The Effect of Dependency Representation Scheme on Syntactic Language Modelling

Sunghwan Mac Kim
Department of Computing
Macquarie University
Sydney, NSW 2109, Australia
sunghwan.kim@mq.edu.au

John K Pate
Department of Computing
Macquarie University
Sydney, NSW 2109, Australia
john.pate@mq.edu.au

Mark Johnson
Department of Computing
Macquarie University
Sydney, NSW 2109, Australia
mark.johnson@mq.edu.au

Abstract

There has been considerable work on syntactic language models and they have advanced greatly over the last decade. Most of them have used a probabilistic context-free grammar (PCFG) or a dependency grammar (DG). In particular, DG has attracted more and more interest in the past years since dependency parsing has achieved great success. While much work has evaluated the effects of different dependency representations in the context of parsing, there has been relatively little investigation into them on a syntactic language model. In this work, we conduct the first assessment of three dependency representations on a transition-based dependency parsing language model. We show that the choice of dependency representation has an impact on overall performance from the perspective of language modelling.

1 Introduction

Syntactic language models have been successfully applied to a wide range of domains such as speech recognition, machine translation, and disfluency detection. Although n-gram based language models are the most widely used due to their simplicity and efficacy, they suffer from a major drawback: they cannot characterise the long-range relations between words. A syntactic language model, which exploits syntactic dependencies, can incorporate richer syntactic knowledge and information through syntactic parsing. In particular, syntactic structure leads to better performance of language model compared to traditional n-gram language models (Chelba and Jelinek, 2000). Most of the syntactic language models have used a probabilistic context-free grammar (PCFG) (Roark, 2001; Charniak, 2001) or a dependency grammar (DG) (Wang and Harper, 2002; Gubbins and Vlachos, 2013) in order to capture the surface syntactic structures of sentences.

Researchers have shown an increased interest in dependency parsing and it has increasingly been recognised as an alternative to constituency parsing in the past years. Accordingly, various representations and associated parsers have been proposed with respect to DG. DG describes the syntactic structure of a sentence in terms of head-dependent relations between words. Unlike constituency models of syntax, DG directly models relationships between pairs of words, which leads to a simple framework that is easy to lexicalise and parse with. Moreover, a DG-based, particularly transition-based, parser is fast and even achieves state-of-the-art performance compared to the other grammar-based parsers. It is therefore suitable for identifying syntactic structures in incremental processing, which is a useful feature for online processing tasks such as speech recognition or machine translation.

The aim of this study is to explore different dependency representations and to investigate their effects on the language modelling task. There are publicly available converters to generate dependencies. Ivanova et al. (2013) investigated the effect of each dependency scheme in terms of parsing accuracy. Elming et al. (2013) evaluated four dependency schemes in five different natural language processing (NLP) applications. However, to our knowledge, no previous work has investigated the effect of dependency representation on a syntactic language model.

The remainder of this paper is organised as follows. Section 2 gives a brief overview of related work on dependency schemes and language modelling. In Section 3 we discuss three dependency schemes and Section 4 describes our dependency parsing language model. Then, the experimental settings and a series of results are presented in Section 5. Finally, conclusions and directions for future work are given in Section 6.

2 Related Work

A number of studies have been conducted on dependency representations in NLP tasks. Several constituent-to-dependency conversion schemes have been proposed as the outputs of the converters (Johansson and Nugues, 2007; de Marneffe and Manning, 2008; Choi and Palmer, 2010; Tratz and Hovy, 2011). Previous work has evaluated the effects of different dependency representations in various NLP applications (Miwa et al., 2010; Popel et al., 2013; Ivanova et al., 2013; Elming et al., 2013). A substantial literature has examined the impact of combining DG with another diverse grammar representation, particularly in the context of parsing (Sagae et al., 2007; Øvrelid et al., 2009; Farkas and Bohnet, 2012; Kim et al., 2012).

Many works on syntactic language models have been carried out using phrase structures. Chelba and Jelinek (2000) experiment with the application of syntactic structure in a language model for speech recognition. Their model builds the syntactic trees incrementally in a bottom-up strategy while processing the sentence in a left-to-right fashion and assigns a probability to every word sequence and parse. The model is very close to the arc-standard model that we investigate in this paper. Roark (2001) implements an incremental top-down and left-corner parsing model, which is used as a syntactic language model for a speech recognition task. The model effectively exploits rich syntactic regularities as features and achieves better performance than an *n*-gram model. Charniak (2001) describes a syntactic language model based on immediate-head parsing, which is called a Trihead model and empirically shows that the Trihead model is superior to both a trigram baseline and two previous syntactic language models.

