incrementally increasing the size of the training set and find that they can scale to using training data of up to 32 million words. All these experiments demonstrate the superiority of the TRF models and the effectiveness of the AugSA training algorithms. To our knowledge, these results represent the first strong empirical evidence supporting the power of using RF approach to language modeling.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. We introduce the TRF model formulation in Section 3 and present the training algorithm, AugSA with TransMS, in Section 4. Additional statistical and computational techniques for training TRF models are described in Section 5. The experimental evaluations are reported in Sections 6, 7, and 8. We conclude the paper with a discussion in Section 9.

Preliminary part of this work was published in [21]. New features in this paper include model comparison in Section 3.1, a detailed derivation of the learning algorithm (Section 4.1), an importance sampling extension (Section 4.2, 5.3), simpler configuration of learning rates (Section 5.1), a more effective method of rescaling SA update (Section 5.2), dealing with rare high-dimensional observations (Section 5.4), stop decision (Appendix D), and extensive new empirical results using new datasets and "tied" features and providing new comparisons with Adam optimiser [22], annealed importance sampling (AIS) [23] based estimates of normalizing constants, and LSTM LMs.

2 Related work

The central problem addressed in this paper involves building RF models for trans-dimensional observations. There are few directly related studies, e.g. [4] and [5]. However, there have been many studies of using RF models in the fixed-dimensional setting or for discriminative modeling (mainly related to CRFs). In the following, we will briefly discuss these studies and the connection to our work. Moreover, we further examine related work in language modeling, which is our motivating application in trans-dimensional modeling.

2.1 Learning with random fields

There is an extensive literature devoted to maximum likelihood (ML) learning of random fields, which is in general difficult because calculating the log-likelihood and its gradient is analytically intractable. Roughly speaking, there are two types of approximate methods — gradient methods and lower bound methods. The gradient methods make explicit use of the gradient. An important class is stochastic approximation methods [16], which approximates the model expectations by Monte Carlo sampling for calculating the gradient. The classic algorithm, initially proposed in [24], is often called stochastic maximum likelihood (SML). In the literature on training restricted Boltzmann machines (RBMs), SML is also known as persistent contrastive divergence (PCD) [25] to emphasize that the Markov chain is not reset between parameter updates. Among the lower bound methods for ML learning of random fields, an important class is iterative scaling

3. In LMs, the size of state space is the vocabulary size, usually above 10^4.

methods such as GIS [15] and IIS [4]. These algorithms iteratively optimize lower bounds of the likelihood function to find the maximum likelihood estimates and are mostly studied in the context of maximum entropy (maxent) parameter estimation[4] and logistic regression which is a conditional version of maxent. In practice the gradient methods are shown to be much faster than the lower bound methods [26], [27].

Remarkably, the above studies are mostly restricted to training a single fixed-dimensional RF model, and the parameter and the normalizing constant are estimated separately. Beyond the simple application of SA as in SML, we develop the augmented SA algorithm for jointly estimating the model parameter and multiple normalizing constants. Basically, stochastic approximation provides a mathematical framework for stochastically solving a root finding problem, which has the form of expectations being equal to zeros [16], [17], [18]. The starting point for augmented SA is to formulate a system of simultaneous equations in this form for both the maximum likelihood estimates and the normalizing constants. The most relevant previous works that inspire our algorithmic development are [28] on SA for fitting a single RF, [29] on sampling and estimating normalizing constants from multiple RFs of the same dimension, and [30] on trans-dimensional MCMC.

Finally, it is worthwhile to point out that by only modeling the conditional distribution over labels given the observations, CRFs deal with trans-dimensional data but with a significantly reduced sample space of labels. In order for CRFs to be computationally tractable, usual CRF formulations are mostly limited to incorporating pairwise interactions among labels. In order to allow for higher-order interactions, there presents the challenge of building RF models in the trans-dimensional setting with greater tree-widths, which is the same concern addressed in this paper.

2.2 Language modeling

Language modeling 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 to language modeling is the directed or conditional modeling, which decomposes the joint probability of x^l into a product of conditional probabilities[5] by using the chain rule,

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

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

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

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 [6]. 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 [7] or a recurrent NN [8], [9].

Remarkably, an alternative approach is used in whole-sentence maximum entropy (WSME) language modeling [5]. Specifically, a WSME model has the form

$$p(x; \lambda) = \frac{1}{Z(\lambda)} e^{\lambda^T f(x)}. \tag{1}$$

Here $f(x)$ is a vector of features, which are computable functions of x such as n-grams conventionally used, λ is the corresponding parameter vector, and $Z(\lambda) = \sum_x e^{\lambda^T f(x)}$ is the global normalizing constant, which is typically analytically intractable.

While there has been extensive research on conditional LMs, there has been little work on the whole-sentence LMs, mainly in [5], [13], [14]. Although the whole-sentence approach has the potential advantage of being able to flexibly integrate a richer set of features, the empirical results of previous WSME LMs are not satisfactory, almost the same as traditional n-gram LMs. After incorporating lexical and syntactic information, a mere relative improvement of 1% and 0.4% respectively in perplexity and in WER was reported for the resulting WSME LM [5]. Subsequent studies of using WSME LMs with grammatical features, as in [13] and [14], reported perplexity improvement above 10% but no WER improvement when using WSME LMs alone.

3 Model Definition

Suppose that it is of interest to build random field models for multiple sets of observations of different dimensions, such as images of different sizes or sentences of different lengths. Denote by x^j an observation in a sample space $\mathcal{X}^j$ of dimension j, ranging from 1 to m. The space of all observations is then the union $\mathcal{X} = \cup_{j=1}^m \mathcal{X}^j$. To emphasize the dimensionality, each observation in $\mathcal{X}$ can be represented as a pair $x = (j, x^j)$, even though j is identified as the dimension of x^j. By abuse of notation, write $f(x) = f(x^j)$ for features of x.

For $j = 1, \ldots, m$, assume that the observations x^j are distributed from a random field in the form

$$p_j(x^j; \lambda) = \frac{1}{Z_j(\lambda)} e^{\lambda^T f(x^j)},$$

where $f(x^j) = (f_1(x^j), f_2(x^j), \ldots, f_d(x^j))^T$ is a feature vector, $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_d)^T$ is the corresponding parameter vector, and $Z_j(\lambda)$ is the normalizing constant:

$$Z_j(\lambda) = \sum_{x^j} e^{\lambda^T f(x^j)}, \qquad j = 1, \ldots, m.$$

For identifiability, assume that no linear combination of the features $f_i(x^j)$, $i = 1, \ldots, d$, is a constant function of x^j. Moreover, assume that dimension j is associated with a probability π_j for $j = 1, \ldots, m$ with $\sum_{j=1}^m \pi_j = 1$. Therefore, the pair (j, x^j) is jointly distributed as

$$p(j, x^j; \pi, \lambda) = \pi_j \, p_j(x^j; \lambda) = \frac{\pi_j}{Z_j(\lambda)} e^{\lambda^T f(x^j)}, \tag{2}$$

where $\pi = (\pi_1, \ldots, \pi_m)^T$.

A feature $f_i(x^j)$, $i = 1, \ldots, d$, can be any computable function of the input x^j. For various applications such as language modeling, the parameters of local potential functions across locations are often tied together [4]. In our current experiments, each feature $f_i(x^j)$ is defined in the form $f_i(x^j) = \sum_k f_i(x^j, k)$, where $f_i(x^j, k)$ is a binary function of x^j evaluated at position

4. It can be proved [4] that the maxent distribution is the same as the maximum likelihood distribution from the closure of the set of RF distributions.

5. 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.

k. In a trigram example, $f_i(x^j, k)$ equals to 1 if three specific words appear at positions k to $k+2$ and $k \leq j-2$. The binary features $f_i(x^j, k)$ share the same parameter λ_i for different positions k and dimensions j, so called position-independent and dimension-independent. Hence, the feature $f_i(x^j)$ indicates the count of nonzero $f_i(x^j, k)$ over k in the observation x^j and takes values as non-negative integers.

3.1 Comparison between WSME and TRF models

We comment on the connection and difference between WSME and TRF models. Suppose that we add the dimension features in the WSME model (1) and obtain

$$p(j, x^j; \lambda, \nu) = \frac{1}{Z(\lambda, \nu)} e^{\nu^T \delta(j) + \lambda^T f(x^j)}, \tag{3}$$

where $\delta(j) = (\delta_1(j), \cdots, \delta_m(j))^T$ denotes the dimension features such that $\delta_l(j) = 1(j = l)$, $f(x^j)$ denotes the ordinary features as used in both models (1) and (2), $\nu = (\nu_1, \ldots, \nu_m)^T$ and λ are the corresponding parameter vectors, and $Z(\lambda, \nu)$ is the global normalizing constant

$$Z(\lambda, \nu) = \sum_{j=1}^{m} \sum_{x^j \in \mathcal{X}^j} e^{\nu^T \delta(j) + \lambda^T f(x^j)} = \sum_{j=1}^{m} e^{\nu_j} Z_j(\lambda). \tag{4}$$

We show later in Proposition 1 that when both fitted by maximum likelihood estimation, model (3) is equivalent to model (2) but with different parameterization. The parameters in model (3) are (λ, ν), whereas the parameters in model (2) are (π, λ). Therefore, an important distinction of our TRF approach for trans-dimensional modeling from previous WSME works lies in the use of dimension features, which has significance consequences in both model definition and model learning.

First, it is clear that model (2) is a mixture of random fields on subspaces of different dimensions, with mixture weights explicitly as free parameters. Hence model (2) will be called a trans-dimensional random field (TRF). Moreover, by maximum likelihood, the mixture weights can be estimated to be the empirical dimension probabilities (see Proposition 1).

Second, it is instructive to point out that model (1) is essentially also a mixture of RFs, but the mixture weights implied are fixed to be proportional to the normalizing constants $Z_j(\lambda)$:

$$p(j, x^j; \lambda) = \frac{Z_j(\lambda)}{Z(\lambda)} \cdot \frac{1}{Z_j(\lambda)} e^{\lambda^T f(x^j)}, \tag{5}$$

where $Z(\lambda) = \sum_x e^{\lambda^T f(x)} = \sum_{j=1}^m Z_j(\lambda)$. Typically the unknown mixture weights (5) may differ from the empirical length probabilities and also from each other by orders of magnitudes, e.g. 10^{40} or more in our experiments. As a result, it is very difficult to efficiently sample from model (1), in addition to the fact that the length probabilities are poorly fitted for model (1). Setting mixture weights to the known, empirical length probabilities enables us to develop an effective learning algorithm, as introduced in Section 4. Basically, the empirical weights serve as a control device to improve sampling from multiple distributions [29], [31].

4 Model Learning

We develop a novel learning algorithm in the SA framework [17], [18] to estimate both the parameters λ and the normalization constants $Z_1(\lambda), \ldots, Z_m(\lambda)$. Equivalently, we estimate the parameters λ and the log ratios of $Z_j(\lambda)$:

$$\zeta_j^*(\lambda) = \log \frac{Z_j(\lambda)}{Z_1(\lambda)}, \quad j = 1, \ldots, m,$$

where $Z_1(\lambda)$ is chosen as the reference value such that it can be calculated exactly. The core ingredients newly designed are the augmented SA (AugSA) scheme for simultaneously updating parameters and normalizing constants (Section 4.2) and trans-dimensional mixture sampling (TransMS) (Section 4.3), which is used in the MCMC sampling step of AugSA.

4.1 Maximum likelihood learning

Suppose that the training data consist of a subset of n_j observations of dimension j, denoted by D_j, for $j = 1, \ldots, m$. We derive the maximum likelihood estimates of (π, λ) for model (2).

