A Beam-Search Decoder for Disfluency Detection

Xuancong Wang[1,3] **Hwee Tou Ng**[1,2] **Khe Chai Sim**[2]
[1]NUS Graduate School for Integrative Sciences and Engineering
[2]Department of Computer Science, National University of Singapore
[3]Human Language Technology, Institute for Infocomm Research, Singapore
xuancong84@gmail.com, {nght, simkc}@comp.nus.edu.sg

Abstract

In this paper[1], we present a novel beam-search decoder for disfluency detection. We first propose node-weighted max-margin Markov networks (M3N) to boost the performance on words belonging to specific part-of-speech (POS) classes. Next, we show the importance of measuring the quality of cleaned-up sentences and performing multiple passes of disfluency detection. Finally, we propose using the beam-search decoder to combine multiple discriminative models such as M3N and multiple generative models such as language models (LM) and perform multiple passes of disfluency detection. The decoder iteratively generates new hypotheses from current hypotheses by making incremental corrections to the current sentence based on certain patterns as well as information provided by existing models. It then rescores each hypothesis based on features of lexical correctness and fluency. Our decoder achieves an edit-word F1 score higher than all previous published scores on the same data set, both with and without using external sources of information.

1 Introduction

Disfluency detection is a useful and important task in Natural Language Processing (NLP) because spontaneous speech contains a significant proportion of disfluency. The disfluencies in speech introduce noise in downstream tasks like machine translation and information extraction. Thus, the task of disfluency detection not only can help improve the readability of automatically transcribed speech, but also the performance of downstream NLP tasks.

There are mainly two types of disfluencies: filler words and edit words. Filler words include filled pauses (e.g., 'uh', 'um') and discourse markers (e.g., "I mean", "you know"). They are insertions in spontaneous speech that indicate pauses or mark boundaries in discourse. Thus, they do not convey useful content information. Edit words are words that are spoken wrongly and then corrected by the speaker. For example, consider the utterance:

Edit Filler Repair

I want a flight ~~*to Boston*~~ *uh I mean* to Denver

The phrase "to Boston" forms the edit region to be replaced by "to Denver". The words "uh I mean" are filler words that serve to cue the listener about the error and subsequent correction. So, the cleaned-up sentence would be "I want a flight to Denver", which is what the speaker originally intended to say. In general, edit words are more difficult to detect than filler words, and so edit word prediction accuracy is much lower. Thus, in this work, we mainly focus on edit word detection.

In Section 2, we briefly introduce previous work. In Section 3, we describe our improved baseline system that will be integrated into our beam-search decoder. Section 4 presents our beam-search decoder in detail. In Section 5, we describe our experiments and results. Section 6 gives the conclusion.

[1]The research reported in this paper was carried out as part of the PhD thesis research of Xuancong Wang at the NUS Graduate School for Integrated Sciences and Engineering.

This work is licensed under a Creative Commons Attribution 4.0 International Licence. Page numbers and proceedings footer are added by the organisers. Licence details: http://creativecommons.org/licenses/by/4.0/

Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers, pages 1457–1467, Dublin, Ireland, August 23-29 2014.

2 Previous Work

Researchers have tried many models for disfluency detection. Johnson and Charniak (2004) proposed a TAG-based (Tree-Adjoining Grammar) noisy channel model, which showed great improvement over a boosting-based classifier (Charniak and Johnson, 2001). Maskey et al. (2006) proposed a phrase-level machine translation approach for this task. Liu et al. (2006) used conditional random fields (CRFs) (Lafferty et al., 2001) for sentence boundary and edit word detection. They showed that CRFs significantly outperformed maximum entropy models and hidden Markov models (HMM). Zwarts and Johnson (2011) extended this model using minimal expected F-loss oriented n-best reranking. Georgila (2009) presented a post-processing method during testing based on integer linear programming (ILP) to incorporate local and global constraints. In addition to textual information, prosodic features extracted from speech have been incorporated to detect edit words in some previous work (Kahn et al., 2005; Liu et al., 2006; Zhang et al., 2006). Zwarts and Johnson (2011) also trained extra language models on additional corpora, and compared the effects of adding scores from different language models as features during reranking. They reported that the language models gained approximately 3% in F1-score for edit word detection on the Switchboard development dataset. Qian and Liu (2013) proposed multi-step disfluency detection using weighted max-margin Markov networks (M3N) and achieved the highest F-score of 84.1% without using any external source of information. In this paper, we incorporate the M3N model into our beam-search decoder framework with some additional features to further improve the result.