Wang and Harper (2002) present a syntactic DG-based language model (SuperARV) for speech recognition. Multiple knowledge sources are tightly integrated based on their constraint DG but SuperARV does not construct explicit syntactic dependencies between words. Nonetheless, it achieves better perplexity than both a baseline trigram and other syntactic language models. Recently, Gubbins and Vlachos (2013) showed how to use unlabelled and labelled dependency grammar language models to solve the Sentence Completion Challenge set (Zweig and Burges, 2012). Their models performed substantially better than *n*-gram models.

3 Alternative Dependency Representations

Previous work has defined different dependency schemes, and provided software tools for converting the syntactic constituency annotations of existing treebanks to dependency annotations. We experiment with the following three schemes for the language modelling task.

- LTH: The LTH dependency scheme is extracted from the automatic conversion of Penn Treebank using the LTH converter (Johansson and Nugues, 2007).[1] The converter has a lot of options that generate linguistic variations in dependency structures and it was configured to produce a functional rather than lexical DG in this study.

- P2M: The P2M scheme is obtained by running the Penn2Malt converter (Nivre, 2006)[2] based on a standard set of head rules. This converter was deprecated by the LTH converter but it is still used to make a comparison with previous results.

- STD: The STD scheme is used in the Stanford parser (de Marneffe and Manning, 2008)[3], which comes with a converter. In this study, the Penn Treebank was converted to Stanford basic dependencies for projective dependency parsing.

The syntactic representations differ in systematic ways as shown in Figure 1. For instance, auxiliaries take the lexical verb as a dependent in all schemes except for STD, where the lexical verb is the head of a VP. In addition to the typological differences, there is rich variation in dependency labels with respect to the schemes. The STD scheme has 49 dependency labels, which is the largest set

[1]http://nlp.cs.lth.se/software/treebank_converter/

[2]http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html

[3]http://nlp.stanford.edu/software/lex-parser.shtml

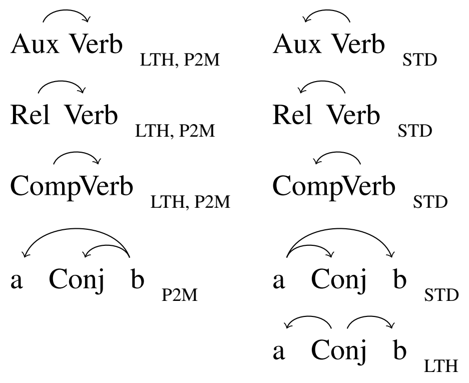


Figure 1: Auxiliaries, relative/subordinate clauses and coordination in the DG schemes.

of dependency labels used here. The LTH dependencies are defined by 22 label types, whereas the P2M scheme employs 12 labels. The fine-grained set of dependency labels in STD allows a more precise expression of grammatical relations. For instance, the label *adv* (unclassified adverbial) in LTH can be expressed using either *advcl* (adverbial clause modier) or *advmod* (adverbial modier) in STD. We evaluate the dependency schemes by incorporating them into a language model architecture in the experiments.

4 A Dependency Parsing Language Model

In this section we describe our syntactic language model in terms of parsing and language modelling. Using the theoretical framework for generative transition-based dependency parsing introduced by Cohen et al. (2011), we propose a generative version of the arc-standard model that defines probability distributions over transitions and finds the most probable ones given stack features.

4.1 Generative Dependency Parsing Model

Following Nivre (2008), a transition-based dependency parsing model is defined as a tuple $S = (C, T, I, C_t)$, where C is a set of configurations, T is a set of permissible transitions, I is an initialisation function, and C_t is a set of terminal configurations. A transition sequence for a sentence is a sequence of configurations where each non-initial configuration is obtained by applying a transition to a previous configuration. A configuration is a triple (α, β, A), where α is stack, β is queue and A is a set of dependency arcs. The stack α stores partially processed words and the queue β records the remaining input words, respectively. The function I maps a sentence to an initial configuration with empty α and A, and β containing the words of the sentence. The set C_t has a terminal configuration, where α contains a single word and β is empty.

The arc-standard model has three distinct types of transitions as follows:

- $\textit{Shift}_{pw}(SH_{pw})$: move the first item i in the queue onto the stack and predict POS p and word w for the item
- $\textit{Left-Arc}_l(LA_l)$: combine the top two items i, j on the stack and predict a dependency label l over an arc $j \rightarrow i$
- $\textit{Right-Arc}_l(RA_l)$: combine the top two items i, j on the stack and predict a dependency label l over an arc $i \rightarrow j$

This model processes the input sentence from left to right in a bottom-up fashion using three transitions. A parse tree is incrementally derived through the transition sequence. In particular, *Shift* predicts the next POS p and word w in the queue with a probability $P(\textit{Shift}, p, w)$, which is a probability both of the *Shift* transition and the POS/word prediction.

The probability of a parse tree is defined as the product of the probabilities of transitions in Eq (1).

$$P(\pi) = \prod_{i=1}^{2n-1} P(t^i|\alpha^{i-1}, A^{i-1}) \tag{1}$$

where π is a parse tree, n is the number of words in a sentence and t is a transition such that $t \in T$.

