Disfluency Detection using a Noisy Channel Model and Deep Neural Language Model

Paria Jamshid Lou

This thesis is submitted for the degree of Master of Research (MRes)

Department of Computing

Faculty of Science and Engineering

MACQUARIE
University

April 24, 2017

Declaration

I certify that the work in this thesis entitled "Disfluency Detection using a Noisy Channel Model and Deep Neural Language Model" has not previously been submitted for a degree nor has it been submitted as part of the requirements for a degree to any other university or institution other than Macquarie University. I also certify that the thesis is an original piece of research and it has been written by me. Any help and assistance that I have received in my research work and the preparation of the thesis itself have been appropriately acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

Signed: ....

Date: ....

Contents

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Mark Johnson for his continuous support, patience, motivation and immense knowledge.

Abstract

Although speech recognition technology has improved considerably in recent years, current systems still output simply a sequence of words without any useful information about the location of disfluencies. On the other hand, such information is necessary for improving the readability of speech transcripts. In fact, speech transcripts containing a lot of disfluencies are difficult to understand, so removing disfluent words can make speech transcripts more readable. Moreover, many tasks including dialogue systems input spontaneous speech. Such systems are usually trained on fluent, clean corpora, so inputting disfluent data would decrease their performance.

This thesis aims at introducing a model for automatic disfluency detection in spontaneous speech transcripts called *LSTM Noisy Channel Model*. The model uses a Noisy Channel Model (NCM) to find "rough copies" that are likely to indicate disfluencies and generate n-best candidate disfluency analyses. Then, the underlying fluent sentences of each candidate analysis are scored using a Long Short-Term Memory (LSTM) language model. The LSTM language model scores, along with other features, are used in a reranker to identify the most plausible analysis. We show that using LSTM language model scores as features to rerank the analyses generated by an NCM improves the state-of-the-art in disfluency detection.

Publication

- Paria Jamshid Lou and Mark Johnson. 2017. Disfluency Detection using a Noisy Channel Model and a Deep Neural Language Model. To appear in ACL'17.

List of Figures

List of Tables

List of Abbreviations

ASR	Automatic Speech Recognition
CRF	Conditional Random Field
LM	Language Model
LSTM	Long Short-Term Memory
MaxEnt	Maximum Entropy
NCM	Noisy Channel Model
NLP	Natural Language Processing
POS	Part Of Speech
RNN	Recurrent Neural Network
TAG	Tree Adjoining Grammar

Chapter 1

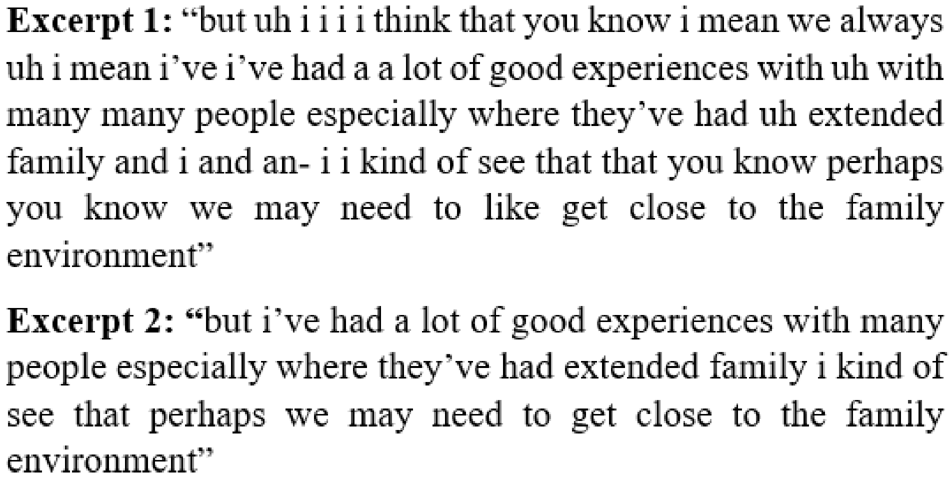
Introduction

Automatic speech recognition (ASR) has made a remarkable improvement in recent years, benefiting from availability of large training data and advances in deep learning (Mohamed et al., 2012; Hinton et al., 2012; Chorowski et al., 2015; Mesnil et al., 2015). In spite of great progress in speech recognition technology, current recognition systems are still far from the ideal goal of generating enriched transcripts, where useful information such as the location of disfluencies is provided in addition to a sequence of words. Disfluencies are an integral part of spontaneous speech, so the output of ASR systems, no matter how perfect and error-less it is, will always contain speech disfluencies.

1.1 Motivation

Disfluency is a characteristic of spontaneous speech that is not present in written text. While disfluency rate varies with the context, age and gender of speaker, Bortfeld et al. (2001) reported disfluencies once in every 17 words. Such frequency is high enough to reduce the readability of speech transcripts. To illustrate the impact of disfluencies on the readability of ASR outputs, consider the following excerpts of spontaneous speech transcripts. The first excerpt in Figure 1.1 shows a human-produced speech transcript containing a number of disfluencies and the second excerpt presents the same transcript with disfluencies removed (Liu et al., 2006). Although the first speech transcript contains no recognition errors, it is difficult to understand mainly due to the presence of speech disfluencies. However, removing disfluencies from the speech transcript makes it more readable and easier to understand even in lack of punctuation and capitalization, as shown in the second excerpt.

Moreover, spoken language is becoming more important as computers become smaller. As an input modality for human-computer interaction, spoken language can eliminate the need for using traditional devices such as mouse and keyboard. However, disfluency as a characteristic of spontaneous speech can negatively influence human-computer interaction. Spontaneous speech input is also necessary for many natural language processing tasks, including dialogue system, machine translation and parsing. Since building, manually transcribing and annotating speech transcripts is prohibitively expensive, these systems are usually trained on large fluent, clean corpora. The mismatch between the training corpora and the actual use case decreases the performance of such systems. Cho et al. (2014) showed that disfluencies in spoken language as input would have a negative impact on machine translation. Therefore, it is crucial to recognize and remove speech disfluencies in order to increase the human readability of speech transcripts, improve human-computer interaction and increase the performance of systems that input spontaneous speech.



Excerpt 1: "but uh i i i i think that you know i mean we always uh i mean i've i've had a a lot of good experiences with uh with many many people especially where they've had uh extended family and i and an- i i kind of see that that you know perhaps you know we may need to like get close to the family environment"

Excerpt 2: "but i've had a lot of good experiences with many people especially where they've had extended family i kind of see that perhaps we may need to get close to the family environment"

Figure 1.1: Excerpt 1 shows a human-produced speech transcript with many disfluencies and Excerpt 2 indicates the same speech transcript with disfluencies removed (Liu et al., 2006)

1.2 Problem Description

Disfluencies are informally defined as interruptions in the normal flow of speech that occur in different forms, including false starts, corrections, repetitions and filled pauses. According to Shriberg's (1994) definition, the basic pattern of speech disfluencies contains three parts: *reparandum*, *interregnum* and *repair*. Example (1.1) illustrates a disfluent structure, where the reparandum *to Boston* is the part of the utterance that is replaced, the interregnum *uh, I mean* is an optional part of a disfluent structure that consists of a filled pause *uh* and a discourse marker *I mean* and the repair *to Denver* replaces the reparandum. The fluent version of Example (1.1) is obtained by deleting reparandum and interregnum words, as shown in Example (1.2).

$$\begin{gathered} \textit{I want a flight } \overbrace{\textit{to Boston,}}^{\text{reparandum}} \\ \underbrace{\textit{uh, I mean}}_{\text{interregnum}} \underbrace{\textit{to Denver}}_{\text{repair}} \textit{ on Friday} \end{gathered} \tag{1.1}$$

$$\textit{I want a flight to Denver on Friday} \tag{1.2}$$

Given a disfluent utterance such as Example (1.1), an effective speech disfluency detection system should be able to identify and remove disfluent words and output a clean version of utterance such as Example (1.2). Although interregnum words are part of a disfluent structure and need to be deleted, the automatic speech disfluency detection task mainly deals with identifying and removing reparandum words. It is because filled pauses and discourse markers belong to a finite set of words and phrases and they are trivial to detect (Johnson and Charniak, 2004). Therefore, the major task of a disfluency detection system is to recognize and delete reparandum words from speech transcripts.

1.3 Method

So far, a wide range of approaches have been proposed to automate the detection and deletion of speech disfluencies. One of the models which has been applied to the speech disfluency detection task is noisy channel model (NCM) (Johnson and Charniak, 2004; Johnson et al., 2004; Zwarts et al., 2010; Zwarts and Johnson, 2011). The main idea behind a noisy channel model of speech disfluency is that repair is usually a “rough copy” of reparandum. In other words, the repair and reparandum are often the same or very similar words in roughly the same order. For example, *to Denver* and *to Boston* in Example (1.1) are both names of city, so they are semantically similar to each other. The NCM uses the similarity between reparandum and repair as a good indicator of disfluency to match the repair to the reparandum.

Although NCM is a strong model for disfluency detection, it is computationally complex to apply an effective language model inside a noisy channel model. The reason is that the NCM uses dynamic programming in order to make the computation tractable. The polynomial-time dynamic programming parsing algorithms of channel model can be used to search for likely repairs if they are used with simple language models such as bigram (Johnson and Charniak, 2004). The bigram language model within the noisy channel model is too simple to capture complicated language structure. On the other hand, many studies show that language modelling is particularly important for identifying disfluent parts of utterance (Stolcke and Shriberg, 1996; Honal and Schultz, 2005; Zwarts and Johnson, 2011). To alleviate this problem in NCM, some researchers use more effective LMs to rescore the NCM disfluency analyses. Johnson and Charniak (2004) applied a syntactic parsing-based LM trained on the fluent version of the Switchboard corpus to rescore the disfluency analyses. Zwarts and Johnson (2011) trained external n-gram LMs on a variety of large speech and non-speech corpora. They used the external LM scores as features into a reranker and improved the baseline NCM.

The idea of applying external language models in the reranking process of the NCM, as well as the state-of-art performance of deep learning techniques in various applications including language modelling (Mikolov et al., 2010; Jozefowicz et al., 2016) motivate us to investigate whether integration of a noisy channel model with a deep neural language modelling improves the state-of-the-art in disfluency detection.

1.4 Objectives

This thesis aims at introducing a new model for detecting disfluencies in spontaneous speech transcripts called the LSTM Noisy Channel Model. The model uses a noisy channel model to generate n-best candidate disfluency analyses, and a long short-term memory (LSTM) language model to score the fluent versions of the NCM analyses. The language model (LM) scores are used as features into a maximum-entropy (MaxEnt) reranker to select the most plausible analysis. The main idea behind using an LSTM LM is that an advanced language model trained on a large clean corpus can model more global properties of sentences. Therefore, an effective language model is expected to assign a high probability to a fluent sentence and a lower probability to a disfluent one.

We also aim at investigating:

- The impact of LSTM LMs: We investigate the effects of using LSTM LMs for scoring the underlying fluent sentences of candidate analyses on the performance of the NCM for detecting disfluencies.

- The difference between LSTM LMs: We compare the performance of different LSTM LMs including forward, backward and bidirectional to find which type of language model is more effective in the disfluency detection task.

- The difference between LSTM and n-gram LMs: We compare the performance of LSTM language models to high-order n-gram models when they are applied to the outputs of the NCM to score the candidate disfluency analyses.

1.5 Contributions

There are three main contributions in this thesis. First, we investigate the use of an advanced language modelling technique, called deep neural LM, to score the candidate disfluency analyses of the NCM. We show that using these LM scores as features into a reranker helps the baseline NCM to select the best disfluency analysis. Second, we explore the use of different forward and backward language models and show that a backward language model is much stronger than a forward one for disfluency detection. Third, we compare the performance of a conventional n-gram model with a deep neural LM on the reranking process of the noisy channel model. We show that capturing longer contexts is important for detecting disfluent parts of speech transcripts and deep neural language model contains all the useful information the n-gram model does.

1.6 Organization

The remainder of this thesis is organized as follows. Chapter 2 gives a theoretical background on speech disfluency phenomenon, reviews automatic detection methods and explains important language modelling techniques. Chapter 3 demonstrates the theoretical framework of the baseline noisy channel model and the proposed model for disfluency detection. Chapter 4 describes the corpora for language modelling, hyper-parameters, model setting, experiments and model performance. Finally, Chapter 5 makes a summary and draws conclusions.

Chapter 2

Literature Review

In this chapter, we review the published research literature on speech disfluency detection. Since the goal of the thesis is to introduce a model for locating the disfluent parts of transcribed speech, we mainly focus on the approaches that make use of lexical information rather than prosodic and acoustic features. Although prosodic cues are beneficial for interruption point detection, the approaches using only lexical features do almost as well as those with both prosodic and lexical features (Ostendorf and Hahn, 2013; Zayats et al., 2014).

This chapter includes four main sections. Section 2.1 reviews some major psycholinguistic studies describing the pattern and different types of disfluencies. The psycholinguistic evidence establishes the theoretical framework for modelling and automating speech disfluency detection and provides a standard annotation scheme for disfluency. Section 2.2 describes the task of automatic disfluency detection and standard approaches to it, including *sequence tagging*, *parsing-based* and *noisy channel models*. Since the noisy channel model forms the baseline system in this thesis, we explain it in more detail. Moreover, the main contribution of the thesis is using deep neural LM probabilities as features into a reranker for selecting the best analysis of the NCM. Therefore, in Section 2.3, we review different language modelling techniques including n-*gram*, *parsing-based* and *recurrent neural network* LMs. Finally, we summarize and conclude this chapter in Section 2.4.

