MCMC algorithms applications in various NLP problems

by

Xiaochang Peng

Supervised by

Professor Daniel Gildea

Department of Computer Science
Arts, Sciences and Engineering
Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester
Rochester, New York

2016

Abstract

MCMC algorithms have been used This paper investigates the applications of a specific type of algorithms, MCMC algorithms, to different NLP problems.

Table of Contents

1 Introduction

MCMC algorithms is useful in reducing the search space.

The rest of the paper is structured as follows. In Chapter 2, I will discuss the MCMC algorithms in general. Chapter 3-6 will separately introduce the application of MCMC algorithms to semantic parsing, decipherment-based machine translation, distributed representation learning and neural machine translation. In Chapter 7, I conclude the proposal paper. For a lot of probabilistic models for NLP applications, doing exact inference is often intractable. Therefore, approximation methods have to be used to overcome the complexity issue. Sampling methods, or known as Monte Carlo methods, are widely used to do approximate inference. The basic idea is to use a certain number of samples from the desired distribution to approximate the exact inference.

Some widely used sampling methods, such as rejection sampling and importance sampling, suffer from severe limitations of high dimensional spaces. Markov Chain Monte Carlo (MCMC) algorithms, on the other hand, allows for sampling from a large class of distributions and scales well to high dimensional spaces. MCMC methods have their origins in physics (Metropolis and Ulam, 1949) and later start to have their uses in other fields of statistics.

MCMC algorithms have also been widely used in NLP such as word segmentation, grammar induction et al. One widely used

2 MCMC algorithms

In this chapter, I mainly go over MCMC algorithms and review some properties of MCMC.

2.1 Introduction

2.2 MCMC

2.2.1 Convergence property

A first order Markov chain is defined as a series of states (assignments) of random variables $z^{(1)}, \cdots, z^{(M)}$ such that the following conditional independence property holds for $m \in 1, \cdots, M-1$:

$$p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(1)}, \cdots, \mathbf{z}^{(m)}) = p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)})$$

We define *transition probabilities* $T(\mathbf{z}^{(m)}, \mathbf{z}^{(m+1)}) \equiv p(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)})$. A $homogeneous$ Markov chain is one that the transition probabilities are the same for all m.

The marginal probability for a state at the $m+1$-th position can be expressed in

terms of the marginal probability for the previous state in the chain:

$$p(\mathbf{z}^{(m+1)}) = \sum_{p(\mathbf{z}^{(m)})} T(\mathbf{z}^{(m+1)}|\mathbf{z}^{(m)})p(\mathbf{z}^{(m)})$$

A distribution is **stationary** with respect to a Markov chain if each step in the chain leaves the distribution invariant. For a homogeneous Markov chain with transition probabilities $T(z', z)$, the distribution $p(z)$ is invariant if

$$p(\mathbf{z}) = \sum_{\mathbf{z}'} T(\mathbf{z}', \mathbf{z})p(\mathbf{z}')$$

One sufficient (but not necessary) condition for ensuring the required distribution $p(z)$ to be stationary is to choose the transition probabilities to satisfy the property of *detailed balance*, defined by

$$T(\mathbf{z}, \mathbf{z}')p(\mathbf{z}) = T(\mathbf{z}', \mathbf{z})p(\mathbf{z}')$$

With detailed balance satisfied, then the distribution is stationary:

$$\sum_{\mathbf{z}'} T(\mathbf{z}', \mathbf{z})p(\mathbf{z}') = \sum_{\mathbf{z}'} T(\mathbf{z}, \mathbf{z}')p(\mathbf{z}) = p(\mathbf{z})$$

Our goal is to use Markov chains to sample from a given distribution. We can achieve this if we set up a Markov chain such that the desired distribution is invariant. However, we must also require that for $m \to \infty$, the distribution $p(\mathbf{z}^{(m)})$ converges to the required invariant distribution $p(\mathbf{z})$, irrespective of the choice of initial distribution $p(\mathbf{z}^{(0)})$. This property is called *ergodicity*, and the stationary distribution is then called the *equilibrium* distribution.

A Markov chain is *irreducible* if for $\forall a, b \in \Pi$, Π is the value space for $\mathbf{z}$, $\exists m \geq 0$, s.t. $p(\mathbf{z}^{(m)} = b|\mathbf{z}^{(0)} = a) > 0$. That is, any state in the value space can be reached within finite numbers of steps from any starting state. An irreducible Markov chain is *aperiodic* if $\forall a \in \Phi$, *gcd*$\{m : p(\mathbf{z}^{(m)} = a|\mathbf{z}^{(0)} = a) > 0\}$ = 1. Where *gcd* is the greatest common divisor of a series of numbers. Based on these definitions, we can define the ergodic theorem.