More specifically, transitions are conditioned on topmost stack items and in particular, the probability of *Shift* is defined as follows:

$$\begin{aligned} &P(Shift_{pw}|\alpha, A) \\ &\quad = P(Shift|\alpha, A) \cdot P(p|Shift, \alpha, A) \\ &\qquad \cdot P(w|p, Shift, \alpha, A) \end{aligned} \tag{2}$$

This factored generative approach alleviates the effect of sparsity. More specifically, first a relatively coarse-grained *Shift* is predicted given stack features and then we predict a fine-grained POS tag for the transition. Then a more fine-grained prediction of a word is made for the POS to further decompose the *Shift* prediction.

For labelled dependency parsing, we simultaneously generate the dependency and its label. In contrast, *Left-Arc* and *Right-Arc* are predicted in unlabelled dependency parsing without considering the dependency label. For instance, as explained in Section 3, 22 dependency labels are in the LTH scheme and thus 45 combinations of transitions and labels are jointly learned and predicted including the *Shift* transition, which does not have a label, in the labelled parsing model. Note that the probability of *Shift* transition is estimated in a factored fashion as described above.

The above probabilities are subject to the following normalisation condition, namely the sum of all transition probabilities should be one:

$$\sum_{p \in P} \sum_{w \in W} P(Shift_{pw}|\alpha, A) + \sum_{l \in L} P(Left\text{-}Arc_l|\alpha, A) + \sum_{l \in L} P(Right\text{-}Arc_l|\alpha, A) = 1$$

where P and W stand for predefined POS tag and word vocabularies, respectively. L is a set of dependency labels corresponding to each dependency scheme.

4.2 Beam Search

Our parsing model maintains a beam containing multiple configurations. To allow the parser to consider configurations with expensive early moves but cheap late moves, we sort our configurations by a figure-of-merit (FOM) that adjusts a configuration's probability with heuristics about. The beam search runs in each word position using a separate priority queue. A priority queue contains configurations that have been constructed by the same numbers of *Shift*, and the same or different numbers of *Left-Arc* or *Right-Arc* transitions. An initial configuration is inserted into the first priority queue corresponding to the first word position and then the algorithm loops until all configurations have terminal conditions. Each configuration is populated off the queue and updated by applying each of permissible transitions to the current configuration. The updated configuration is pushed onto the corresponding queue with respect to the number of *Shift*. The configurations in each priority queue are ranked according to the FOMs. The ranked configurations are discarded if they fall outside the beam. Looping continues for the current word position until the queue is empty. For the configurations in the final queue, we compute the probability of an input sentence by summing over all their probabilities.

Configurations in the same queue cover different amounts of the input string since they could have different numbers of *Left-Arc* or *Right-Arc*. The configurations are not comparable to each other in terms of their probabilities. For this reason, we propose a FOM to penalise the case where the stack has more items. To this end, the FOM incorporates the probability cost of the reductions that will be required to reduce all the items on the stack. An estimate of the probability of reductions uses a rough approximation $1/(4L)^m$, L is the number of labels (if the scheme is unlabelled, L=1) and m is the number of stack items. The FOM is estimated by multiplying the probability of a configuration by the predicted probability of reducing every stack item.

4.3 Random Forest Model for Word Distribution

The probability of a word in Eq (2) is calculated from the word RF model. The training procedure of our RF model is similar to that of Xu and Jelinek (2007). The growing algorithm starts with a root node and a decision tree is constructed by splitting every node recursively from the root node through a two-step splitting rule. Node splitting terminates when each node triggers a stopping rule in order to avoid overfitting.

We overcome data sparsity by first observing that a decision tree essentially provides a hierarchical clustering of the data, since each node splits the data. We linearly interpolate the distributions of the leaf nodes recursively with their parents, ultimately backing off to the root node. The interpolation parameters should be sensitive to the counts in each node so that we back off only when the more specific node does not have enough observations to be reliable. However, giving each node of each decision tree its own interpolation parameter would itself introduce data sparsity problems. We instead use Chen's bucketing approach (Chen and Goodman, 1996) for each tree level, in which nodes on the same level are grouped into buckets and one interpolation parameter is estimated for each bucket. The first step is to divide up the nodes on the same level into bins based on Chen's scores. We then use the EM algorithm to find the optimal interpolation parameter for all the nodes in each bucket using heldout data.