2.1 Psycholinguistics Studies on Speech Disfluencies

Disfluencies had been traditionally viewed as “irregular” phenomena and given little attention until about two decades ago. Shriberg’s (1994) study, however, changed this prevalent viewpoint towards speech disfluencies and influenced the subsequent research trend in this area. Using speaker, acoustic and sentence features, she showed that disfluencies had a regular trend in spontaneous speech. This finding was particularly important for modelling and automating the task of disfluency detection. In fact, she provided a basic pattern of speech disfluencies which established the standard theoretical framework for data annotation and automatic detection task.

According to Shriberg (1994), the standard pattern for disfluency includes the *reparandum* (word or words that the speaker intends to be replaced or ignored), the *interruption point* (+) marking the end of the reparandum, the associated *repair* and an optional *interregnum* after the interruption point (e.g. filled pauses, discourse markers and so on) as shown below:

[reparandum + { interregnum } repair]

Example (2.1) illustrates a disfluency annotated sentence using Shriberg's (1994) standard pattern.

I want a flight [*to Boston* + {*uh I mean*} *to Denver*] *on Friday.* (2.1)

Ignoring the interregnum, disfluencies can be categorized into three types: *restart*, *repetition* and *correction*, based on whether the repair is empty, the same as the reparandum or different, respectively. *Repetitions* are the most common type of disfluencies. Example (2.2) shows a repetition disfluency in which the phrase *I don't* is repeated twice. Example (2.3) illustrates a correction disfluency where the reparandum *good* is replaced by the repair *brilliant*. Example (2.4) indicates a restart disfluency where the speaker abandons a sentence and starts a new one saying *Do you have* . . . . Different types of disfluencies demonstrate speaker variability; for example, some speakers are "deleters" while others are "repeaters" (Shriberg, 1994).

[*I don't* + *I don't*] *want to go to work today.* (2.2)

I think it is a [*good* + {*well*} *brilliant*] *idea.* (2.3)

[*I need to* + {*uh*}] *Do you have any time?* (2.4)

While these are examples of simple disfluency, it is possible to have nested disfluencies where one disfluent structure is entirely contained within another one. Examples (2.5) and (2.6), extracted from Switchboard corpus, show complex disfluent structures, where there are multiple or nested disfluencies. In Example (2.5), both inner and outer pairs of square brackets indicate *correction* disfluencies. The outer set of square brackets in Example (2.6) refer to *correction* disfluency and the inner square brackets demonstrate *repetition* type of disfluency.

[[*it's just* + {*oh*} *I just*] + *I*] *love it.* (2.5)

[[*it* + *it*] + *that's*] *a possibility.* (2.6)

In a disfluent structure, the reparandum is an important component to study because it usually contains important information about the subsequent words in the utterance. Reparandum words are not part of the fluent version of utterance, but they can convey contextual information to help listener anticipate the repair (Lowder and Ferreira, 2016). For example, if speaker says *"Turn right uh I* *mean ...*", the listener can guess the word *left* before hearing the repair using the word *right*. Lau and Ferreira (2005) tested the effect of reparandum words on the processing of garden path[1] sentences which contained main verb/reduced relative ambiguity. They showed that the effect of reparandum would remain even after it was replaced by repair and influence the interpretation of sentence. In Examples (2.7) and (2.8), the repair *selected* is ambiguous because it can be interpreted either as a main verb or as a reduced-relative verb. According to their experiment, listener understands (or parses) Example (2.7) more easily because the reparandum *chosen*, unlike *picked*, is unambiguously a reduced-relative verb. In fact, the passive structure activated by the reparandum i.e. *chosen* in Example (2.7) lingers even after it is replaced by a new verb i.e. *selected*, so the parser uses this information to resolve the ambiguity.

$$\textit{The boy}\left[\textit{ chosen + }\left\{\textit{uh}\right\}\textit{ selected }\right]\textit{for the role celebrated with his family.} \tag{2.7}$$

$$\textit{The boy}\left[\textit{ picked + }\left\{\textit{uh}\right\}\textit{ selected }\right]\textit{for the role celebrated with his family.} \tag{2.8}$$

2.2 Automatic Speech Disfluency Detection

The task of disfluency detection is sometimes called *edit detection*[2]. By automatic edit detection, we usually mean identifying and deleting reparandum words. Since filled pauses and discourse markers belong to a closed set of words, they are trivial to detect (Johnson and Charniak, 2004). Johnson et al. (2004) noted that 4 words *uh*, *um*, *eh* and *ah* comprised 95% of the filled pauses in Switchboard corpus, and 95% of the discourse markers was composed of 2 phrases *you know* and *i mean* and 6 words *like*, *so*, *well*, *oh*, *actually* and *now*. Thus, the challenge in the disfluency detection task is to find and remove reparandum words because filled pauses and discourse markers can be detected using a set of deterministic rules.

In the past two decades, different approaches have been proposed for automatic edit detection, which can be categorized into three main groups: *sequence tagging*, *parsing-based* and *noisy channel models*. The sequence tagging models label each word of an utterance as fluent or disfluent using a variety of different techniques, including hidden Markov model (Liu et al., 2006; Schuler et al., 2010), conditional random field (CRF) (Ostendorf and Hahn, 2013; Zayats et al., 2014) and recurrent neural network (RNN) (Hough and Schlangen, 2015; Zayats et al., 2016). Although sequence tagging models can be easily generalized to a wide range of domains, they require a specific state space for disfluency detection, such as begin-inside-outside (BIO) style states that label words as being inside or outside of a reparandum word sequence. This means that it is usually required to modify the

---

[1] Garden-path refers to grammatically correct sentences that are parsed by the listeners in such a way that their first interpretation will be incorrect. For instance, the first three words in "The old man the boat" lead listeners to parse "The old man" as a noun phrase. However, when they encounter another "the" following the supposed noun man, they are forced to analyse the sentence again. The correct parse of the sentence is [S [NP The old] [VP [V man] [NP the boat]]].

[2] Edit is another name for reparandum.

annotation scheme of the data for training sequence tagging models. The parsing-based approaches refer to parsers that detect disfluencies and identify the syntactic structure of the sentence. Although joint parsing and disfluency detection leads to the optimization of both models, training them requires large annotated tree-banks that contain both disfluencies and syntactic structures. Noisy channel models use the similarity between reparandum and repair as an indicator of disfluency. However, applying an effective language model inside an NCM is computationally complex. In the following subsections, we review previous work on automatic disfluency detection. Since we use an NCM as our baseline model, we describe it in more detail.

2.2.1 Sequence Tagging Models

The task of assigning a single label to each word in a sequence is referred to as a sequence tagging problem. The standard format of sequence tagging in disfluency detection includes 3 states to specify the *begin*, *inside* and *outside* (*BIO*) of edit region. As shown in Example (2.9), *BE*, *IE* and *O* denote beginning, inside and outside of edit region.

$$\begin{array}{ccccccc} \text{it} & \text{is} & \big[\,\text{my} & \text{idea} & +\big\{\,\text{um}\,\big\} & \text{our} & \text{idea}\,\big] \\ \text{O} & \text{O} & \text{BE} & \text{IE} & \text{O} & \text{O} & \text{O} \end{array} \tag{2.9}$$

Some researchers (Georgila, 2009; Ostendorf and Hahn, 2013; Zayats et al., 2016) use a finer grained set of labels by extending the standard 3-state space to specify the word proceeding the interruption point *IP* and a single word edit *BE_IP*, as shown in Examples (2.10) and (2.11).

$$\begin{array}{cccccccc} \text{a} & \text{flight} & \big[\,\text{to} & \text{Boston} & \text{to}\ + & \big\{\,\text{uh}\,\big\} & \text{to} & \text{Denver}\,\big] \\ \text{O} & \text{O} & \text{BE} & \text{IE} & \text{IP} & \text{O} & \text{O} & \text{O} \end{array} \tag{2.10}$$

$$\begin{array}{cccccccc} \text{I} & \text{should} & \text{talk} & \text{to} & \big[\,\text{him}\ + & \big\{\,\text{I} & \text{mean}\,\big\} & \text{her}\,\big] \\ \text{O} & \text{O} & \text{O} & \text{O} & \text{BE\_IP} & \text{O} & \text{O} & \text{O} \end{array} \tag{2.11}$$

The standard *BIO* encoding as well as the extended 5-state model do not provide any information about the type of disfluency. For extracting pattern matching lexical features, however, it is necessary to have an explicit annotation of different disfluency types. Some work has been done to modify the 5-state annotation scheme to specify the type of disfluency. Ostendorf and Hahn (2013) proposed an alternative annotation framework to explicitly model repetition disfluencies. Instead of using a nested annotation scheme, they introduced a flat annotation for multiple repetitions in a row using a single bracket. The advantage of the flattened structure was that it would clearly specify the connection between each instance of the repetition. This was particularly useful for detecting long repetition disfluencies such as stutter-like ones. They also extended the 5-state model to a 9-state one to model repetitions versus other disfluencies. For labelling each word in a sentence, they used a CRF model with 140 feature types including word class features such as part-of-speech (POS) tags and n-gram language models, as well as pattern match ones to capture repeated words and POS tags in a window of words. They compared their model with the baseline 5-state CRF. Using the explicit modelling of stutter-like repetition disfluencies, they improved the general disfluency detection task.

Zayats et al. (2014) augmented Ostendorf and Hahn's (2013) explicit repetition model to detect correction disfluencies, too. A 16-state model, called explicit correction model, was considered as an expansion to the 9-state repetition space. They used Ostendorf and Hahn's CRF model as the baseline and apart from their feature sets, they studied the effect of new features including trigram language models, pattern-match features and coarse grained POS tags. The pattern match features showed the distance from the current word to the nearest word that matched one of the patterns in the baseline. When there existed a match, the distance pattern match feature was an integer between 1 and l, and when there was no match, it was $l + 2$, for a window of length l. Using the new features, especially distance-based pattern match ones, they achieved large improvements in correction detection.

Applying a sequence tagging technique to label each word as being inside or outside edit region sometimes results in invalid label sequences. For example, the model may output an illegal sequence of labels such as **O IE IE IP*, where disfluency is started with *IE* (i.e. inside edit region) tag instead of *BE* (i.e. beginning of edit region). In order to avoid such inconsistencies between neighbouring labels, Georgila (2009) presented a post-processing method based on Integer Linear Programming (ILP) to incorporate local and global constraints. She trained different classifiers including hidden Markov models to label the sequences of words. Each classifier would assign a probability to each of 5 tags. In order to select the best sequence, she used ILP technique. The aim of ILP is to optimize a linear objective function subject to some constraints regarding the disfluency tags. For instance, in a sequence of words, the last word is either *BE-IP* (i.e. single word edit), *IP* (i.e. word proceeding the interruption point) or *O* (i.e. outside edit region) and it cannot be *BE* because then we would expect to see an *IP*. She showed that by applying ILP at the test time, the performance of the classifiers would improve.

Some recent studies have applied recurrent neural networks to disfluency detection. Hough and Schlangen (2015) treated incremental disfluency detection as a word-by-word tagging task where the system would output the predicted tag for the current word as it was going through the utterance word-by-word. Although their model led to good incremental properties with low latency and good output stability, they reported weak performance in comparison to other studies on disfluency detection due to the latency constraints. Moreover, they did not model the transition between tags which was important for identifying multi-word disfluencies.

Word embeddings learned by an RNN have also been used as features in a CRF classifier. Cho et al. (2013) introduced a CRF-based speech disfluency detection system developed on German to improve spoken language translation performance. Apart from word embedding, they considered pattern matching, 4-gram language models trained on words and POS tags and phrase-level information. The phrase-level information was used to detect semantic similarity of words or phrases in a source sentence independently from their syntactic roles. For example, the German word *jetzt* (meaning *now*) is annotated as a disfluency, followed by a word *inzwischen* (meaning *meantime*, *now*). Translating the source sentence as it is generates the translation containing two identical tokens in a row in English. They solved this problem by examining the meaning of the source words in a phrase table. They improved BLEU score[3] by 2 points using the CRF model. They showed that word embeddings and phrase-level information are particularly useful for detecting semantically related disfluencies.

One recent paper has used recurrent neural networks as a sequence tagging model. Zayats et al. (2016) introduced a bidirectional long short-term memory neural network as a classifier to detect repetition and correction disfluencies. Apart from a 5-state model, they considered two extensions: a 8- and a 17-state model to capture repetition and correction, respectively. In order to train their model, they used word index, POS tags and disfluency-based features capturing partial words, distance between reparandum and repair words and type of interregnum as their input vectors. They also trained word and POS embeddings within their model and initialized them using a backward-LSTM language model trained on Switchboard corpus. Linear Integer Processing (Georgila, 2009) was also used as a post-processing step to smooth bidirectional LSTM predictions. They reported state-of-the-art results for both repetition and correction detection. The best result was reported for 8-state model plus ILP.