3 The Improved Baseline System

Weighted max-margin Markov networks (M3N) (Taskar et al., 2003) have been shown to outperform CRF in (Qian and Liu, 2013), since it can balance precision and recall easily by assigning different loss penalty to different label misclassification pairs. In this work, we made use of M3N in expanding the search space and rescoring the hypotheses. To facilitate the integration of the M3N system into our decoder framework, we made several modifications that slightly improve the M3N baseline system. Our improved baseline system has two stages: the first stage is filler word prediction using M3N to detect words which can potentially be fillers, and the second stage is joint edit and filler word prediction using M3N. The output of the first stage is passed as features into the second stage. Both stages perform filler word prediction, since we found that joint edit and filler word detection performs better than edit word detection alone as edit words tend to co-occur with filler words, and the first-stage output can be fed into the second stage to extract additional features. We also augmented the M3N toolkit to support additional feature functions, allow weighting of individual nodes, and control the total number of model parameters.[2]

3.1 Node-Weighted and Label-Weighted Max-Margin Markov Networks (M3N)

A max-margin Markov network (M3N) (Taskar et al., 2003) is a sequence labeling model. It has the same structure as conditional random fields (CRF) (Lafferty et al., 2001) but with a different objective function. A CRF is trained to maximize the conditional probability of the true label sequence given the observed input sequence, while an M3N is trained to maximize the difference between the conditional probability of the true label sequence and the incorrectly predicted label sequence (i.e., maximizing the margin). Thus, we can regard M3N as a support vector machine (SVM) analogue of CRF (Suykens and Vandewalle, 1999).

The dual form of the training objective function of M3N is formulated as follows:

$$\begin{aligned} &\min_{\alpha} \frac{1}{2} C \| \sum_{\mathbf{x},\mathbf{y}} \alpha_{\mathbf{x},\mathbf{y}} \Delta \mathbf{f}(\mathbf{x},\mathbf{y}) \|_2^2 + \sum_{\mathbf{x},\mathbf{y}} \alpha_{\mathbf{x},\mathbf{y}} L(\mathbf{x},\mathbf{y}) \\ s.t. \quad &\sum_{\mathbf{y}} \alpha_{\mathbf{x},\mathbf{y}} = 1, \forall \mathbf{x} \qquad \text{and} \qquad \alpha_{\mathbf{x},\mathbf{y}} \geq 0, \forall \mathbf{x},\mathbf{y} \end{aligned} \tag{1}$$

[2]The source code of our augmented M3N toolkit can be downloaded at http://code.google.com/p/m3n-ext/

where $\mathbf{x}$ is the observed input sequence, $\mathbf{y} \in \mathcal{Y}$ is the output label sequence, $\alpha_{\mathbf{x},\mathbf{y}}$ are the dual variables to be optimized, and $C \geq 0$ is the regularization parameter to be tuned. $\Delta\mathbf{f}(\mathbf{x}, \mathbf{y}) = \mathbf{f}(\mathbf{x}, \widetilde{\mathbf{y}}) - \mathbf{f}(\mathbf{x}, \bar{\mathbf{y}})$ is the residual feature vector, where $\widetilde{\mathbf{y}}$ is the true label sequence, $\bar{\mathbf{y}}$ is the predicted label sequence given the model, and $\mathbf{f}(\mathbf{x}, \mathbf{y})$ is the feature vector. It is implemented as a sum over all nodes:

$$\mathbf{f}(\mathbf{x}, \mathbf{y}) = \sum_t \mathbf{f}(\mathbf{x}, \mathbf{y}, t) \tag{2}$$

where t is the position index of the node in the sequence. Each component of $\mathbf{f}(\mathbf{x}, \mathbf{y}, t)$ is a feature function, $f(\mathbf{x}, \mathbf{y}, t)$. For example, $f(w_0$='so', y_0='F', y_{-1}='O', $t)$ has a value of 1 only when the word at node t is 'so', the label at node t is 'F' (filler word), and the label at node $(t-1)$ is 'O' (outside edit/filler region, i.e., fluent). The maximum length of the y history (for this feature function, it is 2 since only y_0 and y_{-1} are covered) is called the *clique order* of the feature. $L(\mathbf{x}, \mathbf{y})$ is the loss function. A standard M3N uses an unweighted hamming loss, which is the number of incorrect nodes:

$$L(\mathbf{x}, \mathbf{y}) = \sum_t \delta(\widetilde{y}_t, \bar{y}_t) \qquad \text{where } \delta(a, b) = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

Qian and Liu (2013) proposed using label-weighted M3N to balance precision and recall by adjusting the penalty on false positives and false negatives, i.e., $v(\widetilde{y}_t, \bar{y}_t)$ in Eqn. 4. In this work, we further extend this technique to individual nodes to train expert models, each specialized in a specific part-of-speech (POS) class. Our loss function is:

$$L(\mathbf{x}, \mathbf{y}) = \sum_t u_c(t) v(\widetilde{y}_t, \bar{y}_t) \delta(\widetilde{y}_t, \bar{y}_t) \qquad \text{where } u_c(t) = \begin{cases} B_c, & \text{if POS}(t) \in S_c \\ 1, & \text{otherwise} \end{cases} \tag{4}$$

where B_c is a factor which controls the extent to which the model is biased to minimize errors on specific nodes, POS(t) is the POS tag of the word at node t, and S_c is the set of POS tags corresponding to that expert class c. We show that by integrating these expert models into our decoder framework, we can achieve further improvement.