Proposition 1. *For model* (2)*, the maximum likelihood estimate (MLE) of* π *is* $\hat{\pi} = (\hat{\pi}_1, \ldots, \hat{\pi}_m)^T$ *with* $\hat{\pi}_j = n_j/n$ *for* $j = 1, \ldots, m$*. Moreover, the MLE of* λ *is determined by*

$$\tilde{p}[f] - p_\lambda[f] = 0. \tag{6}$$

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

$$\tilde{p}[f] = \frac{1}{n} \sum_{j=1}^{m} \sum_{x^j \in D_j} f(x^j) = \sum_{j=1}^{m} \frac{n_j}{n} \tilde{p}_j[f],$$

with $\tilde{p}_j[f] = n_j^{-1} \sum_{x^j \in D^j} f(x^j)$*, and* $p_\lambda[f]$ *is the expectation of* f *with respect to the joint distribution (2),* $p(j, x^j; \hat{\pi}, \lambda)$*:*

$$p_\lambda[f] = \sum_{j=1}^{m} \hat{\pi}_j p_{\lambda,j}[f],$$

with $p_{\lambda,j}[f] = \sum_{x^j \in \mathcal{X}^j} f(x^j) p_j(x^j; \lambda)$*.*

Proof. See Appendix A. □

Then we demonstrate that maximum likelihood learning of TRF model (2) and of model (3) are equivalent to each other but with different parameterization.

Proposition 2. *(i) If* $\hat{\lambda}$ *is an MLE of* λ *in model* (2)*, then* $(\check{\lambda}, \check{\nu})$ *are MLEs of* (λ, ν) *in model* (3)*, where* $\check{\lambda} = \hat{\lambda}$ *and for a constant* c*,*

$$e^{\check{\nu}_j} = c\hat{\pi}_j e^{-\zeta_j^*(\check{\lambda})}, \quad j = 1, \ldots, m. \tag{7}$$

(ii) If $(\check{\lambda}, \check{\nu})$ *are MLEs of* (λ, ν) *in model* (3)*, then Eq.* (7) *holds and* $\hat{\lambda} = \check{\lambda}$ *is an MLE of* λ *in model* (2)*.*
(iii) In either (i) or (ii), the fitted model (3)*,* $p(j, x^j; \check{\lambda}, \check{\nu})$*, is identical to the fitted model* (2)*,* $p(j, x^j; \hat{\pi}, \hat{\lambda})$*.*

Proof. See Appendix A. □

As suggested by Eq. (7) and motivated by self-adjusted mixture sampling in [29], we define the following distribution of (j, x^j) on $\mathcal{X}$ by a re-parameterization of model (3):

$$p(j, x^j; \lambda, \zeta) \propto (\hat{\pi}_j e^{-\zeta_j}) e^{\lambda^T f(x^j)}, \tag{8}$$

where $\zeta = (\zeta_1, \ldots, \zeta_m)^T$ with $\zeta_1 = 0$, and ζ_j is related to ν_j through $e^{\nu_j} = \hat{\pi}_j e^{-\zeta_j}$ and can be interpreted as the hypothesized value of the true value $\zeta_j^*(\lambda)$. By Proposition 2, after the re-parametrization from (3) to (8), the MLE of λ under model (8)

```
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 $(j^{(t,0)}, x^{(t,0)}) = (j^{(t-1,K)}, x^{(t-1,K)})$
        Step I: MCMC sampling
5:      for $k = 1 \to K$ do
6:          $(j^{(t,k)}, x^{(t,k)})$ = sample($(j^{(t,k-1)}, x^{(t,k-1)})$) (See Tab.1)
7:          set $B^{(t)} = B^{(t)} \cup \{(j^{(t,k)}, x^{(t,k)})\}$
8:      end for
        Step II: SA updating
9:      Compute $\lambda^{(t)}$ based on (16)
10:     Compute $\zeta^{(t)}$ based on (17) and (18)
11: end for
```

Fig. 1. Augmented stochastic approximation (AugSA)

is also that of λ under model (2), satisfying (6). Moreover, for fixed λ, the MLE of ζ_j under model (8) is exactly the true value $\zeta_j^*(\lambda)$, satisfying

$$p(j; \lambda, \zeta) - \hat{\pi}_j = 0, \qquad j = 1, \dots, m. \tag{9}$$

where

$$p(j; \lambda, \zeta) = \frac{\hat{\pi}_j e^{-\zeta_j + \zeta_j^*(\lambda)}}{\sum_{l=1}^m \hat{\pi}_l e^{-\zeta_l + \zeta_l^*(\lambda)}}$$

is the marginal probability of dimension j under (8). This result can also be verified by solving Eq. (9) directly. There are only $m-1$ linearly independent equations in (9) because $\sum_{j=1}^m p(j; \lambda, \zeta) = \sum_{j=1}^m \hat{\pi}_j \equiv 1$, and so are only $m-1$ unknowns $(\zeta_2, \dots, \zeta_m)$ with $\zeta_1 = 0$ for fixed λ.

In summary, the MLE of λ (denoted as $\hat{\lambda}$) and the corresponding log ratios of normalizing constants $\zeta_j^*(\hat{\lambda})$, $j = 1, \dots, m$, are determined jointly from (6) and (9). Both equations (6) and (9) have the form of equating empirical expectations, $\tilde{p}[f]$ and $\hat{\pi}_j$, with theoretical expectations, $p_\lambda[f]$ and $p(j; \lambda, \zeta)$, for the ordinary features $f(\cdot)$ and the dimension features $\delta(\cdot)$ respectively, as similarly found in maximum likelihood estimation of single random field models [2].

4.2 Stochastic approximation

Stochastic approximation (SA) was first studied in [16]. There has since been a vast literature on theory, methods, and applications of SA [17], [18]. Here we develop an augmented SA algorithm in the SA framework to simultaneously address the two learning objectives described in Section 4.1.

We first give a brief review of the SA framework. Suppose that the objective is to find the solution θ^* of $h(\theta) = 0$ with

$$h(\theta) = E_{Y \sim g(\cdot;\theta)}[H(Y; \theta)],$$

where $\theta \in R^d$ is a parameter vector of dimension d, and Y is an observation from a probability distribution $g(\cdot; \theta)$ depending on θ, and $H(Y; \theta) \in R^d$ is a function of Y. For some initial values θ_0 and Y_0, a general SA algorithm is as follows.

Stochastic approximation (SA):

1) Generate $Y^{(t)} \sim K(Y^{(t-1)}, \cdot)$, where $K(\cdot, \cdot)$ is a Markov transition kernel that admits $g(\cdot; \theta)$ as the invariant distribution.
2) Set $\theta^{(t)} = \theta^{(t-1)} + \gamma_t A H(Y^{(t)}; \theta^{(t-1)})$, where γ_t is the learning rate (also called gain rate) and A is an invertible matrix (called gain matrix).

During each SA iteration, it is possible to generate a set of multiple observations Y by performing the Markov transition repeatedly (instead of only one time) and then use the average of the corresponding values of $H(Y; \theta)$ for updating θ, as shown in the following algorithm. This technique can help reduce the fluctuation due to slow-mixing of Markov transitions.

SA with multiple moves:

1) Set $Y^{(t,0)} = Y^{(t-1,K)}$. For k from 1 to K, generate $Y^{(t,k)} \sim K(Y^{(t,k-1)}, \cdot)$, where $K(\cdot, \cdot)$ is a Markov transition kernel that admits $g(\cdot; \theta)$ as the invariant distribution.
2) Set $\theta^{(t)} = \theta^{(t-1)} + \gamma_t A \{\frac{1}{K} \sum_{Y \in B^{(t)}} H(Y; \theta^{(t-1)})\}$, where $B^{(t)} = \{Y^{(t,k)} | k = 1, \cdots, K\}$.

The convergence of SA has been studied under various regularity conditions [17], [18]. For $\gamma_t = \gamma_0 / t^\beta$ with $\gamma_0 > 0$ and $1/2 < \beta \le 1$, it can be shown that $\gamma_t^{-1/2}(\theta^{(t)} - \theta^*)$ converges in law as $t \to \infty$ to a multivariate normal distribution $N(0, \Sigma)$. The maximal rate of variance reduction is reached with $\beta = 1$. Moreover, if $\beta = 1$, then Σ achieves a minimum at $\gamma_t = 1/t$ and $A = -\{\partial h(\theta^*)/\partial \theta\}^{-1}$, resulting in an optimal SA recursion:

$$\theta^{(t)} = \theta^{(t-1)} - \frac{1}{t} \left\{ \frac{\partial h}{\partial \theta}(\theta^*) \right\}^{-1} H(Y^{(t)}; \theta^{(t-1)}), \tag{10}$$

which is reminiscent of Newton's update for root-finding. However, the optimal SA recursion (10) is, in general, infeasible because $\partial h(\theta^*)/\partial \theta$ is unknown. There are broadly two approaches to achieving asymptotic efficiency. One is to estimate both θ^* and $\partial h(\theta^*)/\partial \theta$ by stochastic approximation (e.g. [28]). Another approach is to use the trajectory average $\bar{\theta}^{(t)} = t^{-1} \sum_{i=1}^t \theta^{(i)}$, but set the learning rate $\gamma_t = \gamma_0 / t^\beta$ decreasing more slowly than $O(t^{-1})$ for $1/2 < \beta < 1$ [32]. While a variation of the first approach was explored in our previous work [21] (see Section 5.2), the second approach was not found to perform satisfactorily in our preliminary experiments.

We exploit several new ideas in developing SA algorithms for learning with TRF. To start, we recast (6) and (9) jointly as a system of simultaneous equations in the form of $h(\theta) = 0$, with $\theta = (\lambda, \zeta_{(1)})$, $Y = (j, x^j)$, $g(\cdot; \theta) = p(j, x^j; \lambda, \zeta)$ in (8), and

$$H(Y; \theta) = \left(\begin{array}{c} \tilde{p}[f] - f(x^j) \\ \delta_{(1)}(j) - \pi_{(1)} \end{array} \right), \tag{11}$$

where $\zeta_{(1)} = (\zeta_2, \dots, \zeta_m)^T$, $\pi_{(1)} = (\pi_2, \dots, \pi_m)^T$ and $\delta_{(1)}(j) = (\delta_2(j), \dots, \delta_m(j))^T$ with $\delta_l(j) = 1(j = l)$, defined as 1 if $j = l$ and 0 otherwise (see Appendix B). Then the general SA algorithm can be used to solve (6) and (9) jointly. In the following, we present two further developments, regarding the choices of design distribution $g(\cdot; \theta)$ and disturbance $H(Y; \theta)$ to improve convergence and regading the choice of gain matrix A to approximate the optimal SA recursion (10).

First, we provide more general choices of $g(\cdot; \theta)$ and $H(Y; \theta)$ than (8) and (11) above, such that the system of equations $h(\theta) = 0$ remains equivalent to (6) and (9) jointly.

Proposition 3. *Let $\pi^0 = (\pi_1^0, \dots, \pi_m^0)^T$ be fixed proportions, strictly between 0 and 1 and satisfying $\sum_{j=1}^m \pi_j^0 = 1$. Define the following distribution on the space $\mathcal{X}$:*

$$q(j, x^j; \lambda, \zeta) \propto (\pi_j^0 e^{-\zeta_j}) e^{\lambda^T f(x^j)}. \tag{12}$$

Then (6) *and* (9) *are jointly equivalent to*

$$\tilde{p}[f] - q_{\lambda,\zeta}[\frac{\pi}{\pi^0} f] = 0, \tag{13}$$

$$q_{\lambda,\zeta}[\delta_{(1)}] - \pi_{(1)}^0 = 0, \tag{14}$$

where $q_{\lambda,\zeta}[\frac{\pi}{\pi^0} f]$ is the expectation of $\frac{\pi_j}{\pi_j^0} f(x^j)$ and $q_{\lambda,\zeta}[\delta_{(1)}]$ is that of $\delta_{(1)}(j)$ under the joint distribution (12)*. Eqs.* (13)–(14) *can be expressed in the form of $h(\theta) = 0$, with $g(\cdot;\theta) = q(j, x^j; \lambda, \zeta)$ and*

$$H(Y;\theta) = \begin{pmatrix} \tilde{p}[f] - \frac{\pi_j}{\pi_j^0} f(x^j) \\ \delta_{(1)}(j) - \pi_{(1)}^0 \end{pmatrix}, \tag{15}$$

where $\pi_{(1)}^0 = (\pi_2^0, \dots, \pi_m^0)^T$.