Theorem 1 *If* $(\mathbf{z}^{(0)}, \mathbf{z}^{(1)}, \cdots, \mathbf{z}^{(n)})$ *is an irreducible (homogeneous) discrete Markov Chain with a stationary distribution* $p^*(\mathbf{z})$*, then:*

$$\frac{1}{n}\sum_{i=1}^{n} f(\mathbf{z}^{(i)}) \xrightarrow[n\to\infty]{} Ef(\mathbf{z})$$

where $\mathbf{z}$ $p^*(\mathbf{z})$*, for any bounded function* $f : z \to \mathbb{R}$ *If further, it is aperiodic, then* $p(\mathbf{z}^{(n)}|\mathbf{z}^{(0)}) \xrightarrow[n\to\infty]{} p^*(\mathbf{z})$

Ergodic theorem can be used to define Markov chains that regardless of which state we start from, the distribution of the states will converge to the equilibrium distribution after a number of steps.

2.3 Types of MCMC algorithms

2.3.1 Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm was first introduced by Hastings (1970), which applies to cases where the proposal distribution is not symmetric. Assume the current hidden state at time t to be $\mathbf{z}^{(\mathbf{t})}$, we then draw a sample $\mathbf{z}^*$ from the distribution $q(\mathbf{z}|\mathbf{z}^{(\mathbf{t})})$ and accept it with the following probability:

$$A(\mathbf{z}^*, \mathbf{z}^{(\mathbf{t})}) = \min\left(1, \frac{\tilde{p}(\mathbf{z}^*)q(\mathbf{z}^{(\mathbf{t})}|\mathbf{z}^*)}{\tilde{p}(\mathbf{z}^{(\mathbf{t})})q(\mathbf{z}^*|\mathbf{z}^{(\mathbf{t})})}\right)$$

Evaluating the acceptance rate does not require knowledge about the normalization constant Z_p in the distribution $p(\mathbf{z}) = \frac{\tilde{p}(\mathbf{z})}{Z_p}$. $p(\mathbf{z})$ is a stationary distribution with respect to the Markov chain. We can verify it with proving detailed balance:

$$\begin{aligned} p(\mathbf{z})q(\mathbf{z}|\mathbf{z}')A(\mathbf{z}', \mathbf{z}) &= \min(p(\mathbf{z})q(\mathbf{z}|\mathbf{z}'), p(\mathbf{z}')q(\mathbf{z}'|\mathbf{z})) \\ &= \min(p(\mathbf{z}')q(\mathbf{z}'|\mathbf{z}), p(\mathbf{z})q(\mathbf{z}|\mathbf{z}')) \\ &= p(\mathbf{z}')q(\mathbf{z}'|\mathbf{z})A(\mathbf{z}, \mathbf{z}') \end{aligned} \tag{2.1}$$

Algorithm 1 Gibbs Sampling

1: Initialize $\{z_i : i = 1, \cdots, M\}$
2: **for** $\tau = 1, \cdots, T$: **do**
3: -Sample $z_1^{(\tau+1)} \sim p(z_1 | z_2^{(\tau)}, z_3^{(\tau)}, \cdots, z_M^{(\tau)})$
4: -Sample $z_2^{(\tau+1)} \sim p(z_2 | z_1^{(\tau+1)}, z_3^{(\tau)}, \cdots, z_M^{(\tau)})$
5: $\vdots$
6: -Sample $z_j^{(\tau+1)} \sim p(z_j | z_1^{(\tau+1)}, \cdots z_{j-1}^{(\tau+1)}, z_{j+1}^{(\tau)} \cdots, z_M^{(\tau)})$
7: $\vdots$
8: -Sample $z_M^{(\tau+1)} \sim p(z_M | z_1^{(\tau+1)}, z_2^{(\tau+1)}, \cdots, z_{M-1}^{(\tau+1)})$

For Metropolis-Hastings algorithms the specific choice of proposal distribution has a great effect on the performance. For continuous variable spaces, a common choice is usually a Gaussian centered on the current state. And the choice of variance parameter would balance between acceptance rate and the convergence speed.

2.3.2 Gibbs Sampling

Gibbs sampling (Geman and Geman, 1984) is another widely used MCMC algorithm, which can be seen as a special case of the Metropolis-Hastings algorithm.

Consider we wish to sample from the distribution $p(\mathbf{z}) = p(z_1, \cdots, z_M)$ and we start from an initial state. Each step of the Gibbs sampling algorithm involves replacing the value of one of the variables by a value drawn from the distribution of that variable conditioned on the values of the remaining variables. In the i-th step we replace z_i by a value drawn from the distribution $p(z_i | \mathbf{z}_{\backslash \mathbf{i}})$, where z_i denotes the i-th component of $\mathbf{z}$, and $\mathbf{z}_{\backslash \mathbf{i}}$ denotes $z_1, \cdots, z_M$ but with z_i omitted. This procedure is repeated either by cycling through the variables in some particular order or by choosing the variable to be updated at each step at random from some distribution. In traditional NLP problems, the hidden states $\mathbf{z}$ are usually structured as a sequence, a tree or a directed graph which usually provides explict order for traversal and therefore for Gibbs update.