2.2.2 Parsing-based Models

In contrast to most papers which focus solely on either disfluency detection or parsing, a few attempts have been made to jointly parse sentences with disfluencies. The parsing-based approaches refer to parsers that detect disfluencies, as well as identifying the syntactic structure of the sentence. Rasooli and Tetreault (2013) developed a joint dependency parsing and disfluency detection system based on transition-based parsing models. In transition-based models, a tree is converted to a set of incremental actions and the parser decides to commit an action depending on the current configuration of the partial tree. It is possible to augment the set of actions to extend the functionality of the parser on disfluency detection. They used a bottom-up strategy, called *arc-eager algorithm* for dependency parsing. The standard arc-eager algorithm contains four actions to determine the head, pop out or shift the words between stack and buffer. In order to make it suitable for the disfluency detection task, they added three more actions. The new extended actions were responsible for locating and removing from sentence the reparandums, discourse markers and filled pauses and deleting their dependencies. They applied two classifiers to select the best action for the current configuration of sentence. The first classifier was responsible for deciding between applying the main or the extended set of actions, and the second classifier was used to predict the best parsing transition using arc-eager parsing algorithm. In this method, before getting any right dependents, words would receive a head from their left side. Due to similarity of reparandum and repair in most cases, the reparandum might first get a head but then it would be removed when the parser faced the repair. Hence, the advantage of the model was

---

[3] BLEU score is always a number between 0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts.

that it would use the similarity between the reparandum and repair, delete reparandum words from sentence and clear their dependencies. Using the model, they achieved a very high accuracy for parsing, but their model did not perform as well as the state-of-the-art in disfluency detection.

The arc-eager algorithm has a monotonic behaviour. It means when an action is performed, subsequent actions should be consistent with it. In monotonic parsing, if a word becomes a dependent of another word or obtains a dependent, other actions should not change those dependencies constructed for that word in the action history. This monotonic behaviour is problematic for deleting disfluencies because if an action builds a dependency relation, the other actions cannot modify that dependency relation. As a solution, Honnibal and Johnson (2014) extended the arc-eager algorithm with a new non-monotonic transition, called *Edit*, to incrementally parse sentences without knowing early that a word was disfluent. The *Edit* transition marks the word i on top of the stack and its rightward descendents as disfluent. The stack is then popped and its leftward children and all dependencies to and from words marked disfluent are deleted. Using this model, they achieved state-of-the-art accuracy for both speech parsing and disfluency detection.

So far, all mentioned papers were based on the unrealistic assumption that input texts were transcribed by human annotators. In real-world applications, however, the input is usually the output of an automatic speech recognition system, so it contains both disfluent words and recognition errors from ASR system. Yoshikawa et al. (2016) proposed a parsing method to handle both disfluencies and ASR errors using arc-eager algorithm. To do so, they added three new actions i.e. *Edit*, *LeftArcError* and *RightArcError* to the original arc-eager algorithm. The *Edit* action was responsible for removing a disfluent token when it was the first element of the stack. The *Edit* action, unlike the action introduced by Honnibal and Johnson (2014), would remove disfluent tokens one-by-one, so the length of the action sequence would be always $2n - 1$ (n is the number of tokens in each sequence) and the parser could use standard beam search without normalization. They applied *LeftArcError* and *RightArcError* to deal with ASR error tokens while the original *LeftArc* and *RightArc* were to handle non-error tokens. The advantage of using two different sets of action for error and non-error tokens was that the weights could not be shared between them. Along with Honnibal and Johnson's (2014) feature set, they used additional features extracted from a word confusion network (WCN) generated by ASR models to find the erroneous regions of the text. Word confusion networks (WCN) represent a special case of general word lattices, with a more compact, regular structure. A confusion network contains a sequence of slots, with each slot containing one or more word arcs. Selecting a path through the confusion network therefore reduces to selecting which arc in each slot is being used. They hypothesized that the WCN slots with more arcs tended to correspond to erroneous region. Hence, they calculated mean and standard deviation of arc posteriors and the highest arc posterior in each WCN slot corresponding to each word token. Since the error tokens had a relatively low frequency, the weights for *LeftArcError* and *RightArcError* actions might update too infrequently. To avoid this problem, they used backoff action feature to have an accurate generalization of weights. Using novel features suited to ASR output texts as well as new actions, they improved the baseline for both disfluency detection and parsing. The WCN feature increased disfluency detection accuracy.

2.2.3 Noisy Channel Models

In the noisy channel model of speech disfluency, it is assumed that there is a well-formed source utterance X to which some noise is added, generating a disfluent utterance Y as follows:

X = a flight to Denver
Y = a flight to Boston uh I mean to Denver

Given Y, the goal of the NCM is to find the most likely source sentence $\hat{X}$ such that:

$$\hat{X} = \arg\max_X P(X|Y) = \arg\max_X P(Y|X)P(X) \tag{2.1}$$

As shown above, Bayes rule is applied to decompose the probability $P(X|Y)$ into two probabilities $P(Y|X)$ and $P(X)$. We assume that X is a substring of Y, so the source sentence X is obtained by deleting words from Y. For each sentence Y, there are only a finite number of potential source sentences. However, with the increase in the length of Y, the number of possible source sentences X grows exponentially, so it is not feasible to do exhaustive search.

According to Equation (2.1), the noisy channel model involves two components. A *channel model* defines a conditional probability $P(Y|X)$ of sentence Y which may contain disfluencies, given source sentences. A *language model* defines a probability distribution $P(X)$ over the source sentences X, which do not contain disfluencies. Using the channel model and language model probabilities, the noisy channel model outputs n-best candidate disfluency analyses as follows:

1. *~~a flight~~ to Boston uh I mean to Denver*
2. *a flight ~~to Boston~~ uh I mean to Denver*
3. *a flight to Boston uh I mean ~~to Denver~~*
4. *a ~~flight~~ to Boston uh I mean to ~~Denver~~*
5. *a flight to ~~Boston~~ uh I mean to Denver*

⋮

where the potential disfluent words in each analysis are indicated by strikethrough text. Another example is shown below where there is a long-range repair in the sentence:

1. *she said she 'll never put her child ~~in a in a in a in a~~ in a preschool*
2. *she said she 'll never put her child ~~in a in a~~ in a in a in a preschool*
3. *she said she 'll never put her child in a in a ~~in a in a~~ in a preschool*

⋮

Johnson and Charniak's (2004) used a noisy channel model[4] to identify and remove disfluent words of speech transcripts. In order to estimate the language model $P(X)$ component of NCM,

---

[4] Since the thesis uses this noisy channel model to generate n-best candidate disfluency analyses, we explain it in more detail in Chapter 3.

they used a bigram language model trained on Switchboard corpus. They defined the channel model $P(Y|X)$ over the aligned reparandum/repair strings. To estimate this distribution and obtain the disfluent versions of the source sentence, they applied a Tree Adjoining Grammar (TAG) based transducer. TAG is a mildly context sensitive grammar which is responsible for matching words from reparandum to repair. The main idea behind the NCM is that the reparandum and repair are closely related to each other. The advantage of TAG over context-free grammars is that it is powerful enough to capture the crossing dependencies between reparandum and repair. They trained the NCM on Switchboard corpus. Switchboard is a disfluency annotated corpus which includes explicitly marked repair strings for each speech disfluency. Therefore, it can be used for training a disfluency model. Using the NCM probabilities derived from training data, they produced n-best candidates of the disfluencies for each sentence in testing time. Then, in order to select the most probable candidate, they calculated the probability of the fluent part of the sentence using a language model. They tried three different language models to score the outputs of the NCM: *bigram*, *trigram* and *parser-based* language model. They demonstrated that using a parser-based language model significantly improved f-score in comparison with bigram and trigram. The results indicate that this probabilistic model is successful for identifying long repairs where reparandum and repair are similar to each other.

Johnson et al. (2004) applied a maximum entropy reranker to the outputs of the baseline NCM (Johnson and Charniak, 2004) in order to select the most plausible disfluency analysis. The advantage of using a maximum entropy reranker is that it allows for incorporation of a wide range of different features. As features to their reranker, they used the log probabilities produced by TAG channel model, parsing-based language model scores, as well as different variables indicating partial words, POS tags, identical words and so on. They evaluated the performance of their model both on manually transcribed speech and output of ASR. They showed that extending the baseline noisy channel model with a MaxEnt reranker would improve the performance of disfluency detection. In the following part, we describe Johnson et al.'s (2004) MaxEnt reranker[5] in more detail.

In the MaxEnt reranker, it is assumed that there is a training data T which contains information about n possibly disfluent sentences. For the ith sentence, T includes the sequence of words x_i, a set Y_i of 25-best candidate disfluent analyses generated by the noisy channel model and the correct "edited" labelling candidate $y_i^* \in Y_i$. There is a vector $f = (f_1, ..., f_m)$ of feature functions, where each f_j maps a word sequence x and an "edit" labelling y for x to a real value $f_j(x, y)$. Abusing notation somewhat, we write $f(x, y) = (f_1(x, y), ..., f_m(x, y))$. A vector $w = (w_1, ..., w_m)$ of feature weights defines a conditional probability distribution over a candidate set Y of "edited" labellings for a string x as follows:

$$P_w(y|x, Y) = \frac{exp(w \cdot f(x, y))}{\sum_{y' \in Y} exp(w \cdot f(x, y'))} \tag{2.2}$$

The feature weights w are estimated from the training data T by finding a feature weight vector $\hat{w}$ that optimizes a regularised objective function:

---

[5] This thesis applies this MaxEnt reranker to select the most plausible disfluency analysis of the NCM.

$$\hat{w} = \arg\min_{w} L_T(w) + \alpha \sum_{j=1}^{m} w_j^2 \tag{2.3}$$

In Equation (2.3), α is the regulariser weight and L_T is a loss function.

Zwarts and Johnson (2011) investigated the use of two different loss functions in Johnson et al.'s (2004) MaxEnt reranker. They showed that optimising f-score rather than log loss improves disfluency detection performance. According to Zwarts and Johnson (2011), using an asymmetric loss function such as *approximate expected f-score*, shown in Equation (2.4), improves the performance better than a symmetric loss function such as *log loss*. It is because of the skewness of the data in which "edited" words are very infrequent. In such case, a symmetric loss function equally weights each mistake, while an effective loss function is expected to weight "edited" words more highly. In Equation (2.4), g is the number of "edited" words in the gold test data, $E_w[e]$ is the expected number of "edited" words proposed by the system and $E_w[c]$ is the expected number of correct "edited" words generated by the system.

$$FLoss_T(w) = 1 - \frac{2E_w[c]}{g + E_w[e]} \tag{2.4}$$

This approximation, inspired by the evaluation metric f-score, and its derivatives with respect to w are easy to calculate. For example, the expected number of correct edited words is:

$$E_w[c] = \sum_{i=1}^{n} E_w[c_{y_i^*}|Y_i] \tag{2.5}$$

$$E_w[c_{y_i^*}|Y_i] = \sum_{y \in Y_i} c_{y_i^*}(y) P_w(y|x_i, Y_i) \tag{2.6}$$

and $c_{y^*}(y)$ is the number of correct "edited" labels in y given the gold labelling y^*. The derivatives of $FLoss$ are:

$$\frac{\partial FLoss_T}{\partial w_j}(w) = \frac{1}{g + E_w[e]} \left(FLoss_T(w) \frac{\partial E_w[e]}{\partial w_j} - 2\frac{\partial E_w[c]}{\partial w_j} \right) \tag{2.7}$$

where:

$$\frac{\partial E_w[c]}{\partial w_j} = \sum_{i=1}^{n} \frac{\partial E_w[c_{y_i^*}|x_i, Y_i]}{\partial w_j} \tag{2.8}$$

and

$$\frac{\partial E_w[c_{y^*}|x, Y]}{\partial w_j} = E_w[f_j c_{y^*}|x, Y] - E_w[f_j|x, Y] E_w[c_y^*|x, Y] \tag{2.9}$$

$\partial E[e]/\partial w_j$ is obtained by a similar formula.

Apart from applying the approximate expected f-score loss function in n-best reranking, Zwarts and Johnson (2011) showed that using external language model scores as features into the MaxEnt reranker improved the baseline noisy channel model (Johnson and Charniak, 2004). The NCM uses a simple bigram LM and it is computationally complex to use more effective LMs inside the NCM. Thus, as a solution, they applied higher order n-gram models, particularly 4-gram, on the outputs of the NCM to rescore the candidate disfluency analyses. The additional 4-gram language models were trained on large speech and non-speech corpora including Switchboard, Web IT 5-gram, Gigaword and Fisher. In addition to external language model scores, they used boolean indicator features capturing identical words in their reranker[6]. Their work showed that training a language model on large corpora would improve the baseline noisy channel model system. They also showed that using a language model trained on large non-speech corpus would improve the reranking process better than the one trained on a modest-size speech data.

2.3 Language Modelling

Since the proposed model of speech disfluency applies a deep neural network LM on the outputs of the NCM, we provide a background on different language modelling techniques in this section.

Language modelling techniques were originally developed for the problem of speech recognition, but they still play a major role not only in modern ASR systems (Mikolov et al., 2010; Arisoy et al., 2012) but also in a wide variety of natural language processing (NLP) applications such as machine translation (Schwenk et al., 2012), text summarization (Rush et al., 2015) and disfluency detection (Zwarts and Johnson, 2011). A language model is responsible for assigning a probability to sequences of words; therefore, it is useful for identifying words in noisy, ambiguous inputs. Such a model, for instance, can predict that the sequence “I have an amount in my mind” is more likely to appear in a fluent sentence than is the sequence “I I have a an amount in my mind”. This section gives a review to three techniques of language modelling including *n-gram*, *parsing-based* and *recurrent neural networks*, but before that we need to give a formal definition of language model.