3.2 Features

The feature templates for filler word and edit word prediction are listed in Table 1 and Table 2 respectively. w_i refers to the word at the i^{th} position relative to the current node; *window size* is the maximum number of words before and after the current word that the template covers, e.g., $w_{-1}w_0$ with a window size of 4 means $w_{-4}w_{-3}$, $w_{-3}w_{-2}$, ..., w_3w_4; p_i refers to the POS tag at the i^{th} position relative to the current node; $w_{i\sim j}$ refers to any word from the i^{th} position to the j^{th} position relative to the current node; $w_{i,\neq F}$ refers to the i^{th} word (w.r.t. the current node) not being a filler word; the multi-pair comparison function $I(a, b, c, ...)$ indicates whether each pair (a and b, b and c, and so on) are identical, for example, if $a = b \neq c = d$, it will output "101" ('1' for being equal, '0' for being unequal); and *ngram-score* features are the natural logarithm of the following probabilities: $P(w_{-3}, w_{-2}, w_{-1}, w_0)$, $P(w_0|w_{-3}, w_{-2}, w_{-1})$, $P(w_{-3}, w_{-2}, w_{-1})$, $P(\langle/s\rangle|w_{-3}, w_{-2}, w_{-1})$, $P(w_{-3})$, $P(w_{-2})$, $P(w_{-1})$, $P(w_0)$ ("$\langle/s\rangle$" denotes sentence-end). We use language models (LM) in two ways: individual n-gram scores as M3N features, and an overall sentence-level score for rescoring in our beam-search decoder. Our experiments show that this way of using LM gives the best performance.

We set the frequency pruning threshold to 5, so that the resulting model has about the same total number of parameters (7.6M) as (Qian and Liu, 2013). The clique order for each template is determined by considering the total number of features given that template. For example, for pause duration, there are 10 features (after cumulative binning), so we can set its clique order to 3 since there will be $10 \times 3^3 = 270$ weights; but for word 3-grams, there are 5M features, so setting its clique order to 3 or 2 will give rise to too many weights ($5\text{M} \times 3^3 = 135\text{M}$ for order 3; $5\text{M} \times 3^2 = 45\text{M}$ for order 2), thus we will reduce its clique order to 1. The same principle applies to other feature templates.

Feature Template	Window Size	Clique Order
w_0	4	1
$w_{-1}w_0$	4	2
$I(w_i, w_j)$	10	2
$I(w_i, w_j, w_{i+1}, w_{j+1})$	10	2
p_0	4	1
$p_{-1}p_0$	4	2
$p_{-2}p_{-1}p_0$	4	2
$I(p_i, p_j)$	10	2
$I(p_i, p_j, p_{i+1}, p_{j+1})$	10	2
transitions	0	3

Table 1: Feature templates for filler word prediction

Feature Template	Window Size	Clique Order
$w_{-2}w_{-1}w_0$	4	2
$I(w_i, w_j)(w_i$ if w_i=$w_j)$	10	2
$w_0w_{-6\sim-1}$, $w_0w_{1\sim6}$	0	1
$I(p_i, p_j)$	10	3
$I(p_i, p_j)(p_i$ if p_i=$p_j)$	10	3
$p_{-1}w_0$	2	2
$w_{-1}p_0$	2	2
$w_{-2,\neq F}w_{-1,\neq F}$	0	2
$w_{-3,\neq F}w_{-2,\neq F}w_{-1,\neq F}$	0	2
$p_{-2,\neq F}p_{-1,\neq F}$	0	2
$p_{-3,\neq F}p_{-2,\neq F}p_{-1,\neq F}$	0	2
ngram-score features	0	3
pause duration before w_0	0	3
pause duration after w_0	0	3
all features for filler word prediction	same	same

Table 2: Feature templates for edit word prediction

4 The Beam-Search Decoder Framework

4.1 Motivation

There are several limitations in the current M3N or CRF approach. Firstly, current models do not measure the quality of the cleaned-up sentences, i.e., the resulting sentence after removing all predicted filler and edit words. Secondly, one pass of disfluency detection may not be sufficient to detect all disfluencies. Qian and Liu (2013) showed that we can improve the performance significantly by running a second pass of edit detection. Our preliminary experiments also show that additional passes of edit detection further improve the performance. Lastly, we find that edit word detection accuracy differs significantly on words of different POS tags (Table 3). This is because words of different POS tags have different feature distributions. Thus, depending on the POS tag of the current word, the same feature may have different implications for disfluency. For example, consider the feature $I(p_0, p_2)$. When the current word is a determiner, the feature does not strongly suggest an edit word. In "You give me a book, a pen, a pencil, ...", the determiner 'a' gets repeated. However, when the current word is a verb, the feature strongly suggests it is an edit word. In "The hardest thing for us ~~*has been*~~ is to ...", both 'has' and 'is' are third-person singular verbs. Hence, it might be helpful if we train expert models each specialized in detecting edit words belonging to a specific POS and combine them dynamically according to the POS. Motivated by the beam-search decoder for grammatical error correction (Dahlmeier and Ng, 2012) and social media text normalization (Wang and Ng, 2013), we propose a novel beam-search decoder for

disfluency detection to overcome these limitations.