2.4 Conclusion

In this chapter, I have gone through some properties of MCMC algorithms that enable the Markov chain to converge to the desired distribution after finite number of steps. Two widely used MCMC algorithms, Metropolis-Hastings and Gibbs Sampling algorithms, provide simple but power tools to sample from intractable probability distributions.

3 AMR parsing

In this chapter, we apply MCMC algorithm to learn synchronous graph grammars for mapping strings to graphs structure. Specifically, we learn Synchronous Hyperedge Replacement Grammar (SHRG) rules from a forest that represents likely derivations consistent with a fixed string-to-graph alignment. We make an analogy of string-to-AMR parsing to the task of phrase-based machine translation and come up with an efficient algorithm to learn graph grammars from string-graph pairs. We propose an effective approximation strategy to resolve the complexity issue of graph compositions. We also show some useful strategies to overcome existing problems in an SHRG-based parser and present preliminary results of a graph-grammar-based approach.

3.1 Forced decoding

The generative decoding model has a few limitations: 1. the features are very simple and no global feature is included. 2. All the weights of the features are set by hand, which suffers from the limit of huge space of combination of the weights. One way to deal with these limitations is to use a feature-rich discriminative model. The training data is used to extract the SHRG, more specifically rules and the count for each rule. However, the structural information of the derivation forest is not used. Using a dis-

criminative model which learns to search within the forest structure would make use of more structural information in our training data.

Yu et al. (2013) have proposed a structured perceptron algorithm that learns to search for derivations that generate the whole or prefixes of the target reference sentences for phrase-based machine translation. Beam search is used to reduce the intractable search space and each bin B_i maintain a K-best list of hypotheses that covers i words on the source side. A search error happens for a certain hypothesis d when its target side $e(d)$ is not a prefix of the reference y.

3.1.1 Forced decoding for AMR parsing

In our AMR parsing scenario, we have a derivation forest representing possible ways to compose sentence-AMR pairs. Let $< x, y >$ be a sentence-AMR pair in the training data. We can compare the AMR graph side to the target side sentence and each hypothesis d contains a span-subgraph pair derived from a series of rule applications:

$$d = r_1 \circ r_2 \circ \cdots \circ r_{|d|}$$

where each r_i is a SHRG rule and $d = (e(d), g(d))$ is a (partial) derivation whose source side $e(d) = (x_i, \cdots, x_j)$ is a span which covers positions $[i, j]$ and whose target side is a subgraph $g(d)$ generated so far.

We maintain the a bin B_{ij} for each span $[i, j]$. Let $sub(y)$ be all the subgraphs of target side AMR y and $good_{ij}(x, y)$ be set of partial y-good derivations whose graph side is a subgraph of the reference AMR graph y and the sentence side covers span $[i, j]$:

$$good_{ij}(x, y) \triangleq \{d \in D(x) | g(d) \in sub(y), e(d) = (x_i, \cdots, x_j)\}$$

Conversely, we define y-bad partial derivations $bad_{ij}(x, y)$ as follows:

$$bad_{ij}(x, y) \triangleq \{d \in D(x) | g(d) \notin sub(y), e(d) = (x_i, \cdots, x_j)\}$$

The search using Earley algorithm proceeds as follows: use the *scan* and *complete* operation to find items that covers source span $[i, j]$. The cost for each item is scored as:

$$\mathbf{w} \cdot \mathbf{\Phi}(x, d, i, j)$$

where d is the partial derivation constructed so far. Φ is the features extracted within span $[i, j]$ and the partial derivation d. We keep two separate grammars to derive $good_{ij}(x, y)$ and $bad_{ij}(x, y)$. The $good_{ij}(x, y)$ is constructed by searching along the constructed forest, that is, follow the supervision of the composition of the sentence-AMR pair and therefore always derives a partial y-good derivations. $bad_{ij}(x, y)$ is constructed by searching through the whole grammar, which tries to find the derivation that has the greatest global score. Assume we have K items at each bin, we gradually build up each bin using recursion:

$$B_0 = \{\emptyset\}$$

$$B_{ij} = \mathbf{top}^K(good_{ij}(x, y)) \cap \mathbf{top}^K(bad_{ij}(x, y))$$

where $\mathbf{top}^K$ is an operator which computes the top K items of a stack according to the scoring function.

3.1.2 Parameter update

The perceptron works by adjusting the weights whenever search error happens and different variations of the perceptron algorithm make update at different points. The standard perceptron algorithm works by decoding the whole sentence first and update if the predication for the whole sentence does not match the reference. This update has the problem that in cases where search error happens at a very early step and there is no y-good item left in the bin. There is little point in continuing the search and update the parameters afterwards.