In a RF model, the predictions of all decision trees are averaged to produce a more robust pre-

$S1_p, S2_p, S3_p$	POS
$S1_w, S2_w, S3_w$	word
$S1_{lp}, S2_{lp}$	POS of left-most child
$S1_{rp}, S2_{rp}$	POS of right-most child
$S1_{lw}, S2_{lw}$	word of left-most child
$S1_{rw}, S2_{rw}$	word of right-most child
$S1^*_{ll}, S2^*_{ll}$	Label of left-most child
$S1^*_{rl}, S2^*_{rl}$	Label of right-most child

Table 1: Conditioning stack features, where Si represents the i^{th} item on the stack (S1 is a top-most stack item). Note that dependency label features are only used in labelled parsing.*

diction, as in Eq (3):

$$P(w|p, Shift, \alpha, A) = \frac{1}{m}\sum_{t=1}^{m} P^j(w|p, Shift, \alpha, A) \quad (3)$$

where m denotes the total number of decision trees and $P^j(w|\cdot)$ is the probability of the word w calculated by the j^{th} decision tree.

The word probability is calculated conditioned on the stack features of the current configuration. Table 1 shows the stack features that are used in our RF model. Our unlabelled parsing model estimates the conditional probabilities of the word, given 14 different features. The labelled parsing model uses four additional label-related features.

4.4 Maximum Entropy Model for Transition/POS Distribution

Conditional ME models (Berger et al., 1996) are used as another classifier in our parsing-based language model together with RF. The probability of a transition or a POS tag in Eq (2) is calculated from the corresponding transition ME or POS ME model. For instance, the probability of *Shift* is calculated as shown in Eq (4).

$$P^{ME}(Shift|\alpha, A) = \frac{1}{Z(\alpha, A)}\exp(\lambda \cdot f(\alpha, A, Shift)) \quad (4)$$

where $f(\alpha, A, Shift)$ denotes feature functions that return non-zero values if particular stack items appear in (α, A) and the transition is *Shift*. λ is the corresponding real-valued weight vector, and more informative features receive weights further from zero. $Z(\alpha, A) = \sum_{t \in T}\exp(\lambda \cdot f(\alpha, A, t))$ is the partition function that ensures the distribution is properly normalised. The feature weights λ are tuned to maximise the regularised conditional likelihood of the training data. It is equivalent to the minimisation of the regularised negative log conditional likelihood:

$$\hat{\lambda} = \underset{\lambda}{\operatorname{argmin}}(-\sum_i \log P_\lambda(y_i|x_i) + \sum_j \frac{\lambda_j^2}{2\sigma_j^2}) \quad (5)$$

where $\sum_j \lambda_j^2/2\sigma_j^2$ is a Gaussian prior regulariser that reduces overfitting by penalising large weights. In practice, we use a single parameter σ instead of having a different parameter σ_i for each feature.

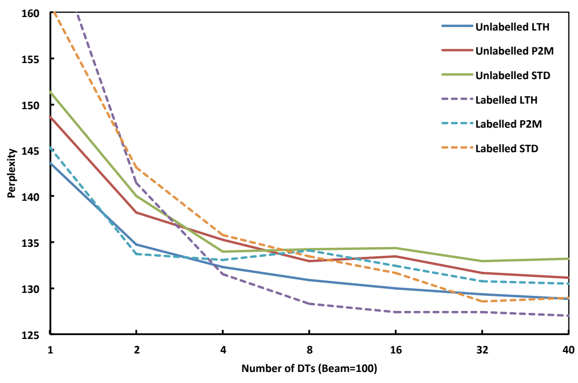
In this work we use the limited memory BFGS (L-BFGS) algorithm (Liu and Nocedal, 1989), which is an efficient numerical optimisation algorithm, to find the optimal feature weights $\hat{\lambda}$. In the ME model, we rely on two kinds of features: (1) atomic features from Table 1, and (2) conjunctive features that are combinations of the atomic features. The feature templates are shown in Table 2.

Type	**Features**
Unigram	$S1_p, S2_p, S3_p, S1_w, S2_w, S3_w$
	$S1_{lp}, S2_{lp}, S1_{rp}, S2_{rp}$
	$S1_{lw}, S2_{lw}, S1_{rw}, S2_{rw}$
	$S1^*_{ll}, S2^*_{ll}, S1^*_{rl}, S2^*_{rl}$
Bigram	$S1_w \circ S1_p, S2_w \circ S2_p, S3_w \circ S3_p$
	$S1_w \circ S2_w, S1_p \circ S2_p$
	$S1_p \circ S1_{lp}, S2_p \circ S2_{lp}$
	$S1_p \circ S1_{rp}, S2_p \circ S2_{rp}$
	$S1_w \circ S1^*_{ll}, S1_w \circ S1^*_{rl}$
	$S1_p \circ S1^*_{ll}, S1_p \circ S1^*_{rl}$
	$S2_w \circ S2^*_{ll}, S2_w \circ S2^*_{rl}$
	$S2_p \circ S2^*_{ll}, S2_p \circ S2^*_{rl}$
Trigram	$S1_p \circ S2_p \circ S3_p, S1_p \circ S2_w \circ S2_p$
	$S1_w \circ S2_w \circ S2_p, S1_w \circ S1_p \circ S2_p$
	$S1_w \circ S1_p \circ S2_w, S2_p \circ S2_{lp} \circ S1_p$
	$S2_p \circ S2_{rp} \circ S1_p, S2_p \circ S1_p \circ S1_{lp}$
	$S2_p \circ S1_p \circ S1_{rp}, S2_p \circ S2_{lp} \circ S1_w$
	$S2_p \circ S2_{rp} \circ S1_w, S2_p \circ S1_w \circ S1_{lp}$
Fourgram	$S1_w \circ S1_p \circ S2_w \circ S2_p$