A language model is formally defined as follows. If V is the vocabulary of the language, a sentence in the language is a sequence of words $x_1x_2...x_n$, where $n \geqslant 1$, we have $x_i \in V$ for $i \in \{1, ..., n\}$. We define $V^\star$ to be the set of all sentences with the vocabulary V. A language model consists of a set $V^\star$ and a function $P(x_1, x_2, ..., x_n)$ such that:

- For any $(x_1...x_n) \in V^\star$, $P(x_1, x_2, ..., x_n) \geqslant 0$
- Moreover, $\sum\limits_{(x_1...x_n) \in V^\star} P(x_1, x_2, ..., x_n) = 1$

The function P is a probability distribution over the sentences in $V^\star$. It can be estimated by different techniques including *n-gram*, *parsing-based* and *recurrent neural networks*.

---

[6] These features are explained in detail in Section 3.4.

2.3.1 N-gram Language Models

A probabilistic language model estimates either the probability of a word x given a history h or the probability of an entire word sequence X. In order to estimate the probability of a word sequence $x_1...x_n$, we may use the chain rule of probability to decompose $p(x_1, ..., x_n)$ as follows:

$$\begin{aligned} P(x_1, ..., x_n) &= P(x_1)P(x_2|x_1)P(x_3|x_1^2), ..., P(x_n|x_1^{n-1}) \\ &= \prod_{k=1}^{n} P(x_k|x_1^{k-1}) \end{aligned} \tag{2.10}$$

where $P(x_n|x_1^{n-1})$ is the conditional probability of word x_n given the preceding sequence $x_1, ..., x_{n-1}$.

In theory, the joint probability of an entire sequence of words is estimated by multiplying together a number of conditional probabilities as shown in Equation (2.10). In practice, however, we cannot compute the exact probability $P(x_n|x_1^{n-1})$ due to data sparsity. In fact, it is not possible to directly estimate the probability by counting the number of times every word occurs following every long sequence of previous words, even if we use a very large corpus. The reason is that language is creative and there is always the possibility of seeing new contexts.

The intuition of the n-gram model is that instead of calculating the probability of a word given its entire history, we can consider just the last several words to approximate the history (Jurafsky and Martin, 2000). For instance, a bigram model looks one word into the past. We can generalize the bigram, to the trigram (which considers two words in the past) and thus to the n-gram which looks $n - 1$ words into the past. The assumption that the probability of a word depends only on last preceding words is called a *Markov* assumption. The intuition of the Markov assumption is that we can predict the probability of future strings without looking too far in the past.

We can estimate the n-gram probabilities by applying a maximum likelihood estimation (MLE) to get counts from a corpus and normalize the counts so that they lie between 0 and 1 as follows:

$$P(x_n|x_{n-N+1}^{n-1}) = \frac{C(x_{n-N+1}^{n-1}x_n)}{C(x_{n-N+1}^{n-1})} \tag{2.11}$$

A 4-gram probability for sentence “Put the fruit tray on the table” can be computed using Equation (2.11) as follows:

$$P(table|tray\ on\ the) = \frac{C(tray\ on\ the\ table)}{C(on\ the\ table)} \tag{2.12}$$

Instead of computing the probability *P(table|put the fruit tray on the)*, the 4-gram model approximates it with the probability *P(table|tray on the)* using the Markov assumption.

We obtain the n-gram counts from a training data; however, any corpus is limited and it is always possible to have unseen word sequences in test time. In this case, MLE assigns zero probabilities to perfectly acceptable word sequences which have not been observed in the training data. To overcome this problem, we can discount the counts for frequent n-grams and distribute some probability mass to the unseen n-grams. The idea of reassigning some probability mass to unseen events is called *smoothing* or *discounting*.

One of the most commonly used and best performing n-gram smoothing methods is *Kneser-Ney* smoothing (Kneser and Ney, 1995) which evolved from absolute discounting. The idea of absolute discounting is to subtract a constant value d from all non-zero n-gram counts and redistribute it proportionally based on the observed data. The intuition of absolute discounting is that subtracting a small discount d from the very high counts will not affect them much. It will mainly modify the low count n-grams, for which we do not have reliable estimation anyway. The absolute discounting probability of a bigram, for instance, is estimated by subtracting a constant value from the bigram count and interpolating it with a weighted unigram probability. The problem of unigram model is that it does not distinguish words that are very frequent but only occur in a restricted set of contexts (e.g. *Angeles* which almost only occurs after *Los*) from words which are less frequent but have a much wider distribution (e.g. *glasses*). *Angeles* is more common since *Los Angeles* is a very frequent word, so a unigram model in the absolute discounting assigns *Angeles* a higher probability than *glasses*.

The Kneser-Ney smoothing uses the concept of absolute discounting interpolation to incorporate information from higher and lower order n-gram language models, but it uses a more complicated way of handling the lower-order unigram distribution. This technique estimates the probability of seeing the word x as a novel continuation in a new unseen context. In other words, it considers the number of bigram types the word x completes, rather than estimating its unigram probability which indicates how likely the word x is. In fact, the Kneser-Ney smoothing is based on the hypothesis that words which have appeared in more contexts in the past are more likely to appear in new contexts, too. As a result, the frequent word *Angeles* occurring in only one context *Los* will have a low probability. Further details of the Kneser-Ney smoothing can be found in Kneser and Ney (1995) and Jurafsky and Martin (2000).

2.3.2 Parsing-based Language Models

The intuition of parsing-based language models is to use the syntactic structure of the sentence to estimate its probability. In fact, we can think of a statistical parser as a generative model of language. A parser aims at finding a parse tree π for a sequence of words (or sentence) s. We can compute $P(\pi, s)$ by defining a language model that assigns a probability to all possible sentences in the language by computing the sum in Equation (2.13).

$$P(s) = \sum_{\pi} P(\pi, s) \tag{2.13}$$

where $P(\pi, s)$ is zero if $\pi \neq s$[7].

So far, different parsing as well as language modelling techniques have been used to develop parsing-based language models. However, we describe Charniak's (2001) model because it has been used in the literature to score the analyses of the noisy channel model (Johnson and Char-

---

[7] $\pi \neq s$ occurs when π is an incorrect parse tree of sentence s.

niak, 2004). Charniak (2001) proposed a language model using lexicalized probabilistic context-free grammar (PCFG) models. The parsing model called *immediate-head parser* conditioned all events of the immediate descendants of a constituent c on the lexical head of c. Consider the parse tree of the sentence "put the fruit tray on the table" as shown in Figure 2.1. The probability that the vp expands to $v\ np\ pp$ is conditioned on the head of the vp i.e. *put*, as it is the case for sub-heads under the vp i.e. *tray* as the head of np and *on* as the head of pp.

The parser assigns a probability to the parse π using a top-down process of considering each constituent c in π. For each c, the pre-terminal of c, its tag i.e. $t(c)$ and the lexical head of c i.e. $h(c)$ is determined. Then, the expansion of c into further constituents $e(c)$ is considered. The probability of a parse is computed as follows:

$$\begin{aligned} p(\pi) = \prod_{c \in \pi} & p(t(c)|l(c), H(c)) \\ & .p(h(c)|t(c), l(c), H(c)) \\ & .p(e(c)|l(c), t(c), h(c), H(c)) \end{aligned} \tag{2.14}$$

where $l(c)$ is the label of c (e.g. np, vp and so on) and $H(c)$ includes the label $m(c)$, head $i(c)$ and head-part-of-speech $u(c)$ for the parents of c. When it gets clear for the model to which constituent it refers, the c is removed such that:

$$\begin{aligned} p(\pi) = \prod_{c \in \pi} & p(t|l, m, u, i) \\ & .p(h|t, l, m, u, i) \\ & .p(e|l, t, h, m, u) \end{aligned} \tag{2.15}$$

As shown in Equation (2.15), only $p(h|t, l, m, u, i)$ which is the distribution for the head of c considers two lexical items i and h. The other conditional distributions are conditioned on one lexical item, either i or h.

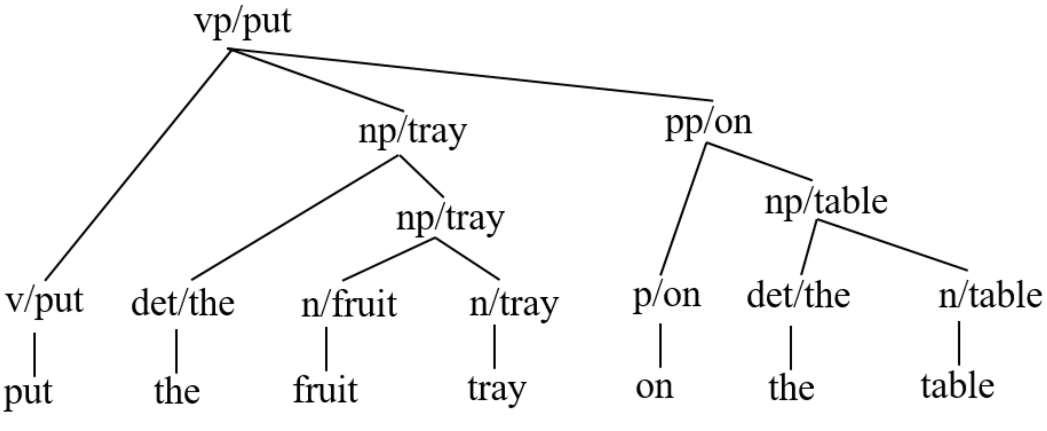


Figure 2.1: A parse tree including head information (adapted from Charniak (2001))

In training time, the PCFG probability distributions are estimated from a tree-bank. The PCFG model defines a tight probability distribution over strings so that the distributions would sum to one, making the proper for LMs. More detail of the parsing-based LM can be found in Charniak (2001) and Johnson et al. (2004).

2.3.3 Recurrent Neural Network Language Models

According to the chain rule explained in Section 2.3.1, in order to estimate the probability of an entire sequence, we have to condition the probability of each word on all previous words. In practice, many models including n-grams cannot represent such long dependencies; therefore, they are limited to considering only a few of the preceding words. However, looking at longer contexts is usually crucial for predicting the next word. Consider this sentence as an example, "I was born in Germany about fifty years ago and I speak fluent German". Looking at the recent context, a classic LM can guess that the last word of the sequence is the name of a language but in order to predict what language exactly, it requires to see further back for the context of *Germany*. In such cases where the gap between the related context and our desire point is large, learning to connect the information is a challenge for traditional LMs which are based on Markov assumption.

Recurrent neural networks (Rumelhart et al., 1986) based language models apply the chain rule to model joint probabilities over word sequences, where the context of all previous words is encoded with a long short-term memory neural network (Hochreiter and Schmidhuber, 1997). RNNs have a chain-like nature which allows information to travel in both directions. In fact, in these models, the computations which are derived from earlier input are fed back into the network and make a loop-like structure. The loop provides RNNs with a memory by which they can theoretically capture information in arbitrarily long sequences. In order to understand the architecture of RNNs, we can think of them as multiple copies of the same network, each passing information to a successor. Consider the following diagram which illustrates an RNN being unfolded into a full network.

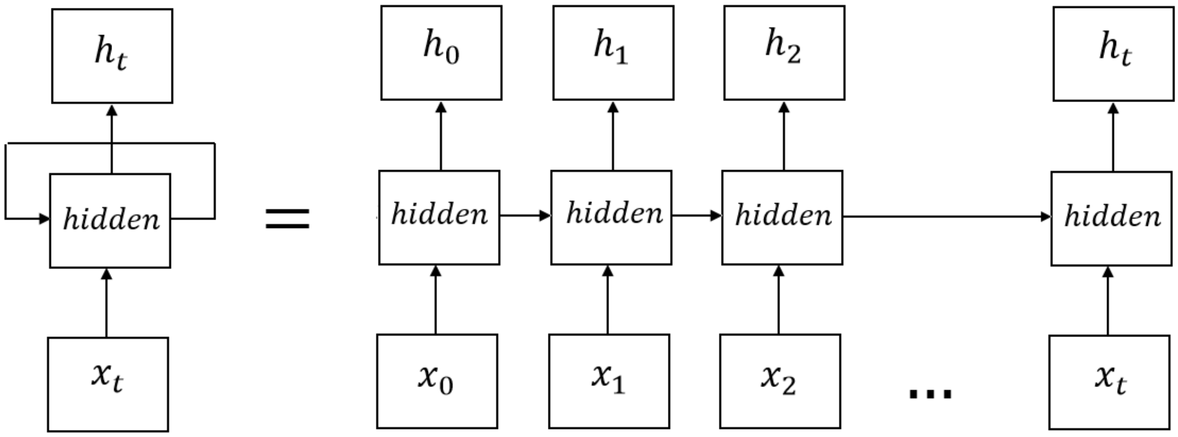


Figure 2.2: An unrolled recurrent neural network

In the above diagram, a number of hidden units take some input x_t and outputs a value h_t. For the language modelling task, the input to the RNN is a sequence of words and the output is a probability distribution over the next word. Although RNNs are theoretically capable of handling long dependencies, in practice they cannot learn to do so. It is because the gradient of a long sequence tends to be exponentially smaller in comparison with that of a short sequence. As a result, gradients which are propagated over many steps will either vanish or explode for long sequential data (Goodfellow et al., 2016).