Early update is a special case of violation-fixing perceptron which stops decoding whenever the gold derivation falls off the beam. It makes update on the partial derivation so far and move on to the next training example when the update finishes. As we don't actually have the gold derivation, we replace it with the item in $good_{ij}(x, y)$ that has the largest score.

$$d_{ij}^{+} \equiv \text{argmax}_{d \in good_{ij}(x,y)} \mathbf{w} \cdot \Phi(x, d, i, j)$$

$$d_{ij}^{-} \equiv \text{argmax}_{d \in bad_{ij}(x,y)} \mathbf{w} \cdot \Phi(x, d, i, j)$$

$$\mathbf{w} \leftarrow \mathbf{w} + \triangle\Phi(x, d_{ij}^{+}, d_{ij}^{-}, i, j)$$

where $\triangle\Phi(x, d, d', i, j) \equiv \Phi(x, d, i, j) - \Phi(x, d', i, j)$ is the notation for the difference of feature vectors.

In practice, there are exponentially many y-good derivations for each sentence-AMR pair and our goal is to make sure the y-good derivation would succeed in the end. It is possible that in a certain span $[i, j]$ all y-good derivations fall off the bin, but search can still succeed from other spans. Therefore, we will continue the search through all possible spans until we reach the condition that there is no hope of finding another y-good derivation.

Another way of doing parameter update is to use *max-violation* proposed by Huang et al. (2012). The basic idea is to update at the point where the mistake is the biggest. More specifically, the update step is:

$$(ij)^{*} \equiv \text{argmin}_{ij} \mathbf{w} \cdot \triangle\Phi(x, d_{ij}^{+}, d_{ij}^{-}, i, j)$$

$$\mathbf{w} \leftarrow \mathbf{w} + \triangle\Phi(x, d_{(ij)^{*}}^{+}, d_{(ij)^{*}}^{-}, i, j)$$

3.2 Conclusion

We have presented an MCMC algorithm for sampling SCFG rules from derivation forests constructed from aligned sentence-AMR pairs. With the extracted grammar, we used Earley algorithm with cubing pruning to decode each sentence. To

overcome the limitations of simple local features and the hand-tuned weights for each feature, we propose a feature-rich discriminative model using forced decoding to include rich global features and adjust the weights using perceptron update.

4 Decipherment-based machine translation

This chapter introduces MCMC applications to decipherment-based machine translation.

4.1 Introduction

4.2 Decipherment-based machine translation

4.3 Feature-based

Optimization function, joint probability:

$$p(f_1, f_2) = \sum_{e_1 e_2} p(f_1 f_2, e_1 e_2)$$

The gradient of the joint log probability is:

$$\frac{\partial L}{\partial w} = E_{e_1 e_2 | f_1 f_2}[\Phi(f_1 f_2, e_1 e_2)] - E_{f_1 f_2, e_1 e_2}[\Phi(f_1 f_2, e_1 e_2)]$$

4.4 Reordering model

One problem with the previous model is that it does model reordering, which is a common phenomenon for machine translation. To model reordering, we additionally add the reordering term.

$$\frac{\partial L}{\partial w} = E_{e_1 e_2, I | f_1 f_2}[\Phi(f_1 f_2, e_1 e_2, I)] - E_{f_1 f_2, e_1 e_2, I}[\Phi(f_1 f_2, e_1 e_2, I)]$$

4.5 Modeling sentence

4.6 Conclusion

5 Distributed representation learning

In this paper, we introduce a variation of the skip-gram model which jointly learns distributed word vector representations and their way of composing to form phrase embeddings. In particular, we propose a learning procedure that incorporates a phrase-compositionality function which can capture how we want to compose phrases vectors from their component word vectors. Our experiments show improvement in word similarity and analogy tasks and a dependency parsing task using the proposed joint models.

5.1 Introduction

Distributed word vector representations learned from large corpora of unlabeled data have been shown to be effective in a variety of NLP tasks, such as POS tagging (**?**), parsing (**??**), language modeling (**??**) and machine translation (**????**). These vector representations are computed using co-occurrence statistics of words and their neighboring context words.

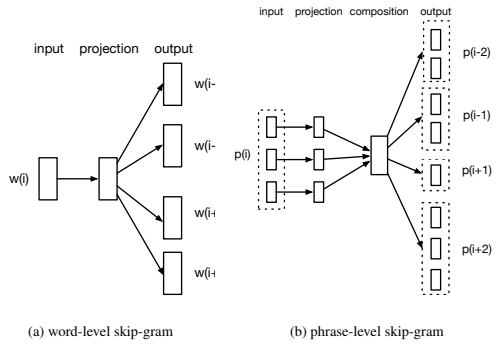
One of the most widely used approaches to learn these vector representations is the skip-gram model described in **?** and **?**. The skip-gram model optimizes the probability of predicting words in the context given the current center word. Figure 5.1 shows a standard skip-gram structure. **?** present a variation of skip-gram which captures the

relative position of context words by using a different weight matrix to connect the hidden layer to the context word at each relative position.