POS	Freq. (%)	Edit F1 (%)
PRP	25.5	92.33
DT	14.2	88.95
IN	10.4	84.45
VBP	8.3	86.88
RB	7.1	81.78
CC	4.6	86.76
BES	4.2	93.37
NN	3.4	52.30
VBD	3.1	86.51
VB	2.1	70.42
VBZ	1.9	79.70
...	...	...

Table 3: Baseline edit F1 scores for different POS tags

4.2 General Framework

The goal of the decoder is to find the best hypothesis for a given input sentence $\mathbf{w}$. A hypothesis $\mathbf{h}$ is a label sequence, one label for every word in the sentence. To find the best hypothesis, the decoder iteratively generates new hypotheses from current ones using a set of *hypothesis producers* and rescores each hypothesis using a set of *hypothesis evaluators*. For each hypothesis produced, the decoder cleans up the sentence by removing all the predicted filler words and edit words so that subsequent operations can act on the cleaned-up sentence $\bar{\mathbf{w}}$ if needed. Each hypothesis evaluator produces a score f which measures the quality of the current hypothesis based on certain aspects of fluency specific to that hypothesis evaluator. The overall score of a hypothesis is the weighted sum of the scores from all the hypothesis evaluators:

$$score(\mathbf{h}, \mathbf{w}) = \sum_i \lambda_i f_i(\mathbf{h}, \mathbf{w}) \tag{5}$$

The weights λ_is are tuned on the development set using minimum error rate training (MERT) (Och, 2003). The decoding algorithm is shown in Algorithm 1.

In our description, h_i denotes the hypothesized label at the i^{th} position; w_i denotes the word at the i^{th} position; $|\mathbf{h}|$ denotes the length of the label sequence; $f_{M3N}(h_i, \mathbf{w})$ denotes the M3N log-posterior probability of the label (at the i^{th} position of hypothesis $\mathbf{h}$) being the hypothesized label; $f_{M3N}(\mathbf{h}, \mathbf{w})$ denotes the normalized joint log probability of hypothesis $\mathbf{h}$ given the M3N model ('normalized' means divided by the length of the label sequence); $\bar{\mathbf{w}}$ denotes the cleaned-up sentence; $f_{LM}(\bar{\mathbf{w}}) = f_{LM}(\mathbf{h}, \mathbf{w})$ denotes the language model score of the sentence $\bar{\mathbf{w}}$ (cleaned up according to hypothesis $\mathbf{h}$) divided by sentence length; and $\bar{\mathbf{h}}$ denotes the sub-hypothesis obtained by running M3N on the cleaned-up sentence with updated features. Note that a sub-hypothesis will have a shorter label sequence if some words are labeled as filler word or edit word in the parent hypothesis. We can obtain $\mathbf{h}$ from $\bar{\mathbf{h}}$ by inserting all predicted filler and edit words from the parent hypothesis into the sub-hypothesis so that its label sequence has the same length as the original sentence.

4.3 Hypothesis Producers

The goal of hypothesis producers is to create a search space for rescoring using various hypothesis evaluators. Based on the information provided by the existing models and certain patterns where disfluencies may occur, we propose the following hypothesis producers for our beam-search decoder:

Confusable-phrase-dictionary: The motivation of using this hypothesis producer is to hypothesize labels for phrases which are commonly misclassified in the development data. We build a dictionary of

Algorithm 1
The beam-search decoding algorithm for a sentence. S: hypothesis stack; $\mathbf{h}$: hypothesis; $\mathbf{f}$: hypothesis evaluator score vector; $\mathbf{\Lambda}$: hypothesis evaluator weight vector

INPUT: a sentence $\mathbf{w}$ with N words
OUTPUT: a sequence of N labels, from {E, F, O}
initialize hypothesis $\mathbf{h}_0$, h_i='O'$\forall i \in [1, N]$
$S_A \leftarrow \{\mathbf{h}_0\}, S_B \leftarrow \emptyset$
for $iter$ = [1, $maxIter$] **do**
  for each $\mathbf{h}$ in S_A **do**
    for each *producer* in hypothesisProducers **do**
      for each $\mathbf{h}'$ in *producer*($\mathbf{h}$) **do**
        compute $\mathbf{f}(\mathbf{h}', \mathbf{w})$ from hypothesisEvaluators
        compute $score(\mathbf{h}', \mathbf{w}) = \mathbf{\Lambda}^\top \cdot \mathbf{f}(\mathbf{h}', \mathbf{w})$
        $S_B \leftarrow S_B + \{\mathbf{h}'\}$
  prune S_B according to $score$
  $S_A \leftarrow S_B, S_B = \emptyset$
return $argmax_{\mathbf{h}}\{score(\mathbf{h}, \mathbf{w})\}, \mathbf{h} \in S_A$