Long short-term memory networks are particular type of RNN which are able to learn long dependencies. The main idea behind LSTMs is to create paths through time so that their derivatives neither vanish nor explode. Thus, LSTMs can be seen as a gated RNN which have a fundamentally similar architecture to recurrent networks but apply a different function for computing hidden states. LSTMs have internal self-loop cell states c_t that are connected to each other and replace the hidden units of ordinary recurrent networks. Each cell has the same inputs and outputs similar to standard recurrent network, but has more parameters. Within each cell state, there are several gates which are responsible for removing or adding information to the cell state. In the following part, we look at the architecture of LSTM and structure of cell state in more detail.

As shown in Figure 2.3, an LSTM is composed of an *input*, *hidden* and *output layer*, where it takes the word x at time t, does some computation in cell states c_t and predicts the next word h_t. Instead of inputting words as unique, discrete ids, LSTMs can input distributed representation of words by constructing word embedding. The idea of word embedding (Mikolov et al., 2013) is to represent words in a dense vector so that each word of the vocabulary is mapped into a vector of real numbers. In this technique, words are represented in a continuous vector space where semantically similar words are mapped to nearby points. Such dense representation of words can avoid data sparsity which is a common problem in count-based LM techniques such as higher-order n-gram models. The output layer of the model usually applies a softmax function to represent a probability distribution over n different possible outcomes.

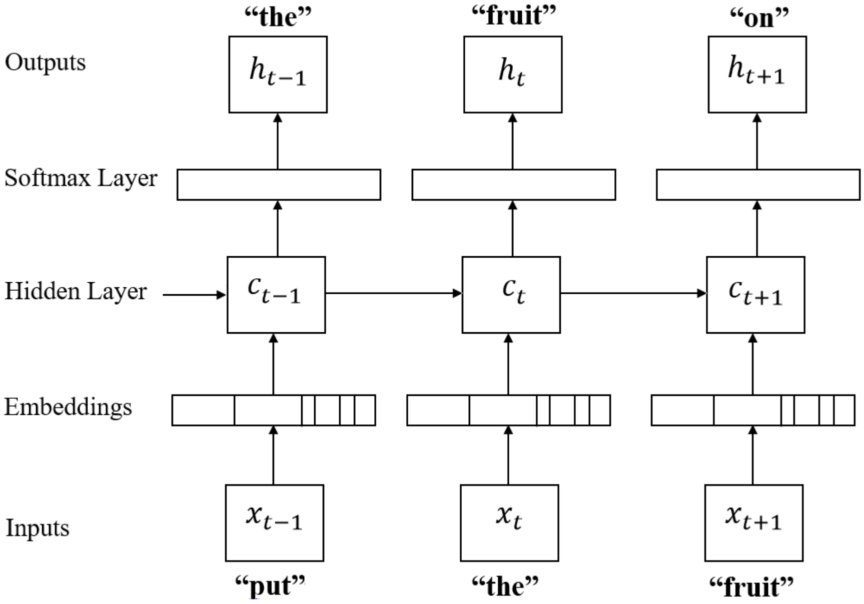


Figure 2.3: The architecture of a long short-term memory neural network

The structure of cell state c_t is shown in Figure 2.4. The cell state is the core component of an LSTM which consists of four neural network layers that are composed out of a sigmoid neural network layer σ and a pointwise multiplication operation $\bigotimes$. The layers within a cell state construct a gating mechanism in the LSTM model. The computations conducted inside the cell state is discussed step-by-step in the following part.

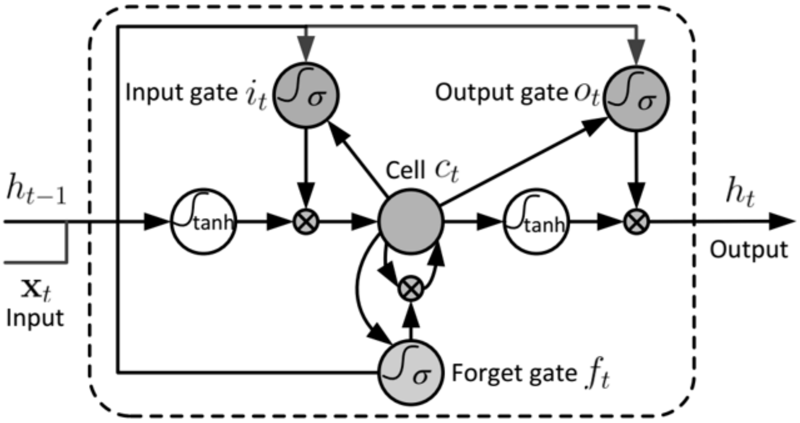


Figure 2.4: The structure of an LSTM cell (Zhu et al., 2016)

Upon receiving an input x_t, the LSTM decides what information should be removed from the cell state using a *forget gate*. The *forget gate layer* looks at the output of the previous cell state h_{t-1} and current input x_t and outputs either 0 (means to discard) or 1 (means to keep) for each number in the cell state c_{t-1}. The computation of this step is given in Equation (2.16), where W and b are weight and bias matrices and f_t is the forget gate.

$$f_t = \sigma(W_f h_{t-1} + W_f x_t + b_f) \tag{2.16}$$

In the next step, the model decides which values to update and which information to store. The information is updated by a sigmoid layer called an *input gate layer*. Then, a tanh layer makes a vector of new candidate values c_t which can be added to the state. Next, these two steps are combined to create an update to the state.

$$i_t = \sigma(W_i h_{t-1} + W_i x_t + b_i) \tag{2.17}$$

$$\widetilde{c_t} = tanh(W_c h_{t-1} + W_c x_t + b_c) \tag{2.18}$$

Now, the LSTM updates the old cell state c_{t-1} into the new cell state c_t. According to Equation (2.19), it first multiplies the old state c_{t-1} by f_t to forget what it decided to forget earlier. Then, it adds $i_t * \tilde{c}_t$ which is the multiplication of new candidate state values by the scale of how much we decided to update each state value.

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \tag{2.19}$$

In the final step, the model uses an *output gate layer* to determine what to output. First, a sigmoid layer decides what parts of the cell state to output. Next, the cell state is putted through tanh and multiplied by the output of the sigmoid gate so that the model only outputs the parts it decided to.

$$o_t = \sigma(W_o h_{t-1} + W_o x_t + b_o) \tag{2.20}$$

$$h_t = o_t * tanh(c_t) \tag{2.21}$$

Training LSTM neural networks is an optimization problem where we aim at minimizing a loss function. The loss function l_t is used to compare the predictions of the network h_t with the target answers $\hat{h}_t$ in order to know how far the model is from the target output. A training method such as *back-propagation algorithm* (Rumelhart et al., 1986) feeds back the error values through the network until each cell state gets an associated error value that demonstrates its contribution to the original output. The goal of the training algorithm is to minimize the loss L over an entire word sequence of length T:

$$L = \sum_{t=1}^{T} l_t \tag{2.22}$$

where L represents the cumulative loss from step t onwards. The back-propagation algorithm reduces the error value until the model learns the training data. The training starts with random weights and the goal is to adjust them in such a way that the error is minimized. To do so, the algorithm computes the gradient of the loss function with respect to the weights and uses it to correct the initial weights:

$$\frac{dL}{dw} = \sum_{t=1}^{T} \sum_{i=1}^{M} \frac{dL_t}{dh_{it}} \frac{dh_{it}}{dw} \tag{2.23}$$

where h_{it} is the scalar corresponding to the output of ith cell state and M is the total number of cell states. The gradient is fed to an optimization method such as *stochastic gradient descent*, shown in Equation (2.24), to update the weights in an attempt to minimize the loss function. The weights of each connection are modified so that the value of the loss function can decrease by some small amount. This process is repeated for adequately large number of training epochs until the network converges to some state where the error is small.

$$\triangle w = -\alpha \frac{dL}{dw} \tag{2.24}$$

where α is the learning rate, a positive scalar determining the size of the step. To update the current weight w:

$$w^{\star} = w - \triangle w \tag{2.25}$$

where $w^{\star}$ is the updated weight.

In order to prevent the model from overfitting, a regularization technique such as *dropout* (Srivastava et al., 2014) is applied. The idea of dropout is to temporarily drop some random units and their incoming and outcoming connections from the neural network during training so that they are not co-adapted too much. As a result, the model becomes less sensitive to specific weights of the units and can generalize better.

2.4 Summary

In this chapter, we have described speech disfluency from theoretical and computational points of view. According to psycholinguistic studies, speech disfluency has a “regular” trend in spontaneous speech, so it is possible to model and automate the task of speech disfluency detection. The approaches to disfluency detection fall into three main categories: *sequence tagging*, *parsing-based* and *noisy channel models*. Noisy channel models are one of the strong methods which have been applied to disfluency detection task. The intuition of the NCMs is that there is a similarity between reparandum and repair, which seems a good indicator of disfluency. However, the NCMs suffer from lack of an effective language model to capture the complex structure of language. On the other hand, language modelling can help identifying disfluency. We have reviewed three important language models including *Kneser-Ney smoothed* 4*-gram*, *parsing-based* and *long short term memory neural network*. The traditional LMs, such as n-gram, are based on Markov assumption, so they condition the probability of the next word only on a few preceding words. In other words, they use a short context of words to predict the next words; therefore, n-gram models are not adequately effective for capturing long dependencies between words. Parsing-based LMs which use parse trees of a sentence to estimate its probability are very specific for language domain. LSTM LMs use a gating mechanism to guarantee proper propagation of information through many steps. LSTMs are the state-of-the-art language modelling technique which enable training long dependencies between words. Capturing long dependencies can be useful for the disfluency detection task.

Chapter 3

LSTM Noisy Channel Model

In this chapter, we introduce a new model for detecting disfluencies in spontaneous speech transcripts, called *LSTM Noisy Channel Model*. The proposed model takes three steps to detect disfluencies as follows:

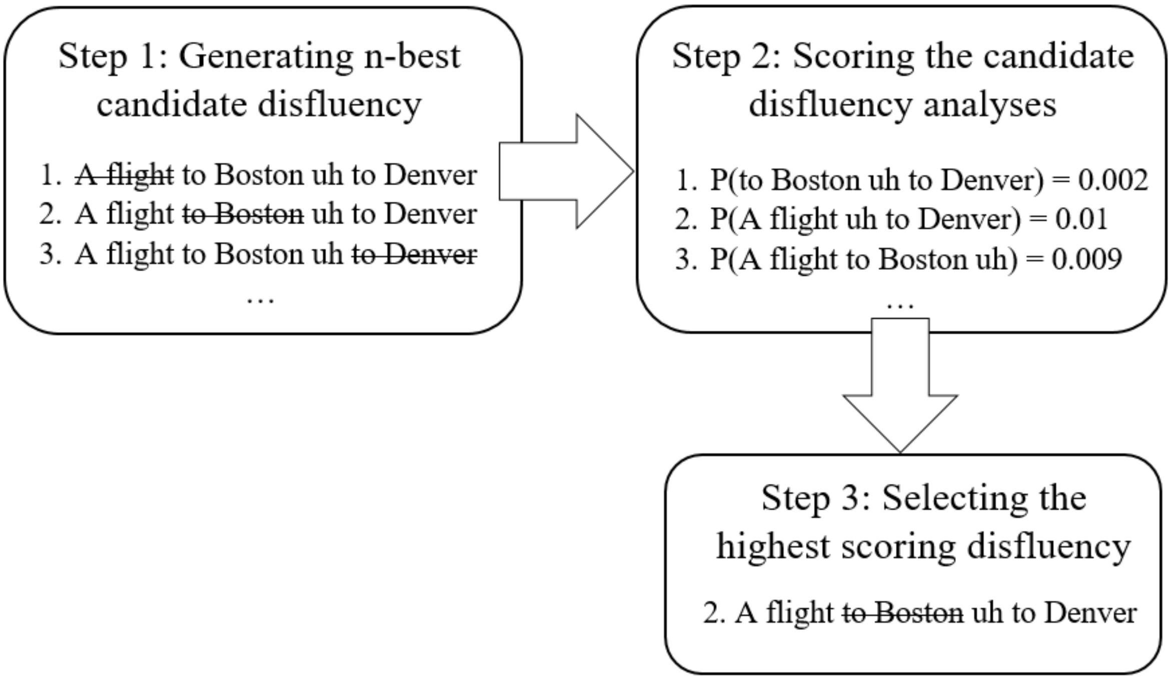

Step 1: Generating n-best candidate disfluency
1. ~~A flight~~ to Boston uh to Denver
2. A flight ~~to Boston~~ uh to Denver
3. A flight to Boston uh ~~to Denver~~
…
Step 2: Scoring the candidate disfluency analyses
1. P(to Boston uh to Denver) = 0.002
2. P(A flight uh to Denver) = 0.01
3. P(A flight to Boston uh) = 0.009
…
Step 3: Selecting the highest scoring disfluency
2. A flight ~~to Boston~~ uh to Denver

Figure 3.1: The block diagram of the LSTM noisy channel model

As shown in Figure 3.1, the first step is generating candidate disfluency analyses. We use the noisy channel model introduced by Johnson and Charniak (2004) to find 25-best candidate disfluency analyses for each sentence. Step 2 is the contribution of this thesis, where we use a long short-term memory neural network language model to score the underlying fluent versions of each disfluency analysis generated by the NCM in step 1. The LSTM LM probabilities along with other features are used in the MaxEnt reranker introduced by Johnson et al. (2004) to select the most plausible analysis. In the following sections, we describe the proposed LSTM-NCM and its components in details.