While word embeddings are usually used to compose distributed representations for larger units, this compositionality is not used during the whole learning procedure. **?** use a recursive neural network to learn weight matrices that capture compositionality. However, these matrices are updated using backpropagation over syntatic nodes of a parse tree and the learning procedure is limited to the substantially smaller number of labeled parse trees from the Penn Treebank. The phrasal information from the large unlabeled corpus is not utilized.

? propose a learning schedule which learns distributed vector representations for words and phrases jointly. However, in the optimization phase, they only maximize the correlation between the phrase representation and the component word representation, which does not fully capture the syntatic information of context. Recently **?** propose a feature-rich compositional transformation (FCT) model which learns weighted combination of word vectors to compose phrase vectors. During the learning procedure, the current phrase embedding is used to predict the context word vectors in the output embedding space and they mainly focus on bigram *NP*s.

In this paper, we extend the skip-gram model to utilize the phrase structure in a large corpus to capture phrase compositionality and the positional information of both words and phrases. We jointly model words in the context of words and phrases in the context of phrases. Additionally, we enforce a compositionality constraint on both the input and output phrase embedding spaces which indicates how we build the distributed vector representations for phrases from their component word vectors. Our results show that using phrase level context information can provide gains in both word similarity and analogy tasks and also syntactic tasks like dependency parsing.



(a) word-level skip-gram (b) phrase-level skip-gram

Figure 5.1: Architecture of the skip-gram model.

5.2 Skip-gram model

The skip-gram model learns distributed vector representations for words by maximizing the probability of predicting the context words given the current word. According to the *word2vec* implementation of **?**, each input word w is associated with a d-dimensional vector $v_w \in \mathbb{R}^d$ called the *input embedding* and each context word w_O is associated with a d-dimensional vector $v'_{w_O} \in \mathbb{R}^d$ called the *output embedding*. w, w_O are words from a vocabulary V of size W. The probability of observing w_O in the context of w is modeled with a softmax function:

$$P(w_O|w) = \frac{\exp({v'_{w_O}}^T v_w)}{\sum_{i=1}^{W} \exp({v'_{w_i}}^T v_w)} \tag{5.1}$$

The denominator of this function involves a summation over the whole vocabulary, which is impractical. One alternative to deal with the complexity issue is to sample several negative samples to avoid computing all the vocabulary. The objective function after using negative sampling is:

$$E_w = \sum_{w \in s} (\log \sigma({v'_{w_O}}^T v_w) + \sum_{i=1}^{K} \log \sigma(-{v'_{w_i}}^T v_w)) \tag{5.2}$$

where s is a chunked sentence. w_i, $i = 1, 2, \ldots, K$, are negative samples sampled from the following distribution:

$$P(w) = \frac{\widetilde{P}(w)^{\frac{3}{4}}}{Z} \tag{5.3}$$

where $\widetilde{P}(w)$ is the unigram distribution of words and Z is the normalization constant. The exponent $\frac{3}{4}$ is set empirically.

5.3 Compositionality-aware skip-gram model

To capture the way of composing phrase embeddings from distributed word vector representations, we extend the skip-gram model to include information from context of phrases and learn their compositionality from word vectors during the optimization procedure. Our phrase-level skip-gram structure is shown in Figure 5.1b.

5.3.1 Phrase-level skip-gram model

The word-level skip-gram model predicts the context words given the current word vector. Our approach further models the prediction of context phrases given the vector representation of the current phrase vector (Figure 5.1b). Assume $v_p \in \mathbb{R}^d$ to be the d-dimensional input embedding for current phrase p and $v'_{p_O} \in \mathbb{R}^d$ to be the output embedding for context phrase p_O. Using negative sampling, we model the phrase-level probability with:

$$E_p = \sum_{p \in s} (\log \sigma({v'_{p_O}}^T v_p) + \sum_{i=1}^{N} \log \sigma(-{v'_{p_i}}^T v_p)) \tag{5.4}$$

where p_i, $i = 1, 2, \ldots, N$, are negative samples sampled according to the unigram probability of phrases raised to the same exponent $\frac{3}{4}$.

In this paper, we jointly model word-level skip-gram and phrase-level skip-gram for each sentence:

$$E = E_w + \beta E_p \tag{5.5}$$

where $\beta > 0$ adjusts the relative importance of the word-level and the phrase-level skipgram.