phrases (up to 5 words) and their corresponding true labels by considering the most frequent incorrectly predicted phrases in the development set. During decoding, whenever such a phrase occurs in a sentence and its label is not the same as that in the dictionary, it is changed to that in the dictionary and a new hypothesis is produced. For example, if the phrase "you know" has occurred 1144 times and has been misclassified 31 times, out of which 9 times it should be 'O', then an entry "you know O || 1144 31 9" will be added to the dictionary. If the original sentence contains "you know" and it is not labeled as O, a new hypothesis will be generated by labeling it as 'O'.

Repetition: Whenever the i^{th} word and the j^{th} word ($j > i$) are the same, all words from the i^{th} position (inclusive) to the j^{th} position (exclusive) are labeled as edit words. For example, in "I want to be able to um I just want it more for multi-tasking", three hypotheses are produced. The first hypothesis is produced by labeling every word in "I want to be able to um" as edit words since 'I' is repeated. The second hypothesis is produced by labeling every word in "want to be able to um I just" as edit words since 'want' is repeated. The third hypothesis is produced by labeling "to be able" as edit words since 'to' is repeated. The window size within which we search for repetitions is set to 12 (i.e., $j - i \leq 12$), since the longest edit region (due to repetition) in the development set is of that size. We introduce this hypothesis producer because the baseline system tends to miss long edit regions, especially when very few words in the region are repeated. However, sometimes a speaker does change what he intends to say by aborting a sentence so that only the beginning few words are repeated, as in the above sentence.

Filler-word-marker: We trained an M3N model for filler word detection. Multiple passes of filler word detection on cleaned-up sentences can sometimes detect filler words that are missed in earlier passes. This hypothesis producer runs before every iteration starts. It performs filler word prediction and modifies the feature table by setting the filler-indicator feature to true so that subsequent operations see the updated feature. However, it does not remove filler words during the clean up process because some words are defined as both filler word and edit word simultaneously.

Edit-word-marker: We run our baseline M3N (the second stage) on the cleaned-up sentence and obtain the N-best hypotheses, i.e., the top N hypotheses $\mathbf{h}$ with $max\{f_{M3N}(\bar{\mathbf{h}}, \bar{\mathbf{w}})\}$. This producer essentially performs multiple passes of disfluency detection.

4.4 Hypothesis Evaluators

Our decoder uses the following hypothesis evaluators to select the best hypothesis:

Fluent language model score: This is the normalized language model score of the cleaned-up sentence, i.e., $f_{fluentLM}(\bar{\mathbf{w}})$. A 4-gram language model is trained on the cleaned-up version of the training

texts (both filler words and edit words are removed). This score measures the fluency of the resulting cleaned-up sentence w.r.t. a fluent language model.

Disfluent language model score: This is the normalized language model score of the cleaned-up sentence, i.e., $f_{disfluentLM}(\bar{\mathbf{w}})$. A 4-gram language model is trained on the original training texts which contain disfluencies. This score measures the fluency of the resulting cleaned-up sentence w.r.t. a disfluent language model. These two LM scores provide contrastive measures because if a cleaned up sentence still contains disfluencies, the disfluent LM will be preferred over the fluent LM.

M3N disfluent score: This is the normalized joint log probability score of the current hypothesis $\mathbf{h}$, i.e., $f_{M3N}(\mathbf{h}, \mathbf{w})$. This score measures how much the baseline M3N model favors the disfluency label assignment of the current hypothesis.

M3N fluent score: This is the normalized joint log probability score of labeling the entire cleaned-up sentence as fluent, i.e.,

$$f_{M3N}(\bar{\mathbf{h}}\text{=}\mathbf{O}, \bar{\mathbf{w}}) = \frac{1}{|\bar{\mathbf{h}}|} \sum_{i=1}^{|\bar{\mathbf{h}}|} f_{M3N}(\bar{h}_i\text{='O'}, \bar{\mathbf{w}}) \tag{6}$$

This score measures how much the baseline M3N model favors the cleaned-up sentence of the current hypothesis. It acts as a discriminative LM in measuring the fluency of the cleaned-up sentence. If the cleaned-up sentence contains disfluencies, this evaluator function will tend to give a lower score.