3.1 Description

We use a noisy channel model (Johnson and Charniak, 2004) to find “rough copies” that are likely to indicate disfluencies. The intuition behind the NCM of disfluency is that reparandum and repair words are closely related to each other. In other words, the repair and reparandum words are usually the same or very similar words in roughly the same order. According to Johnson and Charniak (2004), more than three-fifth of disfluent structures in Switchboard corpus contain identical words in reparandum and repair. Thus, the similarity between reparandum and repair words can be strong evidence of disfluency.

The noisy channel model uses Tree Adjoining Grammars to provide a systematic way of formalising the channel model. The polynomial-time dynamic programming parsing algorithms of TAG can be used to search for likely repairs when they are used with simple language models such as a bigram LM. Using the TAG channel model and bigram language model, the NCM produces 25-best initial analyses about the locations of disfluencies for each sentence. In fact, the NCM estimates the probability of each sentence-level analysis using the product of TAG channel model and a bigram LM, and proposes the analyses with highest probability according to this product model. The fluent version of each disfluency analysis is scored by an LSTM language model. Then, a Maxent classifier reranks the analyses and selects the highest scoring one using different features including LSTM LM probabilities. In the following sections, we look at the components of the NCM i.e. channel model and language model and also describe the LSTM LM and MaxEnt reranker in detail.

3.2 Channel Model

A channel model $P(Y|X)$ estimates the probability of sentence Y being a disfluent version of sentence X by mapping words of source sentence X into those of sentence Y. Due to the relation between reparandum and repair words, disfluent sentences may contain an unbounded number of crossed dependencies, as shown in Figure 3.2. Capturing such dependencies is outside of the expressive power of context-free or finite-state grammars.

X = but their system was great

Y = but their but their system was was great

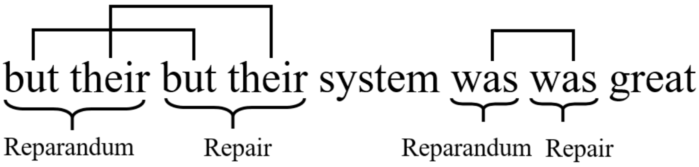


Figure 3.2: A disfluency structure with crossing dependencies between reparandum and repair

Tree Adjoining Grammars (Johnson and Charniak, 2004) are more expressive formalism and mildly context-sensitive grammars that systematically formalize the channel model. The TAG channel model encodes the crossed dependencies of speech disfluency, rather than reflecting the syntactic structure of the sentence. The TAG transducer is effectively a simple first-order Markov model which generates each word in the reparandum conditioned on the preceding word in the reparandum and the corresponding word in the repair. The TAG channel model is responsible for capturing the "rough copy" relationship between repair and reparandum words by generating possible alignments between them. For instance, Z is a possible mapping between words of sentence Y and words of source sentence X. The string Z is a sequence of pairs where the first component of each pair is an element of the string Y, and the second component is the corresponding element of the string X. More information about the TAG channel model can be found in Johnson and Charniak (2004).

Z = but:∅ their:∅ but:but their:their system:system was:∅ was:was great:great

3.3 Language Model

The language model of the NCM is responsible for evaluating the fluency of the sentence with disfluency removed. An effective LM is expected to assign a very high probability to a fluent sentence X (e.g. *but their system was great*) and a lower probability to a sentence Y which still contains disfluency (e.g. *but their but their system was was great*). However, using a strong LM within the NCM is computationally complex because the NCM uses dynamic programming in order to make the computation tractable. In fact, the TAG dynamic programming algorithms can be used to effectively search for repairs if the intersection (in language terms) of the TAG channel model and the language model itself can be describable by a TAG (Johnson and Charniak, 2004). Thus, to guarantee this, a finite state language model such as bigram is used within the noisy channel model. Using the TAG channel model together with a bigram language model, the NCM generates 25-best candidate analyses as follows:

#	**candidate disfluency analysis**
1	but **E** their **E** but _ their **E** system _ was **E** was _ great _ $1.25e-09, 165788, 3, 6.65e-10, 3.47e-07, 0.52, 3.29e-19$
2	but **E** their **E** but _ their _ system _ was **E** was _ great _ $0.0002, 3.44e+06, 2, 0.0001, 3.47e-07, 0.66, 6.75e-16$
3	but _ their **E** but _ their _ system _ was **E** was _ great _ $8.32e-06, 106960, 2, 5.69e-06, 6.58e-06, 0.68, 1.28e-14$
4	but **E** their **E** but _ their _ system _ was _ was _ great _ $0.019, 66934.2, 1, 0.014, 2.57e-05, 0.76, 3.73e-11$
5	but **E** their **E** but _ their _ system _ was **E** was **E** great _ $8.11e-10, 38926.8, 3, 5.07e-10, 1.50e-09, 0.62, 3.91e-21$

Table 3.1: The top 5 candidate disfluency analyses and corresponding noisy channel model scores. *E* and _ indicate the preceding word is reparandum and fluent, respectively.

The bigram LM, trained on the fluent version of training data, is too simple to capture complex language structure. On the other hand, it seems helpful to use an LM that is sensitive to more global properties of the sentence. That's why we use the state-of-the-art recurrent neural network language modelling to rescore the initial disfluency analyses generated by the NCM.

3.3.1 LSTM

We use long short-term memory neural networks for training LMs. LSTMs, described in Section 2.3.3, are particular type of recurrent neural networks achieving state-of-the-art performance in many tasks including language modelling (Mikolov et al., 2010; Jozefowicz et al., 2016). The core of the LSTM model consists of a cell state which includes gates to weigh input and history impact at a particular time. The gating mechanism of LSTM allows the model to determine the relative significance of input and history and alleviate the vanishing gradient problem. Therefore, LSTM is able to learn long dependencies between words, which can be beneficial for the disfluency detection task. LSTM also allows for adopting a distributed representation of words by constructing word embedding.

We train two types of LSTM language models including forward and backward. Using backward LMs can be useful for detecting disfluencies. Backward LMs input sentences in reverse order, so they can probably capture the unexpected word sequence following the interruption point which is an important indicator of disfluency. The task of the LSTM LMs is to score the underlying fluent sentences of candidate disfluency analyses produced by the noisy channel model. When scoring the sentences, we will have sentences of different lengths on input. We need to handle sentences of variable lengths by padding each sentence with a special <PAD> symbol to reach a pre-defined length. This way, all sentences in the mini-batch will have the same length. Since we do not want the model to consider the padded symbols in loss function, we mask out the padded words given the actual length of each sentence. For example, consider the underlying fluent sentences of candidate analyses illustrated in Table 3.1. If the pre-defined maximum length is 6, we will add the <PAD> symbol at the end of each sentence with fewer words, so that all sentences will have equal number of words, as shown in Table 3.2.

#	**padded sentences**					
1	but	system	was	great	<PAD>	<PAD>
2	but	their	system	was	great	<PAD>
3	but	but	their	system	was	great
4	but	their	system	was	was	great
5	but	their	system	great	<PAD>	<PAD>

Table 3.2: The underlying fluent sentences of candidate analyses padded with a special symbol at the end to reach the maximum length of 6 words.

The forward LSTM LM feeds the padded sentences as shown above. For the backward model, the input sentences include the fluent words in reverse order plus some <PAD> symbols at the end[1]. Inputting the padded sentences, each forward and backward LSTM language model assigns a probability to each sentence. The top 5 disfluency analyses of the NCM and corresponding forward and backward LSTM LM scores given to the fluent words (i.e. words preceding "_") are illustrated in Table 3.3.

#	**candidate**	**forward**	**backward**
1	but **E** their **E** but _ their **E** system _ was **E** was _ great _	$9.56e-10$	$9.63e-16$
2	but **E** their **E** but _ their _ system _ was **E** was _ great _	$2.23e-10$	$6.12e-15$
3	but _ their **E** but _ their _ system _ was **E** was _ great _	$2.23e-10$	$6.12e-15$
4	but **E** their **E** but _ their _ system _ was _ was _ great _	$1.19e-13$	$8.27e-18$
5	but **E** their **E** but _ their _ system _ was **E** was **E** great _	$6.77e-13$	$7.68e-14$

Table 3.3: Top 5 candidate disfluency analyses of the NCM along with forward and backward LSTM LM scores. E and _ indicate the preceding word is reparandum and fluent, respectively.

Using the noisy channel model scores, LSTM LM probabilities and other features into the MaxEnt reranker, the second disfluency analysis in Table 3.3 which is the correct answer is selected from the list of candidates proposed by the noisy channel model. As it will be discussed in the next chapter, adding LSTM LM scores will help detecting the correct disfluency analysis comparing to the baseline model.

3.4 Reranker

In order to rerank the 25-best analyses of the NCM and select the most plausible one, we apply the MaxEnt reranker proposed by Johnson et al. (2004). The MaxEnt reranker allows for incorporating a wide range of features, including local features such as those derived from the NCM and global features such as external language model scores. As features into the reranker, we use the feature set introduced by Zwarts and Johnson (2011), but instead of n-gram scores, we use the LSTM language model probabilities. The features used by Zwarts and Johnson (2011) are so good that the reranker without any external language model is already a state-of-the-art system, providing a very strong baseline for our work.

Two types of features including model-based and surface pattern features are used in the reranker. The model-based features are the log probabilities produced by the noisy channel model and the LSTM language model as follows:

- LMP: Forward and backward LSTM language model scores indicating the probabilities of the underlying fluent sentences,

---

[1] For example, the first input sentence of Table 3.2 for the backward LM will be *great was system but <PAD> <PAD>*.

- NCLogP: The log probability of the entire noisy channel model,
- NCTransOdd: The channel model probability,
- LogFom: Sum of the log LM probability and the log TAG channel model probability.

The surface pattern features are boolean indicators which capture identical words, immediate area around an n-gram and location of disfluency as follows:

- CopyFlags_X_Y: If there is an exact copy in the input text of length X ($1 \leq X \leq 3$) and the gap between the copies is Y ($0 \leq Y \leq 3$),
- WordsFlags_L_n_R: Number of flags to the left (L) and to the right (R) of a trigram area ($0 \leq L, R \leq 1$),
- SentenceEdgeFlags_B_L: It captures the location and length of disfluency. The boolean B presents sentence initial or sentence final disfluency and L ($1 \leq L \leq 3$) records the length of the flags.

We give the following analysis as an example (Zwarts and Johnson, 2011):

but E *but* _ *that* _ *does* _ *not* _ *work*

The language model features are the probability calculated over the fluent part. NCLogP, Log- Fom and NCTransOdd are present with their associated value. The following binary flags are present:

CopyFlags_1_0 (E _)
WordsFlags:0:1:0 (but E)
WordsFlags:0:1:0 (but _)
WordsFlags:1:1:0 (E but _)
WordsFlags:1:1:0 (_ that _)
WordsFlags:0:2:0 (but E but _) etc.
SentenceEdgeFlags:0:1 (E)
SentenceEdgeFlags:0:2 (E _)
SentenceEdgeFlags:0:3 (E _ _)

The reranking result for Table 3.4 is shown in the following table. It should be mentioned that if the correct analysis does not appear in the 25-best candidates produced by the noisy channel model, the reranker chooses the candidate closest to the correct analysis.

#	**candidate**
2	but **E** their **E** but _ their _ system _ was **E** was _ great _
1	but **E** their **E** but _ their **E** system _ was **E** was _ great _
4	but **E** their **E** but _ their _ system _ was _ was _ great _
3	but _ their **E** but _ their _ system _ was **E** was _ great _
5	but **E** their **E** but _ their _ system _ was **E** was **E** great _

Table 3.4: The 5 reranked candidate disfluency analyses.

3.5 Summary

In this chapter, we describe the framework of a new model for detecting disfluencies from spontaneous speech transcripts. The proposed model uses a noisy channel model to find an alignment between reparandum and repair words. The TAG channel model together with the bigram language model generates 25-best candidate disfluency analyses. Then, different long short-term memory neural network language models are applied to rescore the fluent parts of each analysis of the noisy channel model. The LSTM language model probabilities, NCM scores and surface pattern features are used into a MaxEnt reranker to select the most plausible disfluency analysis.

Chapter 4

Experiments and Results

In this chapter, we present the experiments and results of applying the proposed model for disfluency detection. We also describe the corpora used for training LSTM language models, hyper-parameters, setting of the LSTM LM and evaluation metrics used to compare the LSTM-NCM model with the baseline and previous disfluency detection models.

4.1 Corpora for Language Modelling

The forward and backward LSTM language models are trained on Fisher (Cieri et al., 2004) and Switchboard (Godfrey and Holliman, 1993) corpora. Fisher consists of 2.2×10^7 tokens of transcribed text, but disfluencies are not annotated in it. Switchboard is the largest available corpus (1.2×10^6 tokens) in which disfluencies are annotated according to Shriberg's (1994) scheme. The bigram LM of the NCM is already trained on the fluent version of Switchboard corpus, but we also aim to use LSTM LMs for reranking. The language model part of the noisy channel model already uses a bigram language model based on Switchboard, but in the reranker we would like to also use LSTM LMs for reranking. Direct use of Switchboard for training LSTM LMs is slightly problematic. The reason is that if the training data of Switchboard is used both for predicting language fluency and optimizing the loss function, the reranker will overestimate the weight related to the LM features extracted from Switchboard. This is because the fluent sentence itself is part of the language model training data (Zwarts and Johnson, 2011). We can solve this problem by applying a k-fold cross-validation technique ($k = 20$) to train the LSTM language models when using this corpus. Applying the k-fold cross validation technique, we divide the Switchboard training data into 20 folds. Each time, we use the other 19 folds to train our LSTM language model and apply the trained LSTM LM to generate scores that will be used for training the reranker on the sentences of the one remaining fold. Using this technique, we have to build 20 different LSTM LMs on Switchboard.