5.3.2 Compositionality model

Assume a phrase p is composed of words $w_1, \ldots, w_{n_p}$, where n_p is the number of component words. The vector representation for p is computed as:

$$v_p = \Phi(\oplus(\sigma(v_{w_1}), \ldots, \sigma(v_{w_i}))) \tag{5.6}$$

where v_p is the vector representation for p. The function σ is a component-wise manipulation over each dimension. The symbol $\oplus$ is an operator over the component word vectors, which can be *linear combination, summation, concatenation* etc. The mapping function Φ is a linear or non-linear manipulation over the resulting vector after the $\oplus$ operation. The same composition function is used to compute the output phrase embeddings v'_{p_O} and v'_{p_i}, except that the component word vectors are v'_{w_i} instead of v_{w_i}.

To show the effect of modeling phrase embeddings, we experiment with a composition function where $\oplus$ is *linear combination* and Φ is passing the resulting matrix to the left of a weight vector $l^p = [l^p_1, l^p_2, \ldots, l^p_{n_p}]$ associated with each phrase p showing how we combine the component word vectors.

$$v_p = \left[\sigma\left(v_{w_1}\right), \ldots, \sigma\left(v_{w_{np}}\right)\right] \begin{bmatrix} l^p_1 \\ \vdots \\ l^p_{n_p} \end{bmatrix} \tag{5.7}$$

where the function σ is a component-wise power function over vector $v = [v_1, \ldots, v_n]$:

$$\sigma(v) = [\phi(v_1), \ldots, \phi(v_n)] \tag{5.8}$$

where $\phi(v_i) = \textit{sign}(v_i)|v_i|^\alpha$, $\alpha \geq 1$, is a power function over each dimension. This manipulation can be interpreted as adjusting dimensional values of word vectors to the phrase vector space.

Stochastic gradient ascent is used to update the word vectors. In equation 5.4, for each word w_j in p', either context phrase p_O or negative phrase sample p_i, the gradient

length	1	2	3	4	≥ 5
frequency	525m	137m	62m	22m	15m

Table 5.1: Statistics of phrase length distributions (m/million)

is:

$$\frac{\partial E_p}{\partial v'_{w_j}} = \nabla\sigma(v'_{w_j})(l_j^{p'}(y - \sigma({v'_{p'}}^T v_p))v_p) \tag{5.9}$$

where $y = 1$ for each word in p_O and 0 for each word in p_i. $\nabla\sigma(v'_{w_j})$ is a diagonal matrix where the i-th diagonal value is $\phi'(v'_{w_j i})$. For each word w_j in the current phrase p, the gradient is:

$$\frac{\partial E_p}{\partial v_{w_j}} = \nabla\sigma(v_{w_j})(l_j^p((1 - \sigma({v'_{p_O}}^T v_p))v'_{p_O} + \sum_{i=1}^{N}(-\sigma({v'_{p_i}}^T v_p))v'_{p_i})) \tag{5.10}$$

Similar compositionality functions trace back to **?**, where $\oplus$ is *concatenation* of component child node vectors (instead of word vectors) and Φ is a linear mapping from the concatenated vector to a d-dimensional vector:

$$v_p = [W_1^p, \ldots, W_{n_p}^p] \begin{bmatrix} \sigma(v_1) \\ \vdots \\ \sigma(v_{n_p}) \end{bmatrix} \tag{5.11}$$

where σ is component-wise identity. $W_i^p \in \mathbb{R}^{m \times d}$ is a matrix associated with the vector representation of the i-th component word of phrase p. Equation 5.7 can be seen as a special case of this composition when $m = d$, $W_i^p = l_i^p I_{d \times d}$ and the weight matrices are not updated during the procedure.

5.3.3 Output phrase embedding space

Following **?** in using different output embeddings at each relative position to capture order information of context words (we call this *positional* model), we use separate output embeddings to capture phrase-compositionality (we call this *compositional* model).

Model	denver_nuggets	fairy_tale	machine_learning
word2vec	dallas_mavericks, colorado_rockies, colorado_avalanche, dallas_desperados, colorado_rapids	cautionary_tales, folk_tales, weird_tales, strange_tales, some_tales	virtual_machines, turing_machines, time_machines, sewing_machines, intelligent_machines
compositional	seattle_supersonics, dallas_mavericks, philadelphia_76ers, colorado_rockies, minnesota_timberwolves	folk_tales, ghost_story, love_story, arthurian_legend, bedtime_story	computer_skills, cognitive_skills, computer_vision, learned_behavior, mathematics_and_computer_science

Table 5.2: Examples of nearest neighbors using word2vec and using additional compositional model

That is, we have a separate component word vector v'' to compose the phrase vectors in the context and the negative samples instead of using v'. The intuition of this choice is that we don't want the compositionality information in the context or negative sample layer to be distorted by word-level updates.