Proof. See Appendix B. □

Proposition 3 represents an importance sampling extension of our previous SA algorithm [21], in allowing the use of dimension probabilities different from the empirical ones when generating the observation (j, x^j) in the SA algorithm. In the special case where $\pi_j^0 = \pi_j$ for $j = 1, \dots, m$, then the choices (12) and (15) for $g(\cdot;\theta)$ and $H(Y;\theta)$ reduce to (8) and (11) earlier. See Section 4.3 for MCMC sampling from $q(j, x^j; \lambda, \zeta)$, and Section 5.3 for further discussion on the choice of π^0.

Second, we discuss the choice of gain matrix A as an approximation of $-\{\partial h(\theta^*)/\partial\theta\}^{-1}$ in the optimal SA recursion (10). For simplicity, we take A as a block-diagonal matrix with diagonal blocks (A_λ, A_ζ), partitioned according to $\theta = (\lambda, \zeta_{(1)})$. For the update of λ, we set A_λ^{-1} to a diagonal matrix, $\sigma = diag(\sigma_1, \cdots, \sigma_d)$, as a diagonal approximation to $\partial p_\lambda[f]/\partial\lambda|_{\lambda=\lambda^*}$, where σ_i is the *stratified* empirical variance of feature f_i for $i = 1, \dots, d$:

$$\sigma_i = \sum_{j=1}^m \frac{n_j}{n} \tilde{p}_j \left[(f_i - \tilde{p}_j[f_i])^2\right].$$

See Section 5.2 for derivation and further discussion. The resulting update of λ at iteration t is then

$$\lambda^{(t)} = \lambda^{(t-1)} + \gamma_{\lambda,t}\sigma^{-1} \left\{ \tilde{p}[f] - \frac{1}{K} \sum_{(j,x^j)\in B^{(t)}} \frac{\pi_j}{\pi_j^0} f(x^j) \right\}, \tag{16}$$

where $\gamma_{\lambda,t}$ is the learning rate of λ (see Section 5.1). For the update of ζ, we provide a trans-dimensional extension of the optimal SA recursion in self-adjusted mixture sampling [29], which is simple and feasible in spite of the fact that such an optimal scheme is generally infeasible as mentioned above.

Proposition 4. *For fixed λ, Eq.* (14) *can be expressed in the form of $h(\zeta_{(1)}) = 0$ with $g(\cdot;\zeta_{(1)}) = q(j, x^j; \lambda, \zeta)$ and $H(Y;\zeta_{(1)}) = \delta_{(1)}(j) - \pi_{(1)}^0$. The optimal gain matrix, $A_\zeta = -\{\frac{\partial}{\partial\zeta_{(1)}} h(\zeta_{(1)})\}^{-1}|_{\zeta=\zeta^*(\lambda)}$, can be shown to satisfy*

$$A_\zeta\, H(Y;\zeta_{(1)}) = \left(\frac{\delta_2(j)}{\pi_2^0}, \dots, \frac{\delta_m(j)}{\pi_m^0}\right)^T - \frac{\delta_1(j)}{\pi_1^0}.$$

Proof. The result follows from the proof of Theorem 1 in [29]. □

By Proposition 4, the resulting update of ζ at iteration t is

$$\zeta^{(t-\frac{1}{2})} = \zeta^{(t-1)} + \gamma_{\zeta,t} \left\{ \frac{\delta_1(B^{(t)})}{\pi_1^0}, \cdots, \frac{\delta_m(B^{(t)})}{\pi_m^0} \right\}^T, \tag{17}$$

$$\zeta^{(t)} = \zeta^{(t-\frac{1}{2})} - \zeta_1^{(t-\frac{1}{2})}, \tag{18}$$

where $\zeta_1^{(t)}$, the first element of $\zeta^{(t)}$, is set to 0 by (18), and $\gamma_{\zeta,t}$ is the learning rate of ζ (see Section 5.1), and $\delta_l(B^{(t)})$ is the proportion of dimension l appearing in $B^{(t)}$:

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

We summarize the augmented SA in Fig.1. At the beginning (Step 1), we set λ as a zero vector and we can calculate the true normalizing constants because $q(j, x^j; \lambda, \zeta)$ is a uniform distribution. At each iteration, we first generate K observations by the trans-dimensional mixture sampling described in Section 4.3 (Step 3 to 8), and then update the parameters λ and log ratios of normalizing constants ζ (Step 9 and 10).

4.3 Trans-dimensional mixture sampling

We describe a trans-dimensional mixture sampling algorithm to simulate from the joint distribution $q(j, x^j; \lambda, \zeta)$ in (12), which is used with $(\lambda, \zeta) = (\lambda^{(t-1)}, \zeta^{(t-1)})$ at time t for MCMC sampling in AugSA (Fig.1). The name "mixture sampling" reflects the fact that $q(j, x^j; \lambda, \zeta)$ represents a labeled mixture [29], because j is a label indicating that x^j is associated with the distribution $p_j(x^j; \lambda)$. With fixed (λ, ζ), this sampling algorithm can be seen as formally equivalent to reversible jump MCMC [30], 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 dimensions and Markov move of observations for a given dimension. In the following, we denote by $j^{(t-1)}$ and $x^{(t-1)}$ the dimension and observation before the current 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 dimension. Assuming $j^{(t-1)} = k$, first we draw a new dimension $j \sim \Gamma(k, \cdot)$. The jump distribution $\Gamma(k, j)$ is defined to be uniform in the neighborhood of k:

$$\Gamma(k, j) = \begin{cases} \frac{1}{3}, \text{ if } k \in [2, m-1], j \in [k-1, k+1] \\ \frac{1}{2}, \text{ if } k = 1, j \in [1, 2] \text{ or } k = m, j \in [m-1, m] \\ 0, \text{ otherwise} \end{cases} \tag{19}$$

where m is the maximum dimension. Eq. (19) restricts the difference between j and k to be no more than one. If $j = k$, we retain the observation $x^{(t-1)}$ and perform the next step directly, i.e. set $j^{(t)} = k$ 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 u from a proposal distribution: $u \sim g_{k+1}(\cdot|x^{(t-1)})$. Then we set $j^{(t)} = j\ (= k+1)$ and $x^{(t)} = \{x^{(t-1)}, u\}$ with probability

$$\left\{ 1, \frac{\Gamma(j, k)}{\Gamma(k, j)} \frac{q(j, \{x^{(t-1)}, u\}; \lambda, \zeta)}{q(k, x^{(t-1)}; \lambda, \zeta) g_{k+1}(u|x^{(t-1)})} \right\}, \tag{20}$$

where $\{x^{(t-1)}, u\}$ denotes a vector with dimension $k + 1$ whose first k elements are $x^{(t-1)}$ and the last element is u.

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

$$\left\{1, \frac{\Gamma(j,k)}{\Gamma(k,j)} \frac{q(j, x^{(t-1)}_{1:j}; \lambda, \zeta) g_k(x^{(t-1)}_k | x^{(t-1)}_{1:j})}{q(k, x^{(t-1)}; \lambda, \zeta)}\right\}, \qquad (21)$$

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

$$g_{k+1}(u|x^k) = \frac{q(k+1, \{x^k, u\}; \lambda, \zeta)}{\sum_w q(k+1, \{x^k, w\}; \lambda, \zeta)}. \qquad (22)$$

where x^k is a vector of dimension k, e.g. $x^k = x^{(t-1)}$ in (20) and $x^{k-1} = x^{(t-1)}_{1:j}$ in (21). In our experiments, the acceptance rate of local jumps is around 30%, suggesting that good mixing is achieved by the jump process.

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

$$x^{(t)} = \begin{cases} x^{(t-1)} & \text{if } j^{(t)} = k \\ \{x^{(t-1)}, u\} & \text{if } j^{(t)} = k+1 \\ x^{(t-1)}_{1:k-1} & \text{if } j^{(t)} = k-1 \end{cases} \qquad (23)$$

Then we perform (for example) Gibbs sampling on $x^{(t)}$, from the first element to the last element. The pseudo-code of trans-dimensional mixture sampling is shown in Algorithm 1 in Tab.1.

5 Algorithm Optimization

The augmented SA algorithm may still suffer from slow convergence, especially when λ is high-dimensional. In this section, we introduce several statistical and computational techniques to improve the convergence of the augmented SA algorithm and reduce the computational cost.

5.1 Learning rates

As mentioned in Section 4.2, setting $\gamma_t = O(t^{-1})$ can achieve asymptotic convergence of the SA algorithm. In practice, to accelerate the initial convergence and improve stability, we introduce a two-stage configuration of the learning rates, following the suggestion of [28] and [29].

We set the learning rates as

$$\gamma_{\lambda,t} = \begin{cases} (t_c + t^{\beta_\lambda})^{-1} & \text{if } t \le t_0 \\ (t_c + t - t_0 + t_0^{\beta_\lambda})^{-1} & \text{if } t > t_0, \end{cases} \qquad (24)$$

$$\gamma_{\zeta,t} = \begin{cases} t^{-\beta_\zeta} & \text{if } t \le t_0 \\ (t - t_0 + t_0^{\beta_\zeta})^{-1} & \text{if } t > t_0, \end{cases} \qquad (25)$$

where $0.5 < \beta_\lambda, \beta_\zeta < 1$, t_0 is an burn-in size, and t_c is an offset constant for the learning rate of λ. These choices are simpler than in our previous work [21] primarily because a different technique for rescaling SA update in λ is used (see Section 5.2). In the first stage ($t \le t_0$), a slow-decaying rate of $t^{-\beta}$ is used to introduce large adjustments for $0.5 < \beta < 1$. This forces the estimates $\lambda^{(t)}$ and $\zeta^{(t)}$ to fall reasonably fast to near the true values. In the second stage ($t > t_0$), a fast-decaying rate of t^{-1} is used according to asymptotic theory of SA. The offset t_c is introduced to avoid the learning rate of λ being too large in the first several SA iterations. Typically, t_0 is selected to ensure that there is no more significant adjustment of the log-likelihood observed on the development set in the first stage.

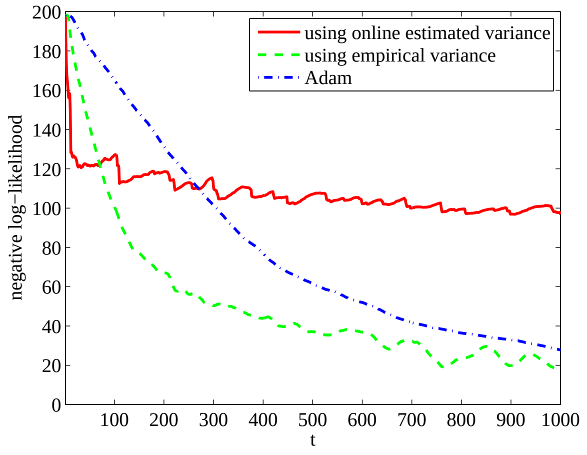


Fig. 2. To compare the convergence of SA training using online estimated variances in [21] (green dash line) and using empirical variances in this paper (red solid line), and Adam method [22] (blue dot line), we plot an example of the scaled negative log-likelihoods on PTB training set along the iterations.

5.2 Rescaling SA update in λ

In (16), we introduce a diagonal matrix σ to rescale the update direction for λ. Here we discuss the rationale for this technique.