We also train Kneser-ney smoothed 4-gram language models to compare their performance with that of LSTM LMs in Section 4.4. Totally, we train two n-gram models, one on Switchboard and one on Fisher, and four LSTM LMs (2 corpora $\times$ 2 models of forward and backward).

We follow Charniak and Johnson (2001) in splitting Switchboard corpus into training, development and test set. The training data consists of all sw[23]$*$.dps files, development training consists of all sw4[5-9]$*$.dps files and test data consists of all sw4[0-1]$*$.dps files. Following Johnson and Charniak (2004), we remove all partial words and punctuations from the Switchboard and Fisher training data. Although partial words are very strong indicators of disfluency, standard speech recognizers never produce them in their outputs, so this makes our evaluation both harder and more realistic. We consider a vocabulary size of 10000 words which are extracted from Switchboard training data. The words that are out of vocabulary are marked with a special <UNK> token in Switchboard and Fisher corpora.

4.2 LSTM Setting

We consider three configurations of the model that we call *small*, *medium* and *large* for training forward and backward LSTM language models. The details of the model configurations are given in Table 4.1. The configurations are different in terms of size of the LSTMs and the set of hyper-parameters used for training.

config. → **hyper-parameters ↓**	**small**	**medium**	**large**
number of layers	2	2	2
number of hidden units	200	650	1500
initial learning rate	1	1	1
number of epochs trained with initial learning rate	4	6	14
learning rate decay for each epoch after max. epoch	0.5	0.8	1/1.15
total number of epochs	13	39	55
initial scale of the weights	0.1	0.05	0.04
probability of keeping weights in the dropout layer	0.5	0.8	0.35
max. permissible norm of the gradient	5	5	10
embedding size	200	650	1500

Table 4.1: Three different configurations of the model and their corresponding hyper-parameters

The model has two LSTM layers and is trained using truncated backpropagation through time algorithm with mini-batch size 20. For training the LSTM LMs on the cleaned-up Switchboard text, we pad each sentence to the maximum length of sentences in the corpus which is 109 words. For training our model on Fisher, however, we limit the maximum length of sentences due to the high computational complexity of longer histories in the LSTM. In our experiments, padding sentences upto maximum 50 words leads to good results on Fisher. All parameters including word embedding have random initialization. We use TensorFlow (Abadi et al., 2015) to write the LSTM codes.

Training one epoch of the small model on Switchboard corpus takes 20 minutes, while it takes 0.5 and 1.5 hours to train one epoch of the medium and large models. The small model on Fisher needs

1.5 hours per epoch, whereas the medium and large models take about 3 and 8.5 hours to train one epoch using one GPU and one CPU. We select the best LSTM LM using the three configurations of the model. The results including f-score and perplexity on the development set are presented for the small, medium and large LSTM LMs in Tables 4.2 and 4.3.

f-score	**small**	**medium**	**large**
Switchboard	86.1	86.4	85.9
Fisher	86.2	85.6	85.2

Table 4.2: F-score on the dev set for the small, medium and large LSTM LMs.

perplexity	**small**	**medium**	**large**
Switchboard	53.4	49.9	58.1
Fisher	100.5	122.4	135

Table 4.3: Perplexity on the dev set for the small, medium and large LSTM LMs.

As illustrated above, the small configuration of the model results in the best performance for Fisher. The medium model, however, slightly improves f-score (0.3%) for Switchboard corpus, comparing to the small and large models but at the expense of increase in model's computational complexity. Due to the high computational cost of medium and large models and decent performance of the small model in terms of f-score and perplexity, we use the LSTM LM trained by the small configuration to report all the subsequent results in Section 4.4.

4.3 Evaluation Technique

The easiest way to evaluate a disfluency detection system is to calculate the accuracy of labels i.e. the fraction of words labelled correctly. However, since only 6% of words are disfluent in Switchboard corpus, a system that labels all words as “not edited” achieves an accuracy of 94%. Thus, accuracy is not a good measure of system performance.

To evaluate our system, we use two metrics *f-score* and *error rate*. Johnson et al. (2004) used the f-score of labelling reparanda or “edited” words, while Fiscus et al. (2004) defined an “error rate” measure, which was the number of words falsely labelled divided by the number of reparanda words. The f-score, unlike accuracy, focuses more on detecting “edited” words, as shown in Equation (4.1). In Equation (4.1), g is the number of “edited” words in the gold test data, e is the number of “edited” words generated by the system and c is the number of correct “edited” words proposed by the system.

$$fscore = \frac{2c}{g + e} \tag{4.1}$$

According to the above equation, a system which correctly labels every word obtains an f-score of 1 and a system which assigns “not edited” labels to every word achieves zero f-score.

4.4 Model Performance

We assess the proposed model for disfluency detection with all MaxEnt features described in Section 3.4 against the baseline model. The noisy channel model with exactly the same reranker features except the LSTM LMs forms the baseline model. We conduct four experiments to evaluate our model and use bootstrap significance test (Berg-Kirkpatrick et al., 2012) with p-value $< .05$ to report the results as follows.

a. LSTM-NCM vs. Baseline

We investigate the effect of using all LSTM language model scores (i.e. four LSTM LMs described in Section 4.1) as features in the MaxEnt reranker to select the best NCM analysis. As shown in Table 4.4, using LSTM language models to score the underlying fluent sentences of the analyses produced by the NCM improves the baseline results. The experiment on Switchboard and Fisher corpora demonstrates that the LSTM LMs provide information about the global fluency of an analysis that the local features of the reranker do not capture.

metric	**baseline**	**LSTM LMs**
f-score	85.3	**86.8**
error rate	27.0	**24.3**

Table 4.4: F-score and error rate on the dev set for the baseline model and LSTM-NCM. Significant results are indicated in bold (*p-value*$< .05$).

b. Different LSTM LMs

We explore the effect of using different LSTM language models including *forward*, *backward* and *bidirectional* on the disfluency detection task. As shown in Tables 4.5 and 4.6, the backward language models have better performance in comparison with the forward ones for both Switchboard and Fisher corpora. It seems when sentences are fed in reverse order, the model can more easily detect the unexpected word order associated with the reparandum to detect disfluencies. For Switchboard, the best result is achieved for the bidirectional language model, but for Fisher the highest f-score and lowest error rate are reported for the backward model. Using both Switchboard and Fisher forward language models results in better performance in comparison with either of them. It means that each forward LM contains new information that when they are used together they improve the model. On the other hand, both backward LMs decrease the f-score in comparison with either of them. It illustrates that both backward models share some information that together they do not add new information to the model. The best performing model is the one which uses all LSTM language models features.

Comparing the results of two corpora, we realize that the LSTM LM built on the fluent version of Switchboard corpus results in the greatest improvement. Both Switchboard and Fisher are transcribed text, but Switchboard is in the same domain as the test data and it is disfluency annotated. Either or both of these might be the reason why Switchboard seems to be better in comparison with Fisher which is a larger corpus and might be expected to make a better language model.

baseline	85.3		
corpus	**forward**	**backward**	**both**
Switchboard	86.1	86.6	**86.8**
Fisher	86.2	86.5	86.3
Both	86.4	86.3	**86.8**

Table 4.5: F-scores on the dev set for a variety of LSTM language models. Significant results are indicated in bold (*p-value*$<$ .05).

baseline	27.0		
corpus	**forward**	**backward**	**both**
Switchboard	25.5	24.8	**24.3**
Fisher	25.6	25.0	25.3
Both	25.1	25.3	**24.3**

Table 4.6: Expected error rates on the dev set for a variey of LSTM language models. Significant results are indicated in bold (*p-value*$<$ .05).

c. LSTM vs. N-gram LMs

We compare the performance of Kneser-Ney smoothed 4-gram language models with the LSTM corresponding on the reranking process of the noisy channel model. We estimate the 4-gram models and assign probabilities to the fluent parts of disfluency analyses using the SRILM toolkit (Stolcke, 2002). As Tables 4.7 and 4.8 show including scores from a conventional 4-gram language model does not improve the model's ability to find disfluencies, suggesting that the n-gram model adds no new information to the reranker and LSTM model contains all the useful information that the 4-gram model does. Moreover, the results show that the ability of LSTM for capturing longer contexts is useful for detecting the disfluent parts of speech transcripts.

baseline	85.3		
corpus	**4-gram**	**LSTM**	**both**
Switchboard	85.9	**86.8**	**86.8**
Fisher	85.9	86.3	86.3
Both	85.7	**86.8**	**86.8**

Table 4.7: F-score for 4-gram, LSTM and combination of both language models. Significant results are indicated in bold (*p-value*$<$ .05).

baseline	27.0		
corpus	**4-gram**	**LSTM**	**both**
Switchboard	25.9	**24.3**	24.4
Fisher	26.1	25.3	25.7
Both	26.5	**24.3**	24.4

Table 4.8: Expected error rates for 4-gram, LSTM and combination of both language models. Significant results are indicated in bold (*p-value*$< .05$).

d. LSTM-NCM vs. Other Disfluency Models

We compare our best model on the development set to the state-of-the-art methods in the literature. According to Table 4.9, the LSTM-NCM outperforms the results of prior work, achieving a state-of-the-art performance of 86.8. It also has better performance in comparison with Ferguson et al. (2015) and Zayat et al.'s (2016) models, even though they use richer input that includes prosodic features or partial words.

Model	**f-score**
Yoshikawa et al. (2016)	62.5
Johnson and Charniak (2004)	79.7
Johnson et al. (2004)	81.0
Rasooli and Tetreault (2013)	81.4
Qian and Liu (2013)	82.1
Honnibal and Johnson (2014)	84.1
Ferguson et al. (2015) *	85.4
Zwarts and Johnson (2011)	85.7
Zayats et al. (2016) *	85.9
LSTM-NCM	**86.8**

Table 4.9: Comparison of the LSTM-NCM to state-of-the-art methods on the dev set. *Models have used richer inputs.

4.5 Discussion

In this section, we discuss the types of disfluent structures that the LSTM-NCM can capture and those which are difficult for the model to detect. In general, the LSTM-NCM model can recognize repetition and correction disfluencies but it has a poor performance in detecting restarts. Giving examples from the development data, we characterize the disfluencies that the LSTM-NCM model captures and the ones that it does not. In the following examples, *GOLD*, *LSTM-NCM* and *N-gram* refer to the correct disfluency analysis, the one predicted by the LSTM-NCM and n-gram prediction, respectively. In cases where no LSTM-NCM prediction is given, it means that the LSTM-NCM prediction is same as the gold analysis. The symbols E and _ indicate the preceding word is reparandum and fluent.

Type A.1: Repetition

As shown below, the LSTM-NCM is able to capture repetition disfluencies, even if they are long or have stutter-like style.

GOLD 1: so _ from E from _ that _ standpoint _ it E ’s E pretty E small E it _ ’s _ pretty _ small _

GOLD 2: i _ mean _ you E did E n’t E have E uh _ you _ did _ n’t _ have _ video _ games _

GOLD 3: she _ said _ she _ ’ll _ never _ put _ her _ child _ in E a E in E a E in E a E in E a E in _ a _ preschool

Type A.2: Correction

Another type of disfluency that the LSTM-NCM can identify is correction, which is more difficult to detect, comparing to repetition.

GOLD 1: but _ uh _ when E i E was E when _ my _ kids _ were _ young _ i _ was _ teaching _ at _ a _ university _

GOLD 2: but _ we E ’re E shopping E around E as E far E as E well _ i _ ’m _ shopping _ around _ as _ far _ as _ trying _ to E get E uh _ that _ ’s _ why _ i _ ’m _ doing _ this _ to _ get _ some _ extra _ money _ and _ uh _ getting _ pledge _ sheets _ for _ the _ boy _ scouts _

GOLD 3: and E and _ i E was E n’t E i _ did _ n’t _ mean _ that _

Type A.3: Complex

The LSTM-NCM can also detect complex, nested disfluencies, where one disfluent structure is embedded in another one.

GOLD 1: but _ really _ i E ’m E i E happy E i E i E well _ i _ ’m _ curious _ how _ other _ people _ live _

GOLD 2: we E do E n’t E we E we _ were _ n’t _ sure _ how _ to _ do _ that _

GOLD 3: so E so _ that _ ’s _ i _ think _ one _ of _ the _ reasons _ i _ do _ n’t _ need _ to _ budget _ is _ that _ i E do E n’t E have E i E do E n’t E i _ do _ n’t _ have _ to _ hold _ myself _ back _ from _ buying _ that _ expensive _ thing _

The following examples show the disfluent structures that the LSTM-NCM could not capture.

Type B.1: Restart

According to the following examples, the LSTM-NCM usually fails at detecting restart disfluencies. The reason is that our proposed model is based on the NCM intuition that there is a similarity between repair and reparandum. This intuition, however, cannot be used for restarts where the repair is null and speaker abandons one sentence to start another one.