We further extend the phrase-level skipgram to include the order information, which uses different output word embeddings to compose phrases at each relative position. Phrases at the same relative position share the same output embeddings (we call this model *positional+compositional*). Without loss of generality, we experiment with the composition described in Equation 5.7. The coefficients l_i^ps are set to be $\frac{1}{n_p}$.

5.4 Experiments

We use the *senna* toolkit (**?**) to chunk our data, which identifies the constituent chunks in each sentence. Even though the chunking model is not trained in the same domain, the chunking result does not degrade much for the simplicity of the task itself. We run the chunker on 20 CPUs, and it takes less than 2 hours to chunk the Wikipedia 2010 corpus we use. The extracted phrases are labeled with *NP, VP, PP*s etc. The statistics of the lengths of the extracted phrases is shown in Table 5.1.

	word353	men	SYN	MIXED
word2vec	0.722	0.758	69.9	77.8
positional	0.692	0.744	71.2	79.7
compositional	**0.735**	**0.763**	70.6	79.2
compositional+positional	0.704	0.746	**72.0**	**80.5**

Table 5.3: Comparison of word-level tasks, including word similarity, analogy.

We train the skip-gram model with negative sampling using *word2vec* as our baseline. We used an April 2010 snapshot of the Wikipedia corpus (**?**), which contains approximately 2 million articles and 990 million tokens. We remove all words that have a frequency less than 20 and use a context window size of 5 (5 words before and after the word occurrence). We set the number of negative samples to be 10 and the dimensionality of vectors to be 300. For phrase-level skip-gram, we also use a context window size of 5 (5 phrases before and after the phrase occurrence).

5.4.1 Phrase nearest neighbors

Table 5.2 shows some phrases and their similar phrases (nearest neighbors). Following the ngram-to-ngram settings of **?**, we consider nearest neighbors that differ by more than one word. The word2vec phrase vectors are composed using summation. For word2vec, we can see that phrase vectors can be greatly biased by one component word. As *colorado* is similar to *denver* (so does *tales* to *tale* and *machines* to *machine*), the nearest neighbors tend to be phrases that have this similar word, even though the phrase meaning is very different. Using the compositional model, it is more likely to find phrases that have similar phrasal meanings. For example, the nearest neighbors of *denver nuggets* are mostly NBA teams. The model also find phrases of story types for *fairy tale* and related phrases like *compute vision* and *math and computer science* for *machine learning*.

	Dev		Test	
	UAS	LAS	UAS	LAS
word2vec	92.21	90.83	91.91	90.54
compositional	92.34	90.91	92.02	90.64
positional	92.29	90.88	92.05	90.67
compositional+ positional	**92.39**	**90.91**	**92.19**	**90.82**

Table 5.4: dependency parsing results on PTB using different embeddings

5.4.2 Word similarity and word analogy

We first consider word similarity and analogy tasks for evaluating the quality of word embeddings. Word similarity measures Spearman's correlation coefficient between the human scores and the embeddings' cosine similarities for word pairs. Word analogy measures the accuracy on syntactic and semantic analogy questions.

We evaluate similarity on two tasks, WordSim-353 and men, respectively containing 353 and 3000 word pairs. We use two word analogy datasets that we call SYN (8000 syntactic analogy questions) and MIXED (19544 syntactic and semantic analogy questions).

On the WordSim-353 task, **?** reported a Spearmans correlation of 0.709, while their word2vec baseline is 0.704. From Table 5.3, we can see that positional information actually degrades the performance on the similarity task, while only adding compositional information performs the best. For syntactic and mixed analogy tasks which involves syntax information, we can see that using phrase-level skip-gram and positional information both help and combining both gives the best performance.

5.4.3 Dependency parsing

The evaluation on dependency parsing is performed on the English PTB, with the standard train, dev and test splits with Stanford Dependencies. We use a neural network as

described in **?**. As we are using embeddings with a different dimensionality, we tune the hidden layer size and learning rate parameters for the neural network parser by grid search for each model and train for 15000 iterations. The other parameters are using the default settings. Evaluation is performed with the labeled (LAS) and unlabeled (UAS) attachment scores. We run each parameter setting for 3 times and then average to prevent randomness.

Table 5.4 shows the performance. We can see that combining phrase-level skip-gram and positional information consistently outperforms the baseline results.

5.5 MCMC sampling

5.5.1 Joint model

The extracted phrases using *senna* suffer from the small average length for each phrase. We can see from Table 5.1 that most of the phrases are of length 1 and compositionality is used in very limited places. Another problem with such kind of phrases is that the phrase boundaries are fixed and there is no way learning compositionality between two adjacent phrases, especially cases where different chunkings are also valid phrases. For example, *new zealand* and *new zealand islanders* are both valid phrases, while *senna* would chunk *new zealand* as a phrase and *islanders* as the second phrase.