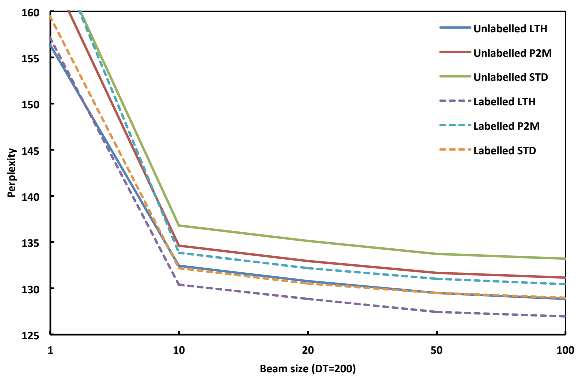
Table 2: Stack feature templates of the ME model.

4.5 Language Model

A generative parsing model assigns a joint probability$P(\pi, s)$ for a parse tree π and an input sentence s. The probability of a sentence $P(s)$ is computed by summing over all parse trees, $P(s) = \sum_\pi P(\pi, s)$. Therefore, our generative parser can be used as a language model, which assigns a probability $P(s)$ to a sentence s, using the probabilities of parse trees from Eq (1).

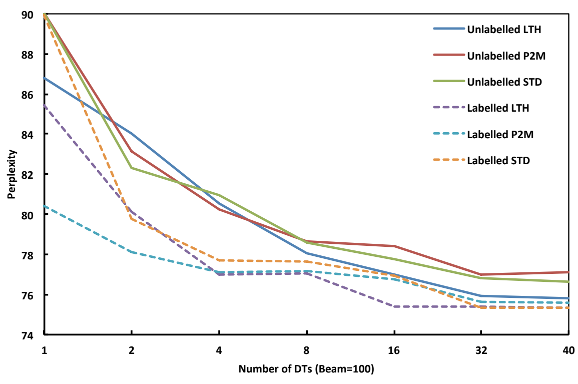


(a) a fixed beam size 100

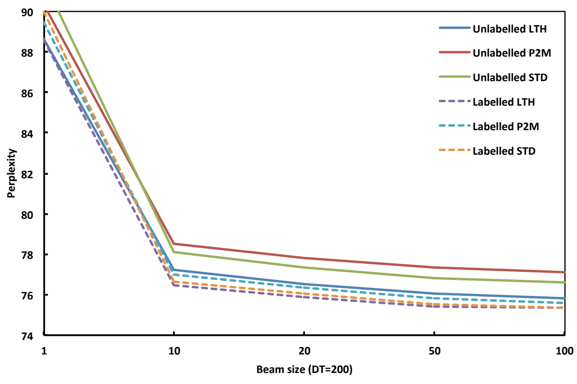


(b) a fixed number of decision trees 40

Figure 2: Perplexity results trained and tested on WSJ.



(a) a fixed beam size 100



(b) a fixed number of decision trees 40

Figure 3: Perplexity results trained and tested on Switchboard.

5 Experiments

5.1 Experimental Settings

We empirically evaluate the effects of three dependency schemes (LTH, P2M, STD) on our dependency language models in terms of perplexity and word error rate (WER). Several experiments are conducted varying the number of decision trees and the beam size on each dependency scheme. Each dependency language model uses a maximum beam size of 100, and up to 40 decision trees are generated for the RF classifier. We train the transition/POS ME classifiers using 400 iterations with no frequency cutoff for features and σ for the Gaussian prior is set to 1. Our experiments use three evaluation datasets for unlabelled and labelled dependency language models.

Perplexity Results trained and tested on WSJ: Most of the syntactic language models (Roark, 2001; Charniak, 2001; Xu and Jelinek, 2007; Wang and Harper, 2002) were evaluated on the Wall Street Journal (WSJ) section of the Penn Treebank (Marcus et al., 1993) in terms of perplexity. The WSJ is speechified by following the conventions of previous work. All punctuation is removed, words are lowercased, and numbers are replaced by a symbol N. All words outside the vocabulary limit (10,000 words) are mapped to a special UNK symbol. Sections 0-20 are used as the training set, sections 21-22 as the heldout set, and sections 23-24 for testing.