For simplicity, consider (13) with only λ unknown and $\zeta = \zeta^*(\lambda)$, which leads back to (6) and can be expressed in the form of $h(\lambda) = 0$ with

$$h(\lambda) = \tilde{p}[f] - p_\lambda[f] = \tilde{p}[f] - \sum_{j=1}^m \frac{n_j}{n} p_{\lambda,j}[f].$$

As mentioned in Section 4.2, the optimal gain matrix A should be $A^{-1} = -\partial h(\lambda^*)/\partial\lambda$. By direct calculation, we find

$$C(\lambda) = -\frac{\partial h(\lambda)}{\partial \lambda} = \sum_{j=1}^m \pi_j \{p_{\lambda,j}[ff^T] - p_{\lambda,j}[f] p_{\lambda,j}[f]^T\},$$

where $p_{\lambda,j}[ff^T] = \sum_{x^j \in \mathcal{X}^j} p_j(x^j;\lambda) f(x^j) f^T(x^j)$. In general, $C(\lambda^*)$ is unknown and needs to be approximated. One method is to estimate $C(\lambda^*)$ along with λ^* by stochastic approximation [28], as used in our previous paper [21]. In this method, the diagonal elements of matrix $C(\lambda^*)$ are estimated concurrently with λ^* during SA iterations and a threshold is introduced to censor the estimates from below[6]. Alternatively, we propose the use of empirical variances to approximate $C(\lambda^*)$. In fact, the empirical *stratified* covariance matrix is

$$\tilde{C} = \sum_{j=1}^m \frac{n_j}{n} \{\tilde{p}_j[ff^T] - \tilde{p}_j[f]\tilde{p}_j[f]^T\},$$

where $\tilde{p}_j[f^T f] = n_j^{-1} \sum_{x^j \in D_j} f(x^j) f(x^j)^T$. If the fitted distribution $p(j, x^j; \lambda^*)$ is close to the empirical distribution $\tilde{p}(j, x^j)$, then $\tilde{C}$ provides a reasonable approximation of $C(\lambda^*)$. Taking the diagonal elements of $\tilde{C}$ yields the diagonal matrix σ used to obtain the update (16) for λ.

The use of empirical variances to rescale the gradient can also be understood from another perspective. For training log-linear models, it has been shown that the convergence of gradient-based

6. In our previous paper [21], only about 20% of the components of λ are larger than the threshold 10^{-4}.

TABLE 1
Trans-dimensional mixture sampling (TransMS).
The algorithm 1 is the normal version introduced in section 4.3, and the algorithm 2 is the class-based version introduced in Appendix C

Algorithm 1

```
1: function SAMPLE ($j^{(t-1)}, x^{(t-1)}$)
2:     set $k = j^{(t-1)}$
3:     set $(j^{(t)}, x^{(t)}) = (j^{(t-1)}, x^{(t-1)})$
       Step I: Local jump
4:     generate $j \sim \Gamma(k, \cdot)$ by (19)
5:     if $j = k + 1$ then
6:
7:         generate $u \sim g_{k+1}(\cdot|x^{(t-1)})$ by (22)
8:         set $j^{(t)} = j$ and $x^{(t)} = \{x^{(t-1)}, u\}$ with probability (20)
9:     end if
10:    if $j = k - 1$ then
11:        set $j^{(t)} = j$ and $x^{(t)} = x^{(t-1)}_{1:j}$ with probability (21)
12:    end if
       Step II: Markov move
13:    for $i = 1 \to j^{(t)}$ do
14:
15:
16:        draw $u \sim q(j^{(t)}, \{x^{(t)}_{1:i-1}, \cdot, x^{(t)}_{i+1:j^{(t)}}\}; \lambda, \zeta)$ by (12)
17:        set $x^{(t)}_i = u$
18:    end for
19:    return $(j^{(t)}, x^{(t)})$
20: end function
```

Algorithm 2

```
1: function FASTSAMPLE ($j^{(t-1)}, x^{(t-1)}$)
2:     set $k = j^{(t-1)}$
3:     set $(j^{(t)}, x^{(t)}) = (j^{(t-1)}, x^{(t-1)})$
       Step I: Local jump
4:     generate $j \sim \Gamma(k, \cdot)$ by (19)
5:     if $j = k + 1$ then
6:         generate $c_0 \sim Q_{k+1}(c)$
7:         generate $u \sim \breve{g}_{k+1}(\cdot|x^{(t-1)}, c_0)$ by (35)
8:         set $j^{(t)} = j$ and $x^{(t)} = \{x^{(t-1)}, u\}$ with probability (20) and (36)
9:     end if
10:    if $j = k - 1$ then
11:        set $j^{(t)} = j$ and $x^{(t)} = x^{(t-1)}_{1:k-1}$ with probability (21) and (37)
12:    end if
       Step II: Markov move
13:    for $i = 1 \to j^{(t)}$ do
14:        draw $c_0 \sim Q_i(c)$
15:        set $c^{(t)}_i = c_0$ with probability (34)
16:        draw $u \sim q(j^{(t)}, \{x^{(t)}_{1:i-1}, \cdot, x^{(t)}_{i+1:j^{(t)}}\}; \lambda, \zeta)$ by (12) with $u \in \mathcal{A}_{c^{(t)}_i}$
17:        set $x^{(t)}_i = u$
18:    end for
19:    return $(j^{(t)}, x^{(t)})$
20: end function
```

optimization algorithms can be accelerated by normalizing the means and variances of features [33]. In fact, the update (16) can be equivalently expressed as

$$\sigma^{1/2}\lambda^{(t)} = \sigma^{1/2}\lambda^{(t-1)} + \gamma_{\lambda,t}\left\{\tilde{p}[g] - \frac{1}{K}\sum_{(j,x^j)\in B^{(t)}} \frac{\pi_j}{\pi_j^0} g(x^j)\right\},$$

where $g(x^j) = \sigma^{-1/2} f(x^j)$ is the normalized feature vector such that the variances of individual features are 1 and $\sigma^{1/2}\lambda$ is the corresponding parameter vector for g.

Compared with estimating $C(\lambda^*)$ concurrently with λ^* during SA iterations in our previous work [21], using empirical variances to rescale the update direction for λ has two advantages. First, memory cost and computational time are reduced, because σ can be calculated offline for one time. Second, the empirical variances σ_i can be censored from below by a very small threshold, so that almost all of the components of λ can be updated without modification of the empirical variances. In our experiments, the threshold is 10^{-15} and 99.5% of the components of λ are updated with non-censored empirical variances. This technique is found to improve the convergence of the SA training algorithm, as shown in Fig.2. It is worthwhile to compare our method with related adaptive stepsize optimisers, such as Adam [22] and RMSProp [34]. They also scale the gradient with curvature information but by applying different preconditioners. As shown in Fig.2, our method of preconditioning using the empirical variances of features in our language modeling experiments shows better convergence than Adam which uses a running average of squared gradients for preconditioning.

5.3 Dimension probabilities π^0

Compared with our previous work [21], another new aspect of the augmented SA algorithm is to allow the dimension probabilities π_j^0 used in the trans-dimensional mixture sampling to differ from the empirical dimension probabilities $\pi_j = n_j/n$, in order to overcome limitations caused by setting $\pi_j^0 = \pi_j$. The sample size n_j from the training set may be zero for some j, in which case the augmented SA algorithm will fail with $\pi_j^0 = 0$ because ζ_j will not be updated without any observation of dimension j sampled. Moreover, when π_j^0 is too small for some j, the estimation of ζ_j would become problematic because the update scheme (17) involves dividing a fraction $\delta_j(B^{(t)})$ by π_j^0.

For illustration, imagine that π_j^0 is 10^{-5} for some j, and $K = 300$ observations are drawn at each iteration. If the dimension j appears at least once in sample set $B^{(t)}$, then $\delta_j(B^{(t)}) \geq 1/300$. Then the update to ζ_j will be at least $300^{-1}/10^{-5} = 1000/3$ in (17) (ignoring the learning rate $\gamma_{\zeta,t}$), which can be much larger than the true value of ζ_j. Moveover, if $j = 1$, this will influence the updates of all $(\zeta_2, \ldots, \zeta_m)$ in (18). To prevent such heavy fluctuations, we need to avoid setting π_j^0 too small and particularly to set π_1^0 reasonable large.

In our experiments, we set

$$\pi_j^0 = \frac{\max\{u_j, c\}}{\sum_{j=1}^m \max\{u_j, c\}}, \tag{26}$$

where c is a constant (set to 10^{-5} in our experiments), and

$$u_j = \begin{cases} n_{max}/n, & \text{if } j \leq j_{max} \\ n_j/n, & \text{if } j > j_{max} \end{cases}$$

with $n_{max} = \max_j n_j$ and $j_{max} = \arg\max_j n_j$. The motivation for introducing u_j above is to make π_j^0 large for $j = 1$ and for subsequent dimensions j, because the trans-dimensional mixture sampling algorithm (Tab.1) only permits a dimension jump from j to $j \pm 1$. In the experiments, we find that this choice of the dimension probabilities π_j^0 helps to considerably improve the accuracy in the estimation of ζ.

5.4 Modeling rare high-dimensional observations

Compared with our previous work [21], we propose still another technique to deal with rare very high-dimensional observations (e.g. rare very long sequences), which are often found in large datasets. For example, the maximum sentence length in Google 1-billion word corpus [35] is more than 1000, but the percentage of sentences longer than 100 is about 0.2% in the datasets used in our experiments (Section 8). To reduce the number of sub-models corresponding to different dimensions, we set the near maximum dimension m to a medium value (such as 100) and modify the joint distribution (8) by introducing a special sub-model that captures all the sequences longer than m as follows:

$$p(j, x^j; \lambda, \zeta) \propto \frac{n_{m+}/n}{e^{\zeta_{m+}}} e^{\lambda^T f(x^j)}, \quad j > m, \tag{27}$$

where $\zeta = (\zeta_1, \dots, \zeta_m, \zeta_{m+})^T$ with $\zeta_1 = 0$ and n_{m+} is the number of the sequences longer than m in the training set. A single log ratio of normalizing constants ζ_{m+} is used such that model (27) can be seen as model (1) but only for sequences longer than m. The augmented SA algorithm can be extended with simple modifications. The maximum dimension in trans-dimensional mixture sampling (Tab.1) is set to be slightly larger than m (specifically $m+2$ in our experiments). Then λ is still updated by (16), with $(j, x^j) \in B^{(t)}$ for both $j = 1, \dots, m$ and $j > m$. Both $(\zeta_1, \dots, \zeta_m)$ and ζ_{m+} are updated by (17) and (18), where $\delta_{m+}(B^{(t)}) = K^{-1} \sum_{(j,x^j)\in B^{(t)}} 1(j > m)$ is the proportion of dimensions greater than m in the sample $B^{(t)}$.

6 Application: Word Morphology

In this section, we perform a pilot experiment — word morphology, to exactly evaluate the performance of TRF models and AugSA training algorithms. This is similar with the experiment in [4] but we provide detailed evaluations of performances, which includes studying the convergence of the log-likelihood using the true normalizing constants, comparing the estimated normalizing constants with the true ones, and comparing the IIS presented in [4] with our AugSA training algorithm.

The objective of word morphology is to model English words as character sequences by assigning probabilities. The training and test data are extracted from the English Gigaword dataset[7]. The training set contains 139K different English words. The test set contains other 15K different English words. The maximum word lengths in the training and test sets are 25 (i.e. $m = 25$).

To construct a TRF model (2), we first extract all the character unigrams, bigrams and trigrams from the training set and define the following features:

$$f_v(x^j) = \sum_{i=1}^{j} f_{i,v}(x^j),$$
$$f_{vw}(x^j) = \sum_{i=1}^{j-1} f_{i,vw}(x^j),$$
$$f_{vwy}(x^j) = \sum_{i=1}^{j-2} f_{i,vwy}(x^j),$$

where x^j denotes the word containing j characters $(x_1^j, \dots, x_j^j)$, v, vw and vwy denote the character unigram, bigram and trigram observed in the training set, and $f_{i,v}$, $f_{i,vw}$ and $f_{i,vwy}$ are the corresponding unigram, bigram and trigram features evaluated at position i:

$$f_{i,v}(x^j) = \begin{cases} 1 & \text{if } x_i^j = v \\ 0 & \text{otherwise,} \end{cases}$$
$$f_{i,vw}(x^j) = \begin{cases} 1 & \text{if } x_i^j = v \text{ and } x_{i+1}^j = w \\ 0 & \text{otherwise,} \end{cases}$$
$$f_{i,vwy}(x^j) = \begin{cases} 1 & \text{if } x_i^j = v \text{ and } x_{i+1}^j = w \text{ and } x_{i+2}^j = y \\ 0 & \text{otherwise.} \end{cases}$$

The feature set contains f_v, f_{vw}, f_{vwy}, $f_{i=0,v}$, $f_{i=end,v}$, $f_{i=0,vw}$, and $f_{i=end,vw}$, where $f_{i=0,v}$ and $f_{i=end,v}$ denote the unigram v appearing at the beginning and the end of a word; $f_{i=0,vw}$ and $f_{i=end,vw}$ denote the bigram vw appearing at the beginning and the end of a word[8]. The total feature number $d = 11929$. We perform the augmented SA (Algorithm 1 in Fig.1)[9] to train the TRF model. At each iteration, a sample of $K = 100$ sequences (i.e. words) are generated by trans-dimensional mixture sampling (Section 4.3). We set $\pi_j^0 = n_j/n$ (denoted by "n_j/n", the red line in Fig.3(b)) or set π_j^0 as (26) (denoted by "π^0", the green line in Fig.3(b)) to study the effect of proper specification of length probabilities. The learning rates are set as (24) and (25) with $t_c = 100$, $\beta_\lambda = 0.8$, $\beta_\zeta = 0.6$, $t_0 = 200$. The maximum iteration number is fixed as $t_{max} = 1000$. The techniques introduced in Sections 5.4 and Appendices C–E are not used in this experiment.