MCMC algorithms have been used to extract phrases by iteratively going through the unlabeled data and sample from left to right, one variable at a time at each sentence. Usually a nonparametric bayesian model based on a nonparametric prior, such as Dirichlet Process, is used to smooth the distribution of the data and avoid overfitting. The resulting posterior distribution is usually of the same family of distribution as the prior and thus has a closed-form solution. During the sampling procedure, the boundary variables would be sampled according to the posterior distribution and the phrases in a sentence would vary whenever some of the boundary variables are flipped during

the sampling procedure.

Therefore, we propose a joint MCMC sampling and compositionality learning model for phrase extraction and way of composing phrase embeddings. We use a log bi-linear model to compute the probability of contexts given the phrase sequence:

$$P(\mathbf{b}, \mathbf{c}|\mathbf{w}) = \frac{1}{Z} e^{\sum_i v'_{c_i} \cdot v_{p_i}}$$

where $\mathbf{w}$ is the word sequence and $\mathbf{b}$ is the binary sequence which models the boundary information for each phrase in the sequence and $\mathbf{c}$ is the context sequence for the phrases in the sequence. At each position j, we sample its binary value:

$$b_j \propto e^{\sum_i v'_{c_i} \cdot v_{p_i}}$$

In practice we don't have to compute all the terms in the summation as most of them are not affected by the choice b_j. The terms that needs to be recomputed involves the phrases that are affected by b_j: (p_1, p_2) when $b_j = 1$ and p_3 when $b_j = 0$.

After we have sampled the boundary variable b_j and whenever we have reached a new phrase, we use the new phrase to predict its context. This can be done using the compositional skip-gram model we have introduced before and the vectors are updated during the procedure. As for the update, we consider a few choices for the time of vector update: 1. whenever a binary variable is flipped. 2. whenever we have reached the end of a phrase (sampled a binary value 1). 3. update when the boundary for the whole sentence is decided.

5.5.2 Phrase independence variation

In the previous section, we have modeled the probability of the whole sequence as a log bi-linear model. Here we consider factorizing the probability into product of probility of individual phrases.

$$P(\mathbf{b}, \mathbf{c}|\mathbf{w}) = \prod_i P(c_i|p_i)$$

Algorithm 2 A joint model for phrase extraction and compositionality learning

```
Initialize binary variables b_1, ··· , b_n
for iter = 1, ··· , T: do
    for s = 1, ··· , S: do
        for j = 1, ··· , n: do
            Let (p_1, p_2) be the two phrases when the b_j = 1 and p_3 be the phrase when b_j = 0
            Sample b_j ∝ e^{Σ_i v'_{c_i} · v_{p_i}}
            if b_j = 1 then
                Update v_w for w ∈ p_1 according to the compositionality model
```

where $P(c_i|p_i)$ is modeled with the standard skip-gram model.

$$P(c_i|p_i) = \prod_{p_j inc_i} \frac{1}{Z} e^{v'_{p_j} v_{p_i}}$$

where Z is the normalization constant which sums over the whole phrase vocabulary Q.

$$Z = \sum_{p \in Q} e^{v'_p v_{p_i}}$$

The summation over the whole vocabulary is impractical. One alternative is to use negative sampling which approximate the sum over the whole vocabulary with a few samples.

$$Z = \sum_{j}^{v'_{p_j} v_{p_i}}$$

where p_js are negative samples sampled from the same uniform distribution described in the previous sections.

We sample the binary value b_j according to the following probabilities:

$$P(b_j = 1) \propto P(c_1|p_1) \cdot P(c_2|p_2) \cdot P(c')$$

$$P(b_j = 0) \propto P(c_3|p_3) \cdot P(c")$$

where $P(c')$ and $P(c")$ evaluate the change of contexts for the surrounding phrases when the binary value varies.

5.6 Evaluation

We evaluate results on the phrase similarity task described in Mitchell and Lapata (2008).

5.7 Conclusion

In this section, we have introduced a joint model for learning phrase embeddings and phrase compositionality. To overcome the limits of the the short length of each phrase and to increase flexibility of phrase boundaries, we propose an MCMC sampling schedule for additionally learning phrase boundary information.

6 Neural machine translation

Neural machine translation.

7 Conclusion

In this paper, I have proposed various applications of MCMC algorithms.

Bibliography

Geman, Stuart and Donald Geman. 1984. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):721–741.

Hastings, W Keith. 1970. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109.

Metropolis, Nicholas and Stanislaw Ulam. 1949. The monte carlo method. *Journal of the American statistical association*, 44(247):335–341.

Mitchell, Jeff and Mirella Lapata. 2008. Vector-based models of semantic composition. In *In Proceedings of ACL-08: HLT*, pages 236–244.

Yu, Heng, Liang Huang, Haitao Mi, and Kai Zhao. 2013. Max-violation perceptron and forced decoding for scalable MT training. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1112–1123. Association for Computational Linguistics, Seattle, Washington, USA.