Expert-POS-class c disfluent score: This is the normalized joint log probability score of the current hypothesis $\mathbf{h}$ under the expert M3N model for POS class c dynamically combined with the baseline M3N model, i.e.,

$$f_c(\mathbf{h}, \mathbf{w}) = \frac{1}{|\mathbf{h}|} \sum_{i=1}^{|\mathbf{h}|} g_c(h_i, \mathbf{w}), \qquad g_c(h_i, \mathbf{w}) = \begin{cases} f_{\text{M3N-}c}(h_i, \mathbf{w}) \text{ if POS}(w_i) \in S_c \\ f_{\text{M3N}}(h_i, \mathbf{w}) \text{ if POS}(w_i) \notin S_c \end{cases} \tag{7}$$

Training of the expert M3N models is described in Section 4.6.

4.5 Integrating M3N into the Decoder Framework

In most previous work such as (Liu et al., 2006) and (Qian and Liu, 2013) that performed filler and edit word detection using sequence models, the begin-inside-end-single (BIES) labeling scheme was adopted, i.e., for edit words (E), 4 labels are defined: E_B (beginning of an edit region), E_I (inside an edit region), E_E (end of an edit region), and E_S (single-word edit region). However, since our beam-search decoder needs to change the labels dynamically among filler words, edit words, and fluent words, it will be problematic if the label sequence has to conform to the BIES constraint especially when the posteriors are concerned. Thus, we use the minimal set of labels: E (Edit word), F (Filler word), O (Outside edit and filler region).

For the first-stage filler word detection, only 'F' and 'O' are used. To compensate for degradation in performance, we increase the clique order of features to 3. We found that increasing the clique order has a similar effect as using the BIES labeling scheme. For example, $f(w_i$='so', y_0='E_B', $t)$ means the previous word is not an edit, both the current word and the next word are edit words, i.e., the previous word, the current word, and the next word can be either O-E-E or F-E-E. So in our minimal labeling scheme, this feature will be decomposed into $f(w_i$='so', y_{-1}='O', y_0='E', y_{+1}='E', $t)$ and $f(w_i$='so', y_{-1}='F', y_0='E', y_{+1}='E', $t)$, both having a higher clique order.

Our preliminary experiments show that by increasing the clique order of features while reducing the number of labels (keeping about the same total number of parameters), we can maintain the same performance. However, training takes a longer time.

4.6 POS-Class Specific Expert Models

We trained 6 expert M3N models, each focusing on disfluency prediction of words belonging to the corresponding set of POS tags. The expert M3N models are trained in the same way as the baseline M3N model, except that we increase the loss weights (Eqn. 4) if the word of that node belongs to

the corresponding POS class. That is, M3N-Expert-POS-class-1 is trained to optimize performance on words belonging to POS-class-1. Nonetheless, it can still predict disfluency for words in other POS classes, except that the error rate may be higher because of the way training is biased.

Class	POS tags	Freq. (%)	F1 range
1	RBS POS PDT NNPS HVS PRP$ BES PRP	33.5	92.3 – 100
2	MD EX CC DT VBP WP WRB	32.2	86.0 – 90.8
3	RB IN	16.7	82.8 – 83.8
4	TO VBD WDT RP JJS	5.1	80.0 – 82.1
5	VBZ VB VBN JJ	6.1	69.3 – 78.1
6	VBG NN CD JJR UH NNS NNP XX RBR	3.2	42.1 – 64.2

Table 4: POS classes for expert M3N models and their baseline F1 scores

We split all POS tags into 6 classes, by first sorting all POS tags in descending order of their F1 scores. Next, for POS tags with higher F1 scores, we form larger classes (higher total proportion), and for POS tags with lower F1 scores, we form smaller classes. The POS classes are shown in Table 4. The algorithm dynamically selects posteriors from different M3N models, depending on the POS tag of the current word.

5 Experiments

5.1 Experimental Setup

We tested all the systems on the Switchboard Treebank corpus (LDC99T42), using the same train/develop/test split as previous work (Johnson and Charniak, 2004; Qian and Liu, 2013). We removed all partial words and punctuation symbols to simulate the condition when automatic speech recognition (ASR) output is used. Our training set contains 1.3M words in 174K sentences; our development set contains 86K words in 10K sentences; and our test set contains 66K words in 8K sentences. The original system has high precision but low recall, i.e., the system tends to miss out edit words. The imbalance can be solved by setting a larger penalty for mis-labeling edits as fluent, i.e., 2 instead of 1 for the weighted hamming loss. We used the loss matrix, $v(\widetilde{y}_t, \bar{y}_t)$, in Table 5 to balance precision and recall. We set the biasing factor B_c to 2, for every class c. We also added two pause duration features (pause duration before and after the current word) from the corresponding Switchboard speech corpus (LDC97S62). We trained our acoustic model on the Fisher corpus and used it to perform forced alignment on the Switchboard corpus to obtain the word boundary time information for calculating pause durations. For the *ngram-score* features, we used the small 4-gram language model trained on the training set with filler words and edit words removed. All continuous features are quantized into 10 discrete bins using cumulative binning (Liu et al., 2006). We set $maxIter$ to 4 in Algorithm 1. The regularization parameter C is set to 0.006, obtained by tuning on the development set.