We implement three existing methods. First is the n-gram model [6] widely used in language modeling. Here we view an English word as a character sequence and train a 3-gram model by the Witten-Bell discounting smoothing method (denoted by WB3), using the SRILM toolkit[10]. This 3-gram model contains the same set of features with the TRF models introduced above. Second, we apply the improved iterative scaling (IIS) method [4] to train the same TRF model. For comparison with our AugSA algorithms, at each iteration, we generate $K = 100$ words of random lengths[11] to estimate the model expectations used in IIS. Third, as the expectation $p_\lambda[f]$ can be calculated exactly in this pilot experiment, we apply the Quasi-Newton method to solve Eq. (6) for the TRF model (denoted by Quasi-Newton). This provides the exact result, relative to which different Monte Carlo training algorithms can be evaluated. For more complex TRFs as in language modeling in Sections 7–8, the Quasi-Newton method is infeasible because exact calculation of $p_\lambda[f]$ is intractable.

To study the performances of different methods, we first plot in Fig.3(a) the scaled negative log-likelihood (NLL) curves, exactly calculated using the true normalizing constants associated with the parameter estimates during iterations by AugSA and IIS algorithms for TRF. The NLLs at the final parameter estimates by Quasi-Newton for TRF and by WB3 for n-gram are also shown as flat lines. Several interesting points can be drawn.

First, IIS (blue solid line) performs poorly in this experiment, with NLL diverging from the exact result (black dot line) obtained by Quasi-Newton. This may be due to an insufficient number

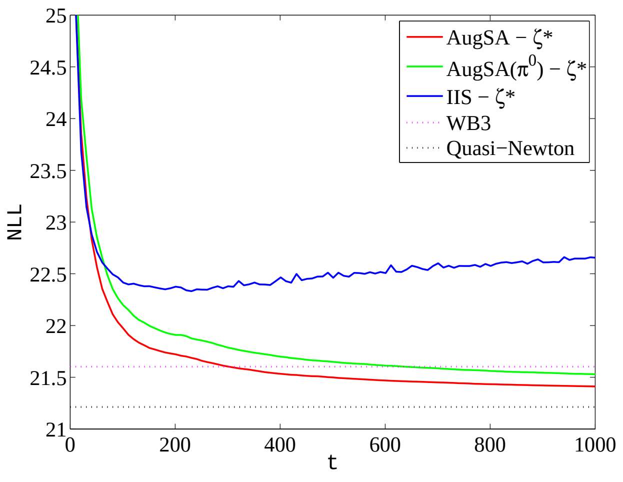
7. https://catalog.ldc.upenn.edu/LDC2003T05

8. Introducing $f_{i=0,v}$,$f_{i=end,v}$,$f_{i=0,vw}$,$f_{i=end,vw}$ is to make TRF models contain the same set of features as the 3-gram model in [6] does.

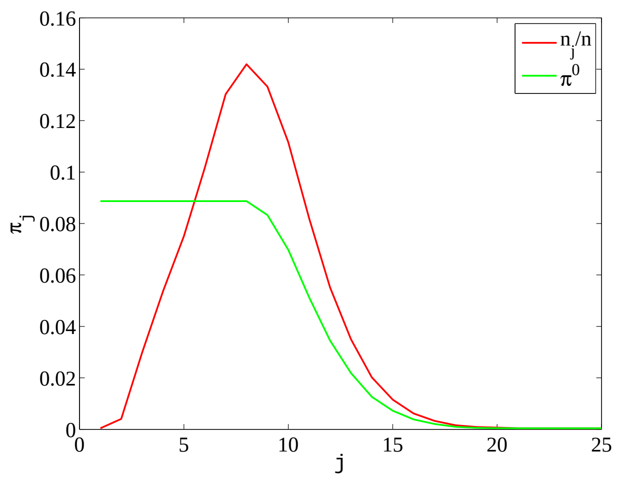
9. In this experiment, there is no class information and there is no need to accelerate the sampling using Algorithm 2 in Fig.1 because the alphabet set $\mathcal{A}$ is very small (26 characters).

10. www.speech.sri.com/projects/srilm/

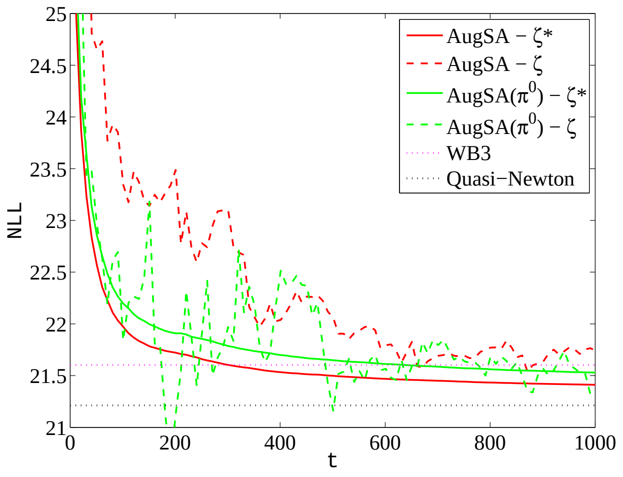
11. For sampling in IIS, we first draw length j with the empirical length probability n_j/n. Then Gibbs sampling is performed to generate a sequence of length j, initialized from the last sequence of the same length.



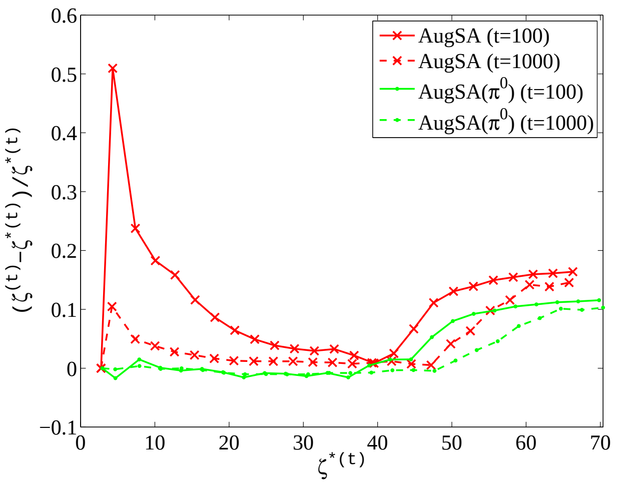
(a) NLL on training set



(b) length probabilities



(c) NLL on training set



(d) $(\zeta^{(t)} - \zeta^{*(t)})/\zeta^{*(t)}$ against $\zeta^{*(t)}$

Fig. 3. Word morphology experiment results. (a)(c) plot the scaled negative log-likelihoods (NLLs) on the training set during iterations or at the final parameter estimates. (b) shows the distribution of n_j/n and π^0 as discussed in Section 5.3. (d) plots the relative difference of the estimated log ratios of normalizing constants $\zeta^{(t)}$ and the true ones $\zeta^{*(t)}$ associated with $\lambda^{(t)}$ at $t = 100$ and 1000. The legends are interpreted as follows:

- "AugSA" denotes the augmented SA with $\pi_j^0 = n_j/n$.
- "AugSA(π^0)" denotes the augmented SA with π^0 set as described in Section 5.3.
- "IIS" denotes the improved iterative scaling (IIS) algorithm.
- "WB3" denotes the 3-gram models with Witten-Bell discounting smoothing.
- "Quasi-Newton" denotes the Quasi-Newton method using the exact gradients.
- "ζ*" or "ζ" after hyphen indicates, respectively, that the NLL is calculated using the true normalizing constants or the estimated normalizing constants associated with the current parameter estimates.

($K = 100$) of draws used for the estimation of the expectation $p_\lambda[f]$. In contrast, our AugSA algorithms (red solid and green solid lines) perform much better (with NLL converging steadily to the exact result), using the same number of draws per iteration as in IIS. Second, TRF models trained by AugSA algorithms yield lower NLL and hence outperform the WB3 model (red dot line), using the same set of features. [12] Third, revising length probabilities π_j^0 from n_j/n (red solid line) to (26) (blue solid line) appears to slightly slow the convergence of the log-likelihood. But such a revision leads to more accurate estimation of normalizing constants and hence of the log-likelihoods as discussed next. Reliable estimation of log-likelihoods for AugSA is important for monitoring the progress of training in practical situations where exact calculation of log-likelihoods is not feasible.

In Fig.3(c), we plot the NLL curves calculated using the true normalizing constants (solid lines) and the estimated ones (dash lines) associated with the parameter estimates during iterations by AugSA algorithms. Setting π^0 as (26) yields more accurate estimation of the log-likelihoods: the green dash line is closer to the green solid line than the red dash line to the red solid line. In fact, using $\pi_j^0 = n_j/n$ leads to systematic overestimation of NLL: the red dash line is always above the red solid line. In addition, Fig.3(d) shows that the log ratios of normalizing constants $\zeta^{(t)}$ estimated by "AugSA(π^0)" (green lines) are closer to the true $\zeta^{*(t)}$ than the $\zeta^{(t)}$ estimated by "AugSA" (red lines) to $\zeta^{*(t)}$ at, for instance, iteration $t = 100$ and 1000.

12. The log-likelihoods on the test set are virtually the same as those on the training set for all the methods studied. This is presumably because the training and test sets are sufficiently matched in this experiment.

TABLE 2
Feature definition in TRF LMs

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	$(c_{-3}c_{-2}c_{-1}w_0)$ $(c_{-2}c_{-1}w_0)(c_{-1}w_0)$
tied	$(c_{-9:-6}, c_0)$ $(w_{-9:-6}, w_0)$

7 Experiments: Language Modeling on PTB corpus

In this section, we apply TRF models for language modeling. We evaluate the performances of different LMs by perplexity[13] (PPL) which is a widely used measure of a LM and by speech recognition word error rate (WER) which quantifies the usefulness of applying a LM in speech recognition. Compared with our pervious work [21], the new results reported in this section mainly include: (1) introducing a novel feature type, the tied long skip-bigram feature "tied" [36], which further demonstrates the flexibility of TRF modeling to use a richer set of features, (2) re-running the experiments with the newly developed techniques described in Sections 5.2, 5.3 and Appendix E to train TRF LMs, (3) obtaining AIS based estimates of normalizing constants for TRF models, and (4) providing comparisons with LSTM LMs in addition to n-gram and RNN LMs.

The training and development datasets are from the Wall Street Journal (WSJ) portion of Penn Treebank (PTB) corpus. In accordance with previous studies [8], 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. The baseline is n-gram LMs with modified Kneser-Ney smoothing [6], denoted by KNn, which are trained by the SRILM toolkits. We use the RNNLM toolkit[14] to train the RNN LM; the number of hidden units is 250[15]. We use the open-source toolkit provided by [20] to train the LSTM LM, which constains 2 hidden layers and each layer contains 250 units[16]. The hyperparameters are tuned to our best through a series of pilot experiments.