Perplexity Results trained and tested on Switchboard: We ran experiments on the Switchboard part of the Penn Treebank. Following Johnson and Charniak (2004), both sections 2 and 3 were used for a training set (sw2005.mrg-sw3993.mrg). We split section 4 into a heldout set (sw4519.mrg-sw4936.mrg) and a test set (sw4004.mrg-sw4153.mrg). The Switchboard corpus was preprocessed so that all disfluencies are removed from constituent trees prior to converting to dependency structure. We applied three converters on the cleaned version of Switchboard to obtain three dependency schemes. The vocabulary consists of all the words in the training data and contains 13,706 word types.

WER Results trained on WSJ and tested on HUB1: We performed a small speech recognition evaluation on *n*-best lists from the DARPA '93 HUB1 test setup. A real task-based evaluation would be worthwhile besides perplexity since previous studies found that lower perplexity does not necessarily mean lower WER in their models (Roark, 2001; Wang and Harper, 2002; Xu and Jelinek, 2007). The HUB1 corpus consists of 213 sentences taken from the WSJ with a total of 3,446 words. The corpus is provided along with a lattice trigram model, which is trained on approximately 40 million words with a vocabulary of 20 thousand words. We used the A* decoding algorithm of SRILM to extract 50-best lists from the lattices measuring the lattice trigram and acoustic scores. The average number of candidates is roughly 21.7 in the lists. The WER of the lattice trigram model is 13.7% and the oracle WER, which is the lowest WER to the references, is 7.9% for the 50-best lists. There are token discrepancies between the Penn Treebank and the HUB1 lattices for contractions, possessives and numbers.[4] For simplicity, we do not speechify the numbers (i.e., not expand them into text), whereas we follow previous work in dealing with the discrepancies of contractions or possessives. We use the Penn Treebank tokenisation, which separates clitics from their base word (i.e. '*can't*' is represented as '*ca n't*'), for training and running our models. Two tokens of treebank format are then combined into one to be aligned with gold standard word sequences for the WER evaluation. Our trained models were used with the same training data (1 million words) and vocabulary as in the above perplexity experiments on WSJ. We followed Roark (2001) in multiplying the language model scores by 15 before interpolating them with the acoustic model scores.

5.2 Experimental Results

The Effects of Decision Trees and Beam Search: To illustrate the effects of decision trees and beam size, we plot perplexity that corresponds to each scheme. As can be seen from Figure 2a, the perplexities are improved for all schemes using more decision trees. Perplexity reductions of two labelled schemes (LTH, STD) are relatively large except for labelled P2M, and particularly the perplexities drop sharply up to random forests of size 8 for labelled LTH. On the other hand, the perplexities of unlabelled schemes are decreased at a sluggish pace and they are more insensitive to the number of decision trees. As Figure 2b illustrates, the general shape of the perplexity curves drops quickly as beam size increases from 1 to 10, and then to flatten as beam size increases further. We do not obtain much benefit if the beam size is larger than 10. Figure 3 shows the perplexity results on Switchboard for different numbers of decision trees and variable beam sizes as on WSJ. We can observe the positive effects of these two parameters (random forest size and beam search) with respect to perplexity. Figure 3b shows similar trends to the WSJ perplexities, which are dramatically reduced up to a beam size of 10, and then level out. The perplexity trends in Figure 3a are quite different, showing that unlabelled schemes have a gentle slope over numbers of decision trees. The number of decision trees has a relatively high impact on the performance of unlabelled schemes on Switchboard.

Performance Comparison: Table 3 presents the perplexity and WER results of unlabelled and labelled schemes for LTH, P2M and STD. We can see that labelled LTH achieves the lowest perplexities, whereas unlabelled P2M and STD perform the worst overall. For labelled schemes we observe their superior performance compared to unlabelled schemes in terms of perplexity. The perplexity results indicate that dependency labels improve the performance of a dependency language model. In particular, labelled STD has the largest perplexity reduction (3.2%) compared to unlabelled STD, which yields the worst perplexity on WSJ. Overall, LTH outperforms both P2M and STD regardless of unlabelled or labelled scheme in terms of perplexity. Although the WERs are not significantly different across the different methods, unlabelled P2M leads to the best performance (14.6%), whereas labelled P2M is the worst performing scheme (15.1%).[5] Labelled LTH and

[4]For instance, "**Bill 's** car **isn 't** worth **$100**" vs. "**Bill's** car **isn't** worth **one hundred dollars**".

[5]All results of statistical significance tests are presented with $p < 0.05$ level using the SCLITE toolkit with the option MAPSSWE, which stands for Matched Pairs Sentence Segment Word Error.