Label	E	F	O
E	0	1	2
F	1	0	2
O	1	1	0

Table 5: Weighted hamming loss, $v(\widetilde{y}_t, \bar{y}_t)$ for M3N for both stages

5.2 Results

We use the standard F1 score as our evaluation metric, the same as (Qian and Liu, 2013). Performance comparison of the baseline model and expert models on subsets belonging to specific POS classes is shown in Table 6. It shows that by assigning larger loss weights to nodes belonging to a specific POS class, we can to various extent boost the performance on words in that POS class. However, doing so

will sacrifice the overall performance on the entire data set especially on POS classes with lower baseline scores (see Table 7). But since we have several expert models, if we combine them, they can complement each other's weakness and give an overall slightly better performance. The result also shows that the gain by training expert models decreases as the baseline performance on that POS class increases. For example, POS class 6 has the poorest baseline performance and the gain is 2.1%. This gain decreases gradually as we move up the table rows because the baseline performance gets better.

POS class	Expert-M3N F1	Baseline-M3N F1
1	92.5	92.2
2	87.1	86.9
3	84.9	84.7
4	85.3	84.1
5	71.8	70.4
6	57.3	55.2

Table 6: Edit detection F1 scores (%) of expert models on all words belonging to that POS class in the test set (expert-M3N column), and baseline model on all words belonging to that POS class in the test set (baseline-M3N column)

System	F1 (%)
Baseline-M3N	84.7
Expert-M3N(1)	84.6
Expert-M3N(2)	84.4
Expert-M3N(3)	84.3
Expert-M3N(4)	84.6
Expert-M3N(5)	84.0
Expert-M3N(6)	83.8

Table 7: Degradation of the overall performance by expert models compared to the baseline model

Table 8 shows the performance comparison of our baseline models and our beam-search decoder. Statistical significance tests show that our best decoder model incorporating all hypothesis evaluators gives a higher F1 score than the 3-label baseline M3N model (statistically significant at $p < 0.001$), and the 3-label baseline M3N model gives a higher F1 score than the M3N system of (Qian and Liu, 2013) (statistically significant at $p = 0.02$). Our three baseline models have about the same total number of parameters (7.6M). The BIES baseline M3N system uses the same feature templates as shown in Table 2 with reduced clique order. The 2-label baseline M3N system uses the same feature templates with the same clique order. Our results also show that joint filler and edit word prediction performs 0.4% better than edit word prediction alone. Direct combination of expert models is done by first running the general model and the expert models on each sentence to obtain all the label sequences (one for each model). Then for every word in the sentence, if its POS belongs to any one of those POS classes, we choose its label from the output of the corresponding expert model; otherwise, we choose its label from the output of the baseline model.

For the decoder, *M3N-disfluent-score* needs to be present in all cases (except when POS experts are present); otherwise, the F1 score is much worse because the entire sequence is not covered (i.e., just looking at the scores from the cleaned-up sentences is not sufficient in deciding how well filler and edit words have been removed). Adding *M3N-fluent-score*, *Fluent-LM*, or *Disfluent-LM* alone with *M3N-disfluent-score* gives about the same improvement; but when combined, higher improvement is achieved.

Similar to (Qian and Liu, 2013), our system does not make use of any external sources of information except for the last two rows in Table 8 where we added pause duration features. We found that adding pause duration features gave a small but consistent improvement in all experiments, about 0.3% absolute gain in F1 score. Our beam-search decoder (multi-threaded implementation) is about 4.5 times slower

System	F1 (%)
M3N system of (Qian and Liu, 2013)	84.1
Our baseline M3N (using BIES for E and F)	84.4
Our baseline M3N (using 2 labels: E,O)	84.3
Our baseline M3N (using 3 labels: E,F,O)	84.7
Direct combination of the 6 POS-expert-models according to each word's POS	85.2
Decoder: M3N-disfluent-score + M3N-fluent-score	85.1
Decoder: M3N-disfluent-score + Fluent-LM	85.2
Decoder: M3N-disfluent-score + Disfluent-LM	85.1
Decoder: M3N-disfluent-score + POS-experts	85.2
Decoder: M3N-disfluent-score + M3N-fluent-score + Fluent-LM + Disfluent-LM	85.6
Decoder: M3N-disfluent-score + M3N-fluent-score + Fluent-LM + Disfluent-LM + POS-experts	85.7
Decoder: M3N-disfluent-score + M3N-fluent-score + Fluent-LM + Disfluent-LM + PauseDur	85.9
Decoder: M3N-disfluent-score + M3N-fluent-score + Fluent-LM + Disfluent-LM + POS-experts + PauseDur	86.1

Table 8: Performance of the beam-search decoder with different combinations of components

than our baseline M3N model (single-threaded). Overall, it took about 0.4 seconds to detect disfluencies in one sentence with our proposed beam-search decoder approach.