For our TRF models, we consider a variety of features as shown in Tab.2, mainly based on word and class information. Each word is deterministically assigned to a single class, by running the automatic clustering algorithm proposed in [37] on the training dataset. In Tab.2, $w_i, c_i, i = 0, -1, \ldots, -5$ denote the word and its class at different position offset i, e.g. w_0, c_0 denotes the current word and its class. The word/class n-gram features "w"/"c", skipping n-gram features "ws"/"cs" [38], higher-order skipping features "wsh"/"csh", and the crossing features "cpw" (meaning class-predict-word) are introduced in [21]. In this paper, we further introduce the tied long-skip-bigram features "tied" [36], in which the skip-bigrams with skipping distances from 6 to 9 share the same parameter. In this way we can leverage long distance contexts without increasing the model size. Note that all the features $f(x)$ in model (2) are constructed from Tab.2 in a position-independent and dimension-independent manner, and only the features observed in the training data are used.

13. The perplexity of a dataset is defined to be the exponential of the mean negative log-likelihood of the dataset per-token.

14. http://rnnlm.org/

15. 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 [8]. RNNLMs were experimented with varying numbers of hidden units (100-500). The best result from using 250 hidden units is reported.

16. No dropout is used, since overfitting is not observed in training LSTM LMs in this experiment.

TABLE 3
The WERs, NLLs and PPLs on the WSJ'92 test data. 10 TRF models are trained by 10 independent AugSA runs. The means ± standard deviations are shown over the 10 TRFs, with four different estimates of the normalizing constants. In addition, the results for one of the 10 TRFs are shown. Sep Std/Sm AIS denotes Separate Standard/Smoothed AIS. "#feat" denotes the feature size (million).

Model	WER (%)	NLL	PPL	#feat
KN5	8.78	97.9	294.4	2.3
RNN [8]	7.90	95.6	257.6	5.1
LSTM [9] [20]	7.87	98.6	306.3	6.0
The best TRF (200 classes) in the previous work [21], using features "w+c+ws+cs+cpw"				
AugSA	7.92	∼96	∼265	6.4
10 TRFs (200 classes) trained from 10 independent runs in this work, using features "w+c+ws+cs+wsh+csh+tied"				
AugSA	7.90±0.07	95.3±0.5	252.7±7.3	7.7
Sep Std AIS	8.33±0.23	102.3±1.4	380.9±32.4	7.7
Sep Sm AIS	7.89±0.09	102.2±1.0	378.9±22.5	7.7
Global AIS	7.90±0.07	116.2±10.3	1066.4±889.5	7.7
Results for one of the 10 TRFs above				
AugSA	8.03	95.1	250.4	7.7
Sep Std AIS	8.41	103.7	412.4	7.7
Sep Sm AIS	8.02	103.3	402.9	7.7
Global AIS	8.03	122.5	1225.7	7.7
Model interpolation with $\alpha = 0.5$				
RNN+KN5	8.00			
RNN+TRF	7.71±0.06			
LSTM+KN5	8.09			
LSTM+TRF	7.59±0.06			

The AugSA (Algorithm 2 in Fig.1) is used to train the TRF models. The maximum lengths of sentences in the training, development and test sets are 82, which is not very large. Hence the technique described in Section 5.4 is not needed. All the other techniques described in Section 5 are applied to train TRF LMs. At each iteration, we generate a sample of $K = 300$ sentences of lengths ranging from 1 to 82 (i.e. $m = 82$). The learning rates γ_λ and γ_ζ are configured as (24) and (25) respectively with $t_c = 3000$, $\beta_\lambda = 0.8$, $\beta_\zeta = 0.6$ and $t_0 = 2000$. The length probabilities π^0 are set as (26). We use the automatic stop decision described in Appendix E, with the L2 regularization constant $\mu = 4 \times 10^{-5}$ in (38). We calculate S_t with $H = 100$ and stop iteration if $S_t < 10^{-4}$. Eight CPU cores are used to parallelize the algorithm, as described in Appendix D. We also implement the IIS method for the language modeling experiments[17]. But the results are unsatisfactory, being similar to those in the word morphology experiment, and hence are not shown.

For evaluations in terms of speech recognition WERs, various LMs obtained using PTB training and development datasets are used to re-score 1000-best lists from recognizing WSJ'92 test data (330 utterances). For each utterance, we obtain a 1000-best list of candidate sentences by the first-pass recognition using the Kaldi

17. The IIS method is incapable of estimating normalizing constants that are needed for evaluating LMs. Similarly as in [29], we apply AugSA to estimate the log normalizing constants ζ through (17)–(18), while fixing λ at the parameter estimates obtained by IIS.

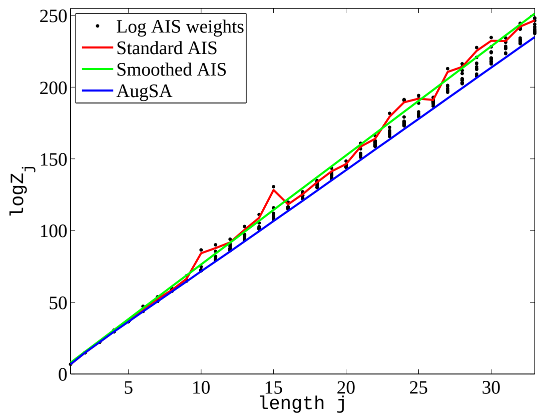


Fig. 4. Log normalization constants estimated by different methods for one of the 10 TRFs, whose WER and PPL result are shown in Tab.3. Dots represent the log weights from the 10 chains used to compute the AIS estimates (solid red line).

toolkit[18] with a DNN-based acoustic model. The oracle WER of the 1000-best lists is 0.93%.

The results on WSJ'92 test set are shown in Tab.3. As the normalizing constants for TRF models are estimated stochastically through ζ_j, we report PPLs and WERs over 10 independent AugSA training runs. The TRF LMs are compared against the KN n-gram LMs, the RNN LMs [8], the LSTM LMs [9] and the previous results in [21][19]. We provide the following comments.

First, in addition to various features used in [21], we introduce the tied skip-bigram features to further demonstrate TRF's flexibility in feature integration which may be difficult for neural network LMs. The resulting TRF LM reduce the WER to about 7.90, which outperforms the KN5 LMs significantly with 10% relative reduction. This result is close to the RNN LM and LSTM LM, which show little difference in this experiment.

Second, we examine how the TRF LMs are complimentary to neural network LMs through model interpolation. The sentence log-probabilities from two models are log-linearly combined with interpolation weights $0 < \alpha < 1$ and $1 - \alpha$ respectively. Interpolated TRF and LSTM LMs (LSTM+TRF) produces the lowest WER. Numerically, the relative reduction is 13.6% compared with the KN5 LM, and 3.2% compared with the single LSTM LM.

Third, for training cost, the single-run training times for TRF, RNN and LSTM are about 20 hours on CPUs, 25 hours on CPUs and 1 hours on GPUs respectively. For testing cost, compared with neural network LMs, TRF models are computationally more efficient in computing sentence probabilities, by simply summing up weights for the activated features in each sentence. The neural network LMs suffer from the expensive softmax computation in the output layer [20]. Empirically in our experiments, the average time costs for re-ranking the 1000-best list of candidate sentences for each utterance are 0.16 seconds, 40 seconds and 10 seconds, based on TRF (using CPU), RNN (using CPU) and LSTM (using GPU) respectively. In our experiments, the RNN and LSTM used the softmax version without classing to achieve good accuracy [8]. According to a recent paper [40], an advanced implementation using class-based softmax can achieve a speed-up from 2 to 10 times, but that would still be much slower than TRF.

Fourth, as suggested by a referee, we run the AIS method to re-estimate the normalization constants for 10 TRFs obtained from 10 independent AugSA runs. There are two approaches. Firstly, by the definition (2), each TRF involves multiple normalizing constants for different dimensions, which can be estimated by AIS separately over individual fixed-dimensional subspaces. Secondly, each TRF together with the normalizing constants estimated by AugSA can also be viewed as model (3) with a single, global normalizing constant, which can be estimated by applying AIS over the trans-dimensional state space. Due to the use of estimated normalizing constants, the two approaches are not equivalent. For the first approach, an AIS run for estimating a normalizing constant consists of 10 chains with 20K intermediate distributions, and the base distribution is set by zeroing all λ's. Since the maximum length in the 1000-best lists is 33, we estimate 33 normalization constants for lengths from 1 to 33 by 33 AIS runs for each TRF, which takes about 24 hours (even parallelizing the computation over lengths and chains using eight CPUs). Due to the independence of AIS runs, there are large oscillations in the standard AIS estimates across different lengths, as shown in Fig.4 (red curve). The resulting rescoring performance appears to be worsened by such non-smooth estimates of normalizing constants. To mitigate this defect, for each of the 10 TRFs, we apply a linear regression over the log AIS estimates at 33 sentence lengths to obtain smoothed estimates of normalizing constants, which are then used to rescore the 1000-best lists. This method, denoted by "Sep Sm AIS" in contrast with "Sep Std AIS", gives WERs close to those obtained by AugSA, albeit with substantial extra time cost (about 24 hours for each TRF). While there are some differences between the AugSA and AIS estimates of normalizing constants, it is reassuring and serves as independent validation of the results that both estimates lead to similar WERs after a correction for the non-smoothness of standard AIS estimates. For the second approach, denoted by "Global AIS", an AIS run for estimating the global normalizing constant in (3) contains 10 chains with 200K intermediate distributions, which takes about 8 hours (parallelizing over chains on ten CPUs) for each TRF. The base distribution is defined by zeroing all λ's and then setting ν based on true ζ^*. The rescoring performance is not changed by the global constant, although the log-likelihoods and PPLs are affected.

8 Experiments: Language Modeling with Training Data of up to 32 Million Words

In this section, we show the scalability of TRF LMs with the AugSA training algorithm to handle tens of millions of features. The experiments are performed on part of Google 1-billion word corpus[21]. The whole training corpus contains 99 files and each file contains about 8M words. The whole held-out corpus contains 50 files and each file contains about 160K words. In our experiments, we choose the first and second file (i.e. "news.en.heldout-00000-of-00050" and "news.en.heldout-00001-of-00050") in the held-out corpus as the development set and test set respectively, and incrementally increase the training set as used in our experiments.

First, we use one training file (about 8 million words) as the training set. We extract the 20K most frequent words from

18. http://kaldi.sourceforge.net/

19. The WSME LMs (1) using the same sets of features as the TRF LMs were also evaluated on this dataset. But the WERs and PPLs of the WSME LMs were even worse than the baseline KN5. See the numerical results and discussions in the previous work [21].

20. This deficiency could be partly alleviated with some speed-up methods, e.g. using word clustering [19] or noise contrastive estimation [39].

21. https://github.com/ciprian-chelba/1-billion-word-language-modeling-benchmark

TABLE 4
The perplexities (PPL) of various LMs with different sizes of training data (8M, 16M, 32M) from Google 1-billion word corpus. The cutoff settings of n-gram LMs are 0002 (KN4), 00002 (KN5) and 000002 (KN6). The RNN [8] and LSTM [20] are configured the same as in the PTB experiment, except RNN uses 200 classes. The feature type of TRF is "w+c+ws+cs+wsh+csh+tied". "#feat" is the feature size (million).

models	8M		16M		32M	
	PPL	#feat	PPL	#feat	PPL	#feat
KN5	128	18.1	112	33.6	99	61.8
RNN	110	9.0	99	9.0	92	9.0
LSTM	82	9.9	73	9.9	68	9.9
TRF	$\sim$110	17.5	$\sim$98	29.2	$\sim$91	48.7

TABLE 5
The WERs on WSJ'92 test set, continuing from Tab.4. The interpolation weight α is tuned from 0.1 to 0.9 per 0.1.

models	8M	16M	32M
KN5	8.24	8.26	7.66
RNN	7.47	7.73	7.73
LSTM	7.19	6.88	6.69
TRF	7.08	7.17	7.10
RNN+KN5	7.60	7.66	7.43
LSTM+KN5	7.14	6.82	6.59
RNN+TRF	7.08	7.27	6.93
LSTM+TRF	6.76	6.72	6.33

the training set to construct the lexicon[22]. Then for the training, development and test set, all the words out of the lexicon are mapped to an auxiliary token <UNK>. The word clustering algorithm in [37] is performed on the training set to cluster the words into 200 classes. We train a TRF model with feature type "w+c+ws+cs+wsh+csh+tied" (Tab.2). Then we increase the training size to around 16 million words (2 training files). We still extract the 20K most frequent words from the current training set to construct a new lexicon. After re-clustering the words into 200 classes, a new TRF model with the same feature type "w+c+ws+cs+wsh+csh+tied" is trained over the new training set. We repeat the above process and train the third TRF model on 32 million words (4 training files).