Models	WSJ Perplexity	SWBD Perplexity	HUB1 WER
Unlabelled LTH	128.84	75.81	14.8
Unlabelled P2M	131.12	77.09	14.6
Unlabelled STD	133.18	76.61	14.7
Labelled LTH	126.97	75.33	14.9
Labelled P2M	130.45	75.59	15.1
Labelled STD	128.96	75.34	14.7

Table 3: Perplexities and WERs of unlabelled and labelled LTH, P2M and STD schemes. We measured their performance on WSJ, Switchboard and HUB1 datasets.

STD perform roughly on par with unlabelled LTH and STD, respectively.

Analysis and Discussion: Our results indicate that the choice of dependency representation affects the performance of the syntactic language model. One thing to note is that it is hard to tell what scheme is overall best in our experiments. An interesting observation is that the dependency labels are somewhat ineffective for the speech recognition task in contrast to perplexity evaluation. The results demonstrate that each evaluation measure tends to have its own preferred scheme. For instance, the LTH scheme is preferred for the perplexity evaluation, whereas STD is preferred under WER. Our finding is in line with previous work, which claims that a task-based evaluation does not correlate well with a theoretical evaluation (Rosenfeld, 2000; Jonson, 2006; Och et al., 2004; Miwa et al., 2010; Elming et al., 2013; Smith, 2012). They commonly claim that lower perplexity does not necessarily mean lower WER, and the relation between two measures is clearly not transparent. Miwa et al. (2010) found that STD performs better for event extraction, whereas LTH outperforms STD in terms of parsing accuracy.

5.3 Comparison with Previous Work

In this section, we compare our model to previous work discussed in the literature. As baselines, we use two influential models, namely a modified Kneser-Ney (mKN) trigram model (Chen and Goodman, 1998) and Charniak's Trihead language model (Charniak, 2001).[6] The two models were trained with the same training data and vocabulary on WSJ as mentioned in Section 5.1. The SRILM toolkit (Stolcke, 2002) was used to build the trigram mKN smoothed language model.

[6]The mKN is still considered one of the best smoothing methods for *n*-gram language models and the Trihead model achieves the better perplexity and WER compared to Chelba's and Roark's models (Lease et al., 2005). Moreover, there is no significant difference between Chelba's SLM and Wang's SuperARV in terms of WER although the SuperARV even obtains a much lower perplexity (Wang and Harper, 2002). For these reasons, we think that Charniak's model is the strongest competitor among syntactic language models.

Models	Perplexity	WER
mKN trigram	148.65	17.2
Trihead	131.40	15.0
Unlabelled LTH	128.84	14.8
Labelled LTH	126.97	14.9

Table 4: Perplexity and WER comparisons with previous work.

The performance of the two baselines in Table 4 was measured conducting the same test procedures on WSJ and HUB1. Our perplexities and WERs were selected from the previous sections for the LTH scheme, which performs well uniformly. It is shown that both unlabelled and labelled LTH schemes significantly improve upon the mKN baseline in terms of WER. The WER improvement of LTH over the Trihead model is not statistically significant, although the LTH scheme achieves better perplexity than Trihead regardless of dependency labels.

6 Conclusion and Future Work

We explored three different dependency schemes using a dependency parsing language model. Our study indicates that the choice of scheme has an impact on overall performance. However, no dependency scheme is uniformly better than the others in terms of both perplexity and WER. Nevertheless, there are some generalisations we can make. When evaluated on WSJ and Switchboard, unlabelled and labelled LTHs are generally better than the others in terms of perplexity. In contrast, unlabelled and labelled STDs yield the best overall performance in the HUB1 WER evaluation. It is interesting to see that perplexity has a weak correlation with WER in dependency parsing language models. We note that it is hard to figure out from our results which dependency directions are preferred in structures such as prepositional phrase attachment. For instance, "Is the rightmost noun favoured? or Is the leftmost noun favoured?" as a head in noun sequences in the context of language modelling. As future work, we plan to carry out further investigation of the effect of each structure and explore what is the most or least preferable combination of structures on a syntactic language model.

Acknowledgments

We would like to thank the reviewers for their thoughtful comments and suggestions.

References

Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996. A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39–71.

Eugene Charniak. 2001. Immediate-Head Parsing for Language Models. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 124–131, Toulouse, France.

Ciprian Chelba and Frederick Jelinek. 2000. Structured language modeling. *Computer Speech & Language*, 14(4):283–332.

Stanley F. Chen and Joshua Goodman. 1996. An Empirical Study of Smoothing Techniques for Language Modeling. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco. Morgan Kaufmann Publishers.

Stanley F. Chen and Joshua Goodman. 1998. An Empirical Study of Smoothing Techniques for Language Modeling. Technical Report TR-10-98, Center for Research in Computing Technology, Harvard University.

Jinho D Choi and Martha Palmer. 2010. Robust constituent-to-dependency conversion for English. In *Proceedings of 9th Treebanks and Linguistic Theories Workshop (TLT)*, pages 55–66.

Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Exact Inference for Generative Probabilistic Non-Projective Dependency Parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1245, Edinburgh, Scotland, UK, July. Association for Computational Linguistics.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford Typed Dependencies Representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK, August.

Jakob Elming, Anders Johannsen, Sigrid Klerke, Emanuele Lapponi, Hector Martinez Alonso, and Anders Søgaard. 2013. Down-stream effects of tree-to-dependency conversions. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 617–626, Atlanta, Georgia, June. Association for Computational Linguistics.

Richárd Farkas and Bernd Bohnet. 2012. Stacking of Dependency and Phrase Structure Parsers. In *Proceedings of COLING 2012*, pages 849–866, Mumbai, India, December. The COLING 2012 Organizing Committee.

Joseph Gubbins and Andreas Vlachos. 2013. Dependency Language Models for Sentence Completion. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1405–1410, Seattle, Washington, USA, October. Association for Computational Linguistics.

Angelina Ivanova, Stephan Oepen, and Lilja Øvrelid. 2013. Survey on parsing three dependency representations for English. In *51st Annual Meeting of the Association for Computational Linguistics Proceedings of the Student Research Workshop*, pages 31–37, Sofia, Bulgaria, August. Association for Computational Linguistics.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for English. In *Proceedings of the 16th Nordic Conference on Computational Linguistics (NODALIDA)*.

Mark Johnson and Eugene Charniak. 2004. A TAG-based noisy channel model of speech repairs. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 33–39.

Rebecca Jonson. 2006. Generating Statistical Language Models from Interpretation Grammars in Dialogue Systems. In *Proceedings of 11th Conference of the European Association of Computational Linguistics*, pages 57–65. The Association for Computer Linguistics.

Sunghwan Mac Kim, Dominick Ng, Mark Johnson, and James Curran. 2012. Improving Combinatory Categorial Grammar Parse Reranking with Dependency Grammar Features. In *Proceedings of COLING 2012*, pages 1441–1458, Mumbai, India, December. The COLING 2012 Organizing Committee.

Matthew Lease, Eugene Charniak, and Mark Johnson. 2005. Parsing and its Applications for Conversational Speech. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 5, pages 961–964.

Dong C. Liu and Jorge Nocedal. 1989. On the Limited Memory BFGS Method for Large Scale Optimization. *Math. Program.*, 45(3):503–528, December.

Michell P. Marcus, Mary A. Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330.

Makoto Miwa, Sampo Pyysalo, Tadayoshi Hara, and Jun'ichi Tsujii. 2010. Evaluating Dependency Representations for Event Extraction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 779–787, Beijing, China, August. Coling 2010 Organizing Committee.

Joakim Nivre. 2006. *Inductive dependency parsing*. Springer.

Joakim Nivre. 2008. Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553.

Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. 2004. A Smorgasbord of Features for Statistical Machine Translation. In Daniel Marcu Susan Dumais and Salim Roukos, editors, *HLT-NAACL 2004: Main Proceedings*, pages 161–168, Boston, Massachusetts, USA, May 2 - May 7. Association for Computational Linguistics.

Lilja Øvrelid, Jonas Kuhn, and Kathrin Spreyer. 2009. Cross-framework parser stacking for data-driven dependency parsing. *Traitement Automatique des Langues (TAL) Special Issue on Machine Learning for NLP*, 50(3):109–138.

Martin Popel, David Mareček, Jan Štpánek, Daniel Zeman, and Zdnk Žabokrtský. 2013. Coordination Structures in Dependency Treebanks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 517–527, Sofia, Bulgaria, August. Association for Computational Linguistics.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Ronald Rosenfeld. 2000. Two decades of statistical language modeling: where do we go from here? *Proceedings of the IEEE*, 88(8):1270–1278, Aug.

Kenji Sagae, Yusuke Miyao, and Jun'ichi Tsujii. 2007. HPSG Parsing with Shallow Dependency Constraints. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 624–631, Prague, Czech Republic, June. Association for Computational Linguistics.

Noah A. Smith. 2012. Adversarial Evaluation for Models of Natural Language. *CoRR*, abs/1207.0245.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing*, pages 901–904.

Stephen Tratz and Eduard Hovy. 2011. A Fast, Accurate, Non-Projective, Semantically-Enriched Parser. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1257–1268, Edinburgh, Scotland, UK, July.

Wen Wang and Mary P. Harper. 2002. The SuperARV Language Model: Investigating the Effectiveness of Tightly Integrating Multiple Knowledge Sources. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 238–247. Association for Computational Linguistics, July.

Peng Xu and Frederick Jelinek. 2007. Random forests and the data sparseness problem in language modeling. *Computer Speech & Language*, 21(1):105–152.

Geoffrey Zweig and Chris J.C. Burges. 2012. A Challenge Set for Advancing Language Modeling. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 29–36, Montréal, Canada, June. Association for Computational Linguistics.