To the best of our knowledge, the best published F1 score on the Switchboard Treebank corpus is 84.1% (Qian and Liu, 2013) without the use of external sources of information, and 85.7% (Zwarts and Johnson, 2011) with the use of external sources of information (large language models from additional corpora were used in (Zwarts and Johnson, 2011)). Without the use of external sources of information, our decoder approach achieves an F1 score of 85.7%, significantly higher than the best published F1 score of 84.1% of (Qian and Liu, 2013). Our decoder approach also achieves an F1 score of 86.1% after adding external sources of information (pause duration features), higher than the F1 score of 85.7% of (Zwarts and Johnson, 2011).

5.3 Discussion

We have manually analyzed the improvement of our decoder over the M3N baseline. For example, consider the sentence in Table 9. Both the baseline M3N system and the first pass output of the decoder will give the cleaned-up sentence "*are these do these programs ...*", which is still disfluent and has a relatively lower fluent LM score but a relatively higher disfluent LM score because of the erroneous n-gram "*are these do these*". The decoder makes use of the fluent LM and disfluent LM hypothesis evaluators during the beam search and performs additional passes of cleaning and eventually gives the correct output.

Sentence	*Um*	*and*	*uh*	*are*	*these*	*like*	*uh*	*do*	*these*	*programs*	...
Reference	F	F	F	E	E	E	F	O	O	O	...
M3N baseline	F	F	F	O	O	F	F	O	O	O	...
Decoder	F	F	F	E	E	E	F	O	O	O	...

Table 9: An example showing the effect of measuring the quality of the cleaned-up sentence.

Overall, our proposed decoder framework outperforms existing approaches. It also overcomes the limitations mentioned in Section 4.1. For example, hypothesis evaluators like *fluent language model score* and *M3N fluent score* achieve the purpose of measuring the quality of cleaned-up sentences. Repeatedly applying the *edit-word-marker* hypothesis producer on a sentence achieves the purpose of cleaning up

the sentence in multiple passes. Hypothesis evaluators corresponding to expert models achieve the purpose of combining POS class-specific expert models. All of these components extend the flexibility of the decoder framework in performing disfluency detection.

6 Conclusion

In conclusion, we have proposed a beam-search decoder approach for disfluency detection. Our beam-search decoder performs multiple passes of disfluency detection on cleaned-up sentences. It evaluates the quality of cleaned-up sentences and use it as a feature to rescore hypotheses. It also combines multiple expert models to deal with edit words belonging to a specific POS class. In addition, we also proposed a way (using node-weighted M3N in addition to label-weighted M3N) to train expert models each focusing on minimizing errors on words belonging to a specific POS class. Our experiments show that combining the outputs of the expert models directly according to POS tags can give rise to some improvement. Combining the expert model scores with language model scores in a weighted manner using our beam-search decoder achieves further improvement. To the best of our knowledge, our decoder has achieved the best published edit-word F1 score on the Switchboard Treebank corpus, both with and without using external sources of information.

7 Acknowledgments

This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

References

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *Proc. of NAACL.*

Daniel Dahlmeier and Hwee Tou Ng. 2012. A beam-search decoder for grammatical error correction. In *Proc. of EMNLP-CoNLL.*

Kallirroi Georgila. 2009. Using integer linear programming for detecting speech disfluencies. In *Proc. of NAACL.*

Mark Johnson and Eugene Charniak. 2004. A TAG-based noisy channel model of speech repairs. In *Proc. of ACL.*

Jeremy G. Kahn, Matthew Lease, Eugene Charniak, Mark Johnson, and Mari Ostendorf. 2005. Effective use of prosody in parsing conversational speech. In *Proc. of EMNLP.*

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001 Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. of ICML.*

Yang Liu, Elizabeth Shriberg, Andreas Stolcke, Dustin Hillard, Mari Ostendorf, and Mary Harper. 2006. Enriching speech recognition with automatic detection of sentence boundaries and disfluencies. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5).

Sameer Maskey, Bowen Zhou, and Yuqing Gao. 2006. A phrase-level machine translation approach for disfluency detection using weighted finite state transducers. In *Proc. of INTERSPEECH.*

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. of ACL.*

Xian Qian and Yang Liu. 2013. Disfluency detection using multi-step stacked learning. In *Proc. of NAACL.*

J.A.K. Suykens and J. Vandewalle. 1999 Least squares support vector machine classifiers. *Neural Processing Letters*, 9.

Ben Taskar, Carlos Guestrin, and Daphne Koller. 2003. Max-margin Markov networks. In *Proc. of NIPS.*

Pidong Wang and Hwee Tou Ng. 2013. A beam-search decoder for normalization of social media text with application to machine translation. In *Proc. of NAACL.*

Qi Zhang, Fuliang Weng, and Zhe Feng. 2006. A progressive feature selection algorithm for ultra large feature spaces. In *Proc. of ACL.*

Simon Zwarts and Mark Johnson. 2011. The impact of language models and loss functions on repair disfluency detection. In *Proc. of ACL.*