GOLD 1: so _ we E 're E uh _ our _ discussion _ 's _ about _ uh _ the _ care _ of _ the _ elderly _

LSTM-NCM 1: so _ we _ 're _ uh _ our _ discussion _ 's _ about _ uh _ the _ care _ of _ the _ elderly _

GOLD 2: i E i E think E sometimes E you _ know _ i _ 've _ noticed _ uh _ people E asking E for E uh _ some _ of _ the _ patients _ asking _ for _ things _ uh _ just _ repetitively _ and E but _ things _ that _ are _ not _ reasonable _

LSTM-NCM 2: i E i _ think _ sometimes _ you _ know _ i _ 've _ noticed _ uh _ people E asking E for E uh I some _ of _ the _ patients _ asking _ for _ things _ uh _ just _ repetitively _ and _ but _ things _ that _ are _ not _ reasonable _

GOLD 3: uh _ but _ i E uh E i E 've E heard E i _ do _ n't _ know _ this _ for _ a _ fact _ but _ i _ 've _ heard _ that _ a _ lot _ of _ families _ really _ do _ n't _ spend _ a _ great _ deal _ of _ time _ together _

LSTM-NCM 3: uh _ but _ i E uh I i _ 've _ heard _ i _ do _ n't _ know _ this _ for _ a _ fact _ but _ i _ 've _ heard _ that _ a _ lot _ of _ families _ really _ do _ n't _ spend _ a _ great _ deal _ of _ time _ together _

We also discuss the types of disfluencies captured by LSTM-based model, but not by the n-gram one. In general, the LSTM-based model outperforms the n-gram model in finding the disfluent structures which are embedded in long contexts. The n-gram based model, on the other hand, can only capture simple repetition disfluencies where reparandum and repair consist of few words and the gap between reparandum and repair is small. In the following part, we show several examples selected from the development data in order to categorize the types of disfluencies that the LSTM model capture, but not the n-gram.

Type C.1: Correction

One type of disfluency that the LSTM model can capture but the n-gram model cannot is correction disfluencies. It seems that when the correction disfluent structure includes unequal number of words in reparandum and repair (i.e. non-parallel reparandum and repair), the LSTM model excels the corresponding n-gram.

GOLD 1: and _ they E 've E most _ of _ them _ have _ been _ pretty _ good _

N**-gram 1**: and _ they _ 've _ most _ of _ them _ have _ been _ pretty _ good _

GOLD 2: well _ five E are E i _ mean _ four _ of _ them _ are _ grown _

N**-gram 2**: well _ five _ are E i _ mean _ four _ of _ them _ are _ grown _

Type C.2: Complex

The LSTM model can detect complex, nested disfluencies, particularly if it contains a stutter-like repetition. As the examples demonstrate, the n-gram model simply detect the first reparandum words but not the second ones.

GOLD 1: we E we _ could _ n't _ survive _ in E a E in E a E juror E in _ a _ trial _ system _ without _ a _ jury _

N**-gram 1:** we E we _ could _ n't _ survive _ in E a E in _ a _ juror _ in _ a _ trial _ system _ without _ a _ jury _

GOLD 2: so _ you E 're E you E 're E down E you _ 're _ downtown _

N**-gram 2:** so _ you E 're E you _ 're _ down _ you _ 're _ downtown _

Type C.3: Fluent Repetition

According to the following examples, when there are two identical fluent words in a small window of context, the n-gram model incorrectly detect them as disfluency.

GOLD 1: and _ you E probably E if _ it _ were _ you _ you _ probably _ would _ n't _ want _ someone _ choosing _ a _ place _ for _ you _ to _ live _ based _ on _ lowest _ price _

N**-gram 1:** and _ you E probably E if _ it _ were _ you E you _ probably _ would _ n't _ want _ someone _ choosing _ a _ place _ for _ you _ to _ live _ based _ on _ lowest _ price _

GOLD 2: because _ uh _ anytime _ the _ government _ uh _ is _ the _ government _ against _ the _ individual _ you E need E you _ need _ the _ protection _ of _ ordinary _ citizens _

N**-gram 2:** because _ uh _ anytime _ the E government E uh _ is _ the _ government _ against _ the _ individual _ you E need E you _ need _ the _ protection _ of _ ordinary _ citizens _

4.6 Summary

In this chapter, we assess the proposed LSTM noisy channel model on the task on disfluency detection. According to the experiments on the Switchboard and Fisher corpora, using the LSTM language models to score the underlying fluent sentences improves the model's ability to detect and remove disfluencies. In fact, the LSTM LM captures information about the global fluency of an analysis that the local features cannot provide. Moreover, using n-gram language model scores along with LSTM scores does not improve the results, indicating that the LSTM model captures all the plausible information that the n-gram model does. As experimental results show the proposed model outperforms the state-of-art reported in the literature, including models that exploit richer information from the input. We also discuss different types of disfluencies that the LSTM-NCM can capture and the ones it cannot. Furthermore, we analyse the types of disfluencies captured by LSTM-based model, but not by the n-gram one. The detailed analysis of the model shows that the capability of LSTM for capturing longer context is important for detecting disfluencies.

Chapter 5

Summary and Conclusions

In this thesis, we present a new model for disfluency detection from spontaneous speech transcripts. The model uses the baseline noisy channel model to produce 25-best candidate disfluency analyses for each sentence. Then, the underlying fluent sentences of each analysis is scored by a long short-term memory neural network language model. The LSTM language model scores along with noisy channel model probabilities and surface pattern features are used in a MaxEnt reranker to select the most plausible analysis. We show that using an advanced language model technique such as LSTM to rescore the outputs of the noisy channel model improves the model's ability to detect restart and repair disfluencies. The model outperforms other models reported in the literature, including models that exploit richer information from the input. According to the experimental results, the LSTM LM provides information about the global fluency of an analysis that the local features of the reranker do not capture. Moreover, we demonstrate that the backward LMs have better performance in comparison with the forward ones. It seems when sentences are fed in reverse order, the model can more easily detect the unexpected word order associated with the reparandum to detect disfluencies. In other words, that the disfluency is observed "after" the fluent repair in a backward language model is helpful for recognizing disfluencies. We also compare the performance of a Kneyser-ney smoothed 4-gram language model with the LSTM one on the reranking process of the noisy channel model. We show that capturing longer contexts is important for detecting disfluent parts of speech transcripts and deep neural language model contains all the useful information the 4-gram model does.

We make an error analysis on the proposed disfluency detection model. Although the model is strong enough to detect complex, nested repetition and correction disfluencies, it has a relatively poor performance for capturing restarts. The reason is that our proposed model is based on the NCM intuition that there is a similarity between repair and reparandum. This intuition, however, cannot be used for restarts where the repair is null and speaker abandons one sentence to start another one.

As future work, we intend to apply more complex LSTM language models such as sequence-to-sequence on the reranking process of the noisy channel model. We also intend to investigate the effect of integrating LSTM language models into other kinds of disfluency detection models, such as sequence labelling and parsing-based models.

Bibliography

Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Ebru Arisoy, Tara N. Sainath, Brian Kingsbury, and Bhuvana Ramabhadran. 2012. Deep neural network language models. 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*. Association for Computational Linguistics, Stroudsburg, PA, USA, WLM '12, pages 20–28.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in nlp. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, Stroudsburg, USA, EMNLP-CoNLL '12, pages 995–1005.

Heather Bortfeld, Silvia Leon, Jonathan Bloom, Michael Schober, and Susan Brennan. 2001. Disfluency rates in conversation: Effects of age, relationship, topic, role, and gender. *Language and Speech* 44(2):123–147.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '01, pages 124–131.

Eugene Charniak and Mark Johnson. 2001. Edit detection and parsing for transcribed speech. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics on Language Technologies*. Stroudsburg, USA, NAACL'01, pages 118–126.

Eunah Cho, Thanh-Le Ha, and Alex Waibel. 2013. CRF-based disfluency detection using semantic features for german to English spoken language translation.

Eunah Cho, Jan Niehues, and Alexander Waibel. 2014. Tight integration of speech disfluency removal into smt. In *EACL*.

Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, KyungHyun Cho, and Yoshua Bengio. 2015. Attention-based models for speech recognition. *CoRR* abs/1506.07503.

Christopher Cieri, David Miller, and Kevin Walker. 2004. Fisher English training speech part 1 transcripts LDC2004T19. Published by: Linguistic Data Consortium, Philadelphia, USA.

James Ferguson, Greg Durrett, and Dan Klein. 2015. Disfluency detection with a semi-Markov model and prosodic features. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, USA, NAACL'15, pages 257–262.

Jonathan Fiscus, John Garofolo, Audrey Le, Alvin Martin, David Pallet, Mark Przybocki, and Greg Sanders. 2004. Results of the fall 2004 STT and MDE evaluation. In *Proceedings of Rich Transcription Fall Workshop*.

Kallirro Georgila. 2009. Using integer linear programming for detecting speech disfluencies. pages 109–112.

John Godfrey and Edward Holliman. 1993. Switchboard-1 release 2 LDC97S62. Published by: Linguistic Data Consortium, Philadelphia, USA.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. Http://www.deeplearningbook.org.

Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine* 29(6):82–97.

Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Matthias Honal and Tanja Schultz. 2005. Automatic disfluency removal on recognized spontaneous speech - rapid adaptation to speaker dependent disfluencies.

Matthew Honnibal and Mark Johnson. 2014. Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics* 2(1):131–142.

Julian Hough and David Schlangen. 2015. Recurrent neural networks for incremental disfluency detection. In *Proceedings of the 16th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Dresden, Germany, pages 845–853.

Mark Johnson and Eugene Charniak. 2004. A TAG-based noisy channel model of speech repairs. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Barcelona, Spain, ACL'04, pages 33–39.

Mark Johnson, Eugene Charniak, and Matthew Lease. 2004. An improved model for recognizing disfluencies in conversational speech. In *Proceedings of Rich Transcription Workshop*.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *CoRR* abs/1602.02410.

Daniel Jurafsky and James Martin. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*. volume 1, pages 181–184.

Ellen Lau and Fernanda Ferreira. 2005. Lingering effects of disfluent material on comprehension of garden path sentences. *Language and Cognitive Processes* 20(5):633–666.

Yang Liu, Elizabeth Shriberg, Andreas Stolckeand, Dustin Hillard, Mari Ostendorf, and Mary Harper. 2006. Enriching speech recognition with automatic detection of sentence boundaries and disfluencies. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 14(5):1526–1540.

Matthew Lowder and Fernanda Ferreira. 2016. Prediction in the processing of repair disfluencies. *Language, Cognition and Neuroscience* 31(1):73–79.

Gregoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, and Geoffrey Zweig. 2015. Using recurrent neural networks for slot filling in spoken language understanding. *Trans. Audio, Speech and Lang. Proc.* 23(3):530–539.

Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Makuhari, Japan, pages 1045–1048.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems (NIPS)*. Curran Associates Inc., pages 3111–3119.

Abdel-rahman Mohamed, George Dahl, and Geoffrey Hinton. 2012. Acoustic modeling using deep belief networks. *Transactions on Audio, Speech, and Language Processing* 20(1):14–22.

Mari Ostendorf and Sangyun Hahn. 2013. A sequential repetition model for improved disfluency detection. In *Proceedings of the 14th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Lyon, France, pages 2624–2628.

Xian Qian and Yang Liu. 2013. Disfluency detection using multi-step stacked learning. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, USA, NAACL'13, pages 820–825.

Mohammad Sadegh Rasooli and Joel Tetreault. 2013. Joint parsing and disfluency detection in linear time. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Seattle, USA, pages 124–129.

David Rumelhart, James McClelland, and PDP Research Group. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press.

Alexander Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. *CoRR* abs/1509.00685.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2010. Broad-coverage parsing using human-like memory constraints. *Computational Linguistics* 36(1):1–30.

Holger Schwenk, Anthony Rousseau, and Mohammed Attik. 2012. Large, pruned or continuous space language models on a gpu for statistical machine translation. 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. Association for Computational Linguistics, Stroudsburg, PA, USA, WLM '12, pages 11–19.

Elizabeth Shriberg. 1994. *Preliminaries to a theory of speech disfluencies*. Ph.D. thesis, University of California, Berkeley, USA.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Machine Learning Research* 15:1929–1958.

Andreas Stolcke. 2002. SRILM: An extensible language modeling toolkit. In *Proceedings of International Conference on Spoken Language Processing*. Association for Computational Linguistics, Denver, Colorado, USA, volume 2, pages 901–904.

Andreas Stolcke and Elizabeth Shriberg. 1996. Statistical language modeling for speech disfluencies.

Masashi Yoshikawa, Hiroyuki Shindo, and Yuji Matsumoto. 2016. Joint transition-based dependency parsing and disfluency detection for automatic speech recognition texts. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pages 1036–1041.

Victoria Zayats, Mari Ostendorf, and Hannaneh Hajishirzi. 2014. Multi-domain disfluency and repair detection. In *Proceedings of the 15th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. Singapore, pages 2907–2911.

Victoria Zayats, Mari Ostendorf, and Hannaneh Hajishirzi. 2016. Disfluency detection using a bidirectional LSTM. In *Proceedings of the 16th Annual Conference of the International Speech Communication Association (INTERSPEECH)*. San Francisco, USA, pages 2523–2527.

Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, and Xiaohui Xie. 2016. Co-occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks. *CoRR* abs/1603.07772.

Simon Zwarts and Mark Johnson. 2011. The impact of language models and loss functions on repair disfluency detection. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, USA, volume 1 of *HLT'11*, pages 703–711.

Simon Zwarts, Mark Johnson, and Robert Dale. 2010. Detecting speech repairs incrementally using a noisy channel approach. In *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, COLING '10, pages 1371–1378.