In this experiment, the maximum lengths of sentences in the training, development and test sets are more than 1000. The percentages of sentences longer than 100 are all about 0.2% in the training, development and test sets. Hence we apply the technique described in Section 5.4, with the near maximum dimension set to $m = 100$. At each iteration, we generate a sample of $K = 300$ sentences of lengths ranging from 1 to 102. The learning rates γ_λ and γ_ζ are configured as (24) and (25) respectively with $t_c = 1000$, $\beta_\lambda = 0$, $\beta_\zeta = 0.6$ and $t_0 = 2000$. The length probabilities π^0 are set as (26). In this experiment, we fix the iteration number $t_{max} = 50,000$ and the L2 regularization constants 10^{-5} to sufficiently train our TRF LMs. Twelve CPU cores are used to parallelize the training algorithm.

The perplexities on the test set are shown in Tab.4. The WERs obtained from applying various LMs to rescore the WSJ'92 1000-best list are shown in Tab.5. These results show that the TRF LMs can scale to using training data of up to 32 million words. The TRF LMs outperforms the KN5 LMs significantly in terms of both PPLs and WERs. Compared with the RNN LMs, the TRF LMs give close PPLs but smaller WERs. The LSTM LMs alone performs better than TRF LMs, but interpolated TRF and LSTM LMs produces the lowest WER. Numerically, "LSTM+TRF" trained on 32M training corpus achieves 6.33% WER, which represents 17.4% relative reduction over KN5, 5.4% over LSTM and 4.0% over "LSTM+KN5".

9 Conclusion and Future Directions

The undirected graphical modeling approach is rarely used to describe trans-dimensional observations but has its own merits: being more natural for domains where fixing the directions of edges is awkward, being computationally more efficient in computing data probabilities by avoiding local normalization, and being able to flexibly integrate a richer set of features. In this paper, we investigate building RF models for trans-dimensional observations and make the following contributions. (1) We propose the TRF model and develop the AugSA training algorithm, which jointly estimates the model parameters and normalizing constants while using trans-dimensional mixture sampling. (2) Several statistical and computational innovations are introduced to improve the training convergence and reduce computational cost, which together enable us to successfully train TRF models on large datasets. (3) The new TRF model and AugSA training algorithm are thoroughly evaluated. The superiority of the TRF approach is demonstrated in language modeling experiments. The TRF models obtained significantly outperform the classic n-gram LMs and perform competitive with NN LMs but being much faster in calculating sentence probabilities.

There are various worthwhile directions for future research. First, the TRF approach will hopefully open a new door to language modeling in addition to the dominant conditional approach, as once envisioned in [5]. Apart from its success in language modeling, the TRF approach can also be applied to other sequential and trans-dimensional data modeling tasks in general. Second, although this work models trans-dimensional observations using the log-linear form and without hidden variables, it is an important direction of extending TRF models to introduce non-linear features (e.g. implemented by NNs) and hidden variables to further enhance their representational power. Third, another important line of extension arises by applying the TRF approach to discriminative modeling, even though we are mainly concerned with generative modeling of trans-dimensional data in this work. The conditional distributions over labels given the observations in CRFs can be described by TRFs so that higher-order interactions among labels can be utilized. Finally, to facilitate future studies along the TRF approach, we have released an open-source TRF toolkit[23], which contains the code and scripts to reproduce the results in this paper.

10 Acknowledgments

This work is partly supported by NSFC grant 61473168, Tsinghua Initiative 20121088069 and Toshiba Corporation.

References

[1] B. J. Frey and N. Jojic, "A comparison of algorithms for inference and learning in probabilistic graphical models," *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, vol. 27, no. 9, pp. 1392–1416, 2005.

[2] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

22. After converting all the words to upper case, the vocabulary size is reduced to about 18K.

23. https://github.com/wbengine/SPMILM/

[3] J. Lafferty, A. McCallum, and F. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” *Proc. International Conference on Machine Learning (ICML)*, pp. 282–289, 2001.

[4] S. D. Pietra, V. D. Pietra, and J. Lafferty, “Inducing features of random fields,” *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, vol. 19, pp. 380–393, 1997.

[5] R. Rosenfeld, S. F. Chen, and X. Zhu, “Whole-sentence exponential language models: a vehicle for linguistic-statistical integration,” *Computer Speech & Language*, vol. 15, pp. 55–73, 2001.

[6] S. F. Chen and J. Goodman, “An empirical study of smoothing techniques for language modeling,” *Computer Speech & Language*, vol. 13, pp. 359–394, 1999.

[7] H. Schwenk, “Continuous space language models,” *Computer Speech & Language*, vol. 21, pp. 492–518, 2007.

[8] T. Mikolov, S. Kombrink, L. Burget, J. H. Cernocky, and S. Khudanpur, “Extensions of recurrent neural network language model,” in *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011.

[9] M. Sundermeyer, R. Schlüter, and H. Ney, “Lstm neural networks for language modeling.” in *Proc. Interspeech*, 2012, pp. 194–197.

[10] S. Khudanpur and J. Wu, “Maximum entropy techniques for exploiting syntactic, semantic and collocational dependencies in language modeling,” *Computer Speech & Language*, vol. 14, pp. 355–372, 2000.

[11] B. Roark, M. Saraclar, M. Collins, and M. Johnson, “Discriminative language modeling with conditional random fields and the perceptron algorithm,” in *Proc. Ann. Meeting of the Association for Computational Linguistics (ACL)*, 2004, p. 47.

[12] S. F. Chen, “Shrinking exponential language models,” in *Proc. Human Language Technologies: Ann. Conference of the North American Chapter of the Association for Computational Linguistics*, 2009.

[13] F. Amaya and J. M. Benedí, “Improvement of a whole sentence maximum entropy language model using grammatical features,” in *Proc. Ann. Meeting of the Association for Computational Linguistics (ACL)*, 2001.

[14] T. Ruokolainen, T. Alumäe, and M. Dobrinkat, “Using dependency grammar features in whole sentence maximum entropy language model for speech recognition.” in *Baltic HLT*, 2010.

[15] J. N. Darroch and D. Ratcliff, “Generalized iterative scaling for log-linear models,” *The Annals of Mathematical Statistics*, pp. 1470–1480, 1972.

[16] H. Robbins and S. Monro, “A stochastic approximation method,” *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

[17] A. Benveniste, M. Métivier, and P. Priouret, *Adaptive algorithms and stochastic approximations*. New York: Springer, 1990.

[18] H. Chen, *Stochastic approximation and its applications*. Springer Science & Business Media, 2002.

[19] T. Mikolov, “Statistical language models based on neural networks,” *Ph.D. thesis, Brno University of Technology*, 2012.

[20] W. Zaremba, I. Sutskever, and O. Vinyals, “Recurrent neural network regularization,” *arXiv preprint arXiv:1409.2329*, 2014.

[21] B. Wang, Z. Ou, and Z. Tan, “Trans-dimensional random fields for language modeling,” in *Proc. Annu. Meeting of the Association for Computational Linguistics (ACL)*, 2015.

[22] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv:1412.6980 [cs.LG]*, 2014.

[23] R. M. Neal, “Annealed importance sampling,” *Statistics and Computing*, vol. 11, no. 2, pp. 125–139, 2001.

[24] L. Younes, “Parametric inference for imperfectly observed gibbsian fields,” *Probability Theory and Related Fields*, vol. 82, pp. 625–645, 1989.

[25] T. Tieleman, “Training restricted boltzmann machines using approximations to the likelihood gradient,” in *Proc. International Conference on Machine Learning*, 2008.

[26] R. Malouf, “A comparison of algorithms for maximum entropy parameter estimation,” in *Proc. Conference on Natural Language Learning (CoNLL)*, 2002.

[27] T. P. Minka, “A comparison of numerical optimizers for logistic regression,” *Technical report, MSR*, 2003.

[28] M. G. Gu and H.-T. Zhu, “Maximum likelihood estimation for spatial models by markov chain monte carlo stochastic approximation,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, pp. 339–355, 2001.

[29] Z. Tan, “Optimally adjusted mixture sampling and locally weighted histogram analysis,” *Journal of Computational and Graphical Statistics*, vol. 26, pp. 54–65, 2017.

[30] P. J. Green, “Reversible jump markov chain monte carlo computation and bayesian model determination,” *Biometrika*, vol. 82, pp. 711–732, 1995.

[31] F. Liang, C. Liu, and R. J. Carroll, “Stochastic approximation in monte carlo computation,” *Journal of the American Statistical Association*, vol. 102, no. 477, pp. 305–320, 2007.

[32] B. T. Polyak and A. B. Juditsky, “Acceleration of stochastic approximation by averaging,” *SIAM Journal on Control and Optimization*, vol. 30, pp. 838–855, 1992.

[33] S. Wiesler and H. Ney, “A convergence analysis of log-linear training,” in *Advances in Neural Information Processing Systems*, 2011.

[34] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural Networks for Machine Learning*, 2012.

[35] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, P. Koehn, and T. Robinson, “One billion word benchmark for measuring progress in statistical language modeling,” *arXiv preprint arXiv:1312.3005*, 2013.

[36] N. Shazeer, J. Pelemans, and C. Chelba, “Sparse non-negative matrix language modeling for skip-grams,” in *Proc. INTERSPEECH*, 2015.

[37] S. Martin, J. Liermann, and H. Ney, “Algorithms for bigram and trigram word clustering,” *Speech Communication*, vol. 24, pp. 19–37, 1998.

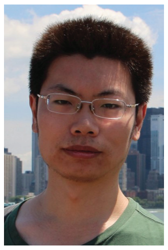
[38] J. Goodman, “A bit of progress in language modeling,” *Computer Speech & Language*, vol. 15, pp. 403–434, 2001.

[39] A. Mnih and K. Kavukcuoglu, “Learning word embeddings efficiently with noise-contrastive estimation,” in *Proc. Neural Information Processing Systems (NIPS)*, 2013.

[40] E. Grave, A. Joulin, M. Cissé, D. Grangier, and H. Jégou, “Efficient softmax approximation for gpus,” *arXiv preprint arXiv:1609.04309*, 2016.

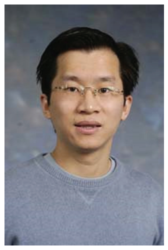
[41] J. Goodman, “Classes for fast maximum entropy training,” in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2001.

[42] G. Hinton, “A practical guide to training restricted boltzmann machines,” *Momentum*, vol. 9, p. 926, 2010.



Bin Wang received the B.S. degree in electronic engineering from Tsinghua University in 2012. Since 2012, he has been pursuing a Ph.D. degree at the Department of Electronic Engineering in Tsinghua University under the supervision of Zhijian Ou. From April 2015 to September 2015, he was a visiting student and worked with Zhiqiang Tan at the Department of Statistics in Rutgers University. His research focuses on trans-dimensional random fields and language modeling.

Zhijian Ou received the B.S. degree with the highest honor in electronic engineering from Shanghai Jiao Tong University in 1998 and the Ph.D. degree in electronic engineering from Tsinghua University in 2003. Since 2003, he has been with the Department of Electronic Engineering in Tsinghua University and is currently an associate professor. From August 2014 to July 2015, he was a visiting scholar at Beckman Institute, University of Illinois at Urbana-Champaign. He has actively led research projects from NSF China (NSFC), China 863 High-tech Research and Development Program, and China Ministry of Information Industry, as well as joint-research projects with Intel, Panasonic, IBM, and Toshiba. His recent research interests include speech processing (speech recognition and understanding, speaker recognition, natural language processing) and machine intelligence (particularly with graphical models).



Zhiqiang Tan is a Professor with the Department of Statistics at Rutgers University. He received the B.S. degree in applied mathematics from Tsinghua University in 1998 and the Ph.D. degree in statistics from University of Chicago in 2003. He was an Assistant Professor at Johns Hopkins University from 2003 to 2007 before joining Rutgers University. He received an NSF CAREER award in 2007, and became an Elected Member of International Statistical Institute in 2015. He has published extensively on statistical theory, methods, and applications in leading statistical and interdisciplinary journals and conferences. His research interests include Monte Carlo methods, causal inference, statistical learning, and related areas.

Appendix A
Proofs of Propositions 1 and 2

For model (2), the MLE of π_j is easily seen to be $\hat{\pi}_j = n_j/n$ for $j = 1, \ldots, m$. Given the dimension probabilities π equal to the empirical $\hat{\pi}$, the log-likelihood of λ scaled by n^{-1} under model (2) is

$$L(\lambda) = \frac{1}{n}\sum_{j=1}^{m}\sum_{x^j \in D_j} \log p(j, x^j; \hat{\pi}, \lambda),$$

By setting to 0 the derivative of $L(\lambda)$ with respect to λ, we obtain that MLE of λ is determined by (6), i.e.

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

This completes the proof of Proposition 1.

For model (3), the log-likelihood of (λ, ν) scaled by n^{-1} is

$$L(\lambda, \nu) = \frac{1}{n}\sum_{j=1}^{m}\sum_{x^j \in D_j} \log p(j, x^j; \lambda, \nu).$$

By setting to 0 the gradient of $L(\lambda, \nu)$, we find that MLEs of (λ, ν) are determined by the equations

$$\frac{\partial L(\lambda,\nu)}{\partial \lambda} = \tilde{p}[f] - p_{\lambda,\nu}[f] = 0, \quad (28)$$

$$\frac{\partial L(\lambda,\nu)}{\partial \nu} = \tilde{p}[\delta] - p_{\lambda,\nu}[\delta] = 0, \quad (29)$$

where $\tilde{p}[\delta]$ and $\tilde{p}[f]$ denote the expectations of δ and f with respect to the empirical distribution as before and $p_{\lambda,\nu}[\delta]$, $p_{\lambda,\nu}[f]$ denote the expectations of δ and f with respect to the distribution (3). By direct calculation, we have $\tilde{p}[\delta_j] = \hat{\pi}_j$ and

$$\begin{aligned} p_{\lambda,\nu}[\delta_j] &= \sum_{x^j \in \mathcal{X}^j} p(j, x^j; \lambda, \nu) \\ &= \frac{e^{\nu_j}}{Z(\lambda,\nu)} \sum_{x^j \in \mathcal{X}^j} e^{\lambda^T f(x^j)} \\ &= \frac{e^{\nu_j}}{Z(\lambda,\nu)} Z_j(\lambda). \end{aligned}$$

Then (29) is equivalent to

$$\hat{\pi}_j = \frac{e^{\nu_j} Z_j(\lambda)}{Z(\lambda,\nu)} = \frac{e^{\nu_j} Z_j(\lambda)}{\sum_{l=1}^{m} e^{\nu_l} Z_l(\lambda)}, \quad j = 1, \ldots, m, \quad (30)$$

where the second equality follows from (4). Given the relation (30), the fitted model (3) can be expressed as

$$p(j, x^j; \lambda, \nu) = \frac{\hat{\pi}_j}{Z_j(\lambda)} e^{\lambda^T f(x^j)}. \quad (31)$$

By direct calculation using (31), we have

$$\begin{aligned} p_{\lambda,\nu}[f] &= \sum_{j=1}^{m}\sum_{x^j \in \mathcal{X}^j} p(j, x^j; \lambda, \nu) f(x^j) \\ &= \sum_{j=1}^{m} \hat{\pi}_j p_{\lambda,j}[f] = p_\lambda[f]. \end{aligned}$$

Then (28) is equivalent to $\tilde{p}[f] - p_\lambda[f] = 0$, that is, (6). In summary, we have shown that (28) and (29) are jointly equivalent to (6) and (30).

(i) If $\hat{\lambda}$ is an MLE of λ in model (2), satisfying (6), then $\check{\lambda} = \hat{\lambda}$ and $\check{\nu}$ defined by (7) satisfy (6) and (30) jointly, and hence are MLEs of (λ, ν) in model (3).

(ii) If $(\check{\lambda}, \check{\nu})$ are MLEs of (λ, ν) in model (3), then $(\check{\lambda}, \check{\nu})$ satisfy (6) and (30) jointly, and $\hat{\lambda} = \check{\lambda}$ is an MLE of λ in model (2).

(iii) The equivalence between the fitted distributions $p(j, x^j; \check{\lambda}, \check{\nu})$ and $p(j, x; \hat{\pi}, \hat{\lambda})$ follows directly from (31).

This completes the proof of Proposition 2.

Appendix B
Simultaneous equations for SA algorithm and proof of Proposition 3

We first recast (6) and (9) jointly as a system of simultaneous equations

$$\tilde{p}[f] - p_{\lambda,\zeta}[f] = 0, \quad (32)$$

$$\hat{\pi}_l - p(l; \lambda, \zeta) = 0, \quad l = 1, \ldots, m, \quad (33)$$

where $p_{\lambda,\zeta}[f]$ is the expectation of f with respect to the joint distribution (8). In fact, (33) is the same as (9) and implies that $\zeta_l = \zeta_l^*(\lambda)$, $l = 1, \ldots, m$ for any fixed λ. Given this relation, the joint distribution (8) reduces to (2) and hence (32) reduces to (6). Eqs. (32)–(33) can then be put in the form of $h(\theta) = 0$, with $g(\cdot; \theta)$ and $H(Y; \theta)$ defined in (8) and (11).

Proposition 3 follows because, by standard calculation for importance sampling, we have $p_{\lambda,\zeta}[f] = q_{\lambda,\zeta}[\frac{\pi}{\pi^0} f]$ and $p(l; \lambda, \zeta) \propto \frac{\pi_l}{\pi_l^0} q(l; \lambda, \zeta)$ for $l = 1, \ldots, m$, where $q(l; \lambda, \zeta) = q_{\lambda,\zeta}[\delta_l]$ is the marginal probability of l under the joint distribution (12):

$$q(l; \lambda, \zeta) = \frac{\pi_l^0 e^{-\zeta_l + \zeta_l^*(\lambda)}}{\sum_{j=1}^{m} \pi_j^0 e^{-\zeta_j + \zeta_j^*(\lambda)}}.$$

Appendix C
Sampling acceleration

For MCMC sampling in Section 4.3, the Gibbs sampling operation of drawing $x_i^{(t)}$ (Step 16 in Tab.1) involves calculating the probabilities of all the possible elements in position i. This is computationally costly, because the size of alphabet set $\mathcal{A}$ (i.e. the number of possible values at each position) may be very large 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 value in $\mathcal{A}$ is assigned to a single class. If the total class number is $|\mathcal{C}|$, then there are, on average, $|\mathcal{A}|/|\mathcal{C}|$ values 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 value belonging to class $c_i^{(t)}$. The computational cost is reduced from $|\mathcal{A}|$ to $|\mathcal{C}| + |\mathcal{A}|/|\mathcal{C}|$ on average.

The idea of using class information to accelerate training has been proposed in various contexts of language modeling, such as maximum entropy models [41] and RNN LMs [8]. However, the realization of this idea is different for training our TRF models.

The pseudo-code of the new sampling method is shown in the Algorithm 2 of Tab.1. Denote by $\mathcal{A}_c$ the subset of $\mathcal{A}$ containing all the elements belonging to class c. In the Markov move step (Step 13 to 18 in Tab.1), at each position i, we first generate a class c_0 from a proposal distribution with probability $Q_i(c)$ for class c and then accept c_0 as the new $c_i^{(t)}$ with probability

$$\min\left\{1, \frac{Q_i(c_i^{(t)})}{Q_i(c_0)} \frac{p_i(c_0)}{p_i(c_i^{(t)})}\right\}, \quad (34)$$

where

$$p_i(c) = \sum_{w \in \mathcal{A}_c} q(j^{(t)}, \{x_{1:i-1}^{(t)}, w, x_{i+1:j^{(t)}}^{(t)}\}; \lambda, \zeta).$$

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

Similarly, in the local jump step with $k = j^{(t-1)}$, if the proposal $j = k + 1$ (Step 5 to 9 in Tab.1), we first generate $c_0 \sim Q_{k+1}(c)$ and then generate u from class c_0 by

$$\breve{g}_{k+1}(u|x^k, c_0) = \frac{q(k+1, \{x^k, u\}; \lambda, \zeta)}{\sum_{w \in \mathcal{A}_{c_0}} q(k+1, \{x^k, w\}; \lambda, \zeta)} \tag{35}$$

with $x^k = x^{(t-1)}$. Then we set $j^{(t)} = j$ and $x^{(t)} = \{x^{(t-1)}, u\}$ with probability as defined in (20), by setting

$$g_{k+1}(u|x^k) = Q_{k+1}(c_0)\breve{g}_{k+1}(u|x^k, c_0). \tag{36}$$

If the proposal $j = k - 1$, similarly we use acceptance probability (21) and

$$g_k(u|x^{k-1}) = Q_k(c)\breve{g}_k(u|x^{k-1}, c) \tag{37}$$

with $x^{k-1} = x_{1:j}^{(t-1)}$, $u = x_k^{(t-1)}$ and $c = c_k^{(t-1)}$.

In our applications, we construct $Q_i(c)$, the proposal probability of class c at position i, dynamically as follows. Write x^l for $\{x^{(t-1)}, u\}$ in Step 8 or for $x^{(t)}$ in Step 11 of Algorithm 2 in Tab.1. 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 $p_l(x^l; \lambda)$. 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.

Appendix D
Parallelization of sampling

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

Appendix E
Regularization and stop decision

In applications, L2 regularization is commonly used to reduce overfitting. Let μ a constant for L2 regularization. The update of λ in (16) should be revised to

$$\lambda^{(t)} = \lambda^{(t-1)} + \frac{\gamma_{\lambda,t}}{\sigma + \mu}\left\{-\mu\lambda^{(t-1)} + \tilde{p}[f] - \frac{1}{K}\sum_{(j,x^j) \in B^{(t)}} \frac{\pi_j}{\pi_j^0} f(x^j)\right\}. \tag{38}$$

An important question in training is when to stop the iteration. One choice is to use the stop criterion in RNN training [8], which monitors the log-likelihoods on the development set. If the change in the log-likelihoods on the development set is no more significant, we stop the iteration. But for augmented SA, the log-likelihoods depend on the estimated log normalizing constants ζ_j, which may fluctuate heavily. As a result, monitoring the log-likelihoods on the development set is unstable. An alternative method is inspired by monitor the overfitting in training restricted Boltzmann machines (RBM) as in [42]. The idea is to compare the difference between the log-likelihoods on the development and training sets to eliminate the influence of the unknown normalizing constants. For a dataset B, we define the log-likelihood of λ and ζ scaled by the dataset size as

$$L_B(\lambda, \zeta) = \frac{1}{|B|}\sum_{x^j \in B} \log p(j, x^j; \lambda, \zeta).$$

Then we calculate the gap between the scaled log-likelihoods on training set B_r and development set B_d at iteration t:

$$D_{B_r,B_d}(t) = L_{B_r}(\lambda^{(t)}, \zeta^{(t)}) - L_{B_d}(\lambda^{(t)}, \zeta^{(t)}),$$

Strictly speaking, because the log ratio of normalizing constants ζ_j depends on the dimension j, $D_{B_r,B_d}(t)$ is independent of the estimates $\zeta^{(t)}$ if and only if the dimension probabilities in B_r and B_d are identical. To reduce the influence of $\zeta^{(t)}$ in our experiments, we calculate the averaged values of $D_{B_r,B_d}(t)$ per H iterations and monitor the relative increase:

$$S_t = \frac{1}{H}\left\{\sum_{k=t-H+1}^{t} D_{B_r,B_d}(k) - \sum_{k=t-2H+1}^{t-H} D_{B_r,B_d}(k)\right\}.$$

If the relative increase S_t is less that a threshold, such as 10^{-4} with $H = 100$, then we stop the iterations. Note that overfitting may already happen at the time t when the gap $D_{B_r,B_d}(t)$ increases. Therefore, we need to introduce the regularization to avoid such overfitting.