Karlsruhe Institute of Technology

Recurrent Neural Networks in Speech Disfluency Detection and Punctuation Prediction

Master's Thesis

Matthias Reisser

1538923

At the Department of Informatics
Interactive Systems Lab (ISL)
Institute of Anthropomatics and Robotics

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Prof. Dr. J. Marius Zöllner
Advisor:	Kevin Kilgour, Ph.D
Second advisor:	Eunah Cho

15th of November 2015

Abstract

With the increased performance of automatic speech recognition systems in recent years, applications that process spoken speech transcripts become increasingly relevant. These applications, such as automated machine translation systems, dialogue systems or information extraction systems, usually are trained on large amount of text corpora. Since acquiring, manually transcribing and annotating spoken language transcripts is prohibitively expensive, natural language processing systems are trained on existing text corpora of well-written texts. However, since spoken language transcripts, as they are generated by automatic speech recognition systems, are full of speech disfluencies and lack punctuation marks as well as sentence boundaries, there is a mismatch between the training corpora and the actual use case. In order to achieve high performance on spoken language transcripts, it is necessary to detect and remove these disfluencies, as well as insert punctuation marks prior to processing them.

The focus of this thesis therefore lies in addressing the tasks of disfluency detection and punctuation prediction on two data sets of spontaneous speech: The multi-party meeting data set (Cho, Niehues, & Waibel, 2014) and the switchboard data set of telephone conversations (Godfrey et al., 1992). The method of choice for tackling these classification problems are deep neural networks, which are flexible tools especially well suited for modelling textual data. In particular, this thesis studies the advantages of using recurrent neural networks, which are capable of modelling sequential data, over the use of standard feedforward neural networks.

Although disfluency detection and punctuation prediction are two different tasks, previous research has shown that modelling them jointly improves the performance of both. Therefore, in this thesis, the advantages of jointly learning neural network models in (recurrent) neural networks are investigated. It is shown that by combining both tasks, the models improve their generalization capabilities.

The meeting data set exhibits distinct properties that distinguishes it from the switchboard data set. Whereas the meeting data set contains transcripts of multi-party meetings, the switchboard data set is made up of telephone conversations between two speakers. Additionally, they have different annotation schemes and the meeting data set can be considered tiny in comparison to the amount of available data in the switchboard data set. Considering these differences, this thesis investigates possibilities of transferring knowledge from the larger switchboard data set to the disfluency detection and punctuation prediction tasks on the meeting data set. Strong performance improvements show the strength of these knowledge transfer approaches.

Contents

List of Figures

List of Tables

1. Introduction

> "This thesis is uhm I mean discusses neural networks uh recurrent neural networks in modelling speech disfluencies ..."

Spontaneously formulated speech differs greatly from well-written texts. This is because humans, in conversational speech, do not always produce grammatically correct phrases, abort previous sentences, start anew and are often interrupted by other speakers. Understanding these incorrect sentences apparently proves no challenge for human listeners. They have acquired a skill set that allows them to deal with disfluent utterances in conversational speech (Ferreira & Bailey, 2004) and can rely on their knowledge about the context of the conversation as well as infer meaning from intonation (Snedeker & Trueswell, 2003). However, designing automated systems that understand spoken language and are able to perform natural language processing (NLP) tasks such as automated translation, information extraction or dialogue systems, is a very challenging task. Machine learning approaches that aim at solving these NLP tasks are trained on large amounts of task-specific annotated texts. These text corpora are usually well written texts and are thus significantly different from spoken language. Firstly, inherent to spoken language, it contains disfluencies such as "uhm", "uh" or repetitions and interruptions. Secondly, automated speech recognition systems (ASR) produce a continuous stream of words, containing neither sentence boundaries nor punctuation marks. When training automated systems on textual data, these machines gain an understanding of human language that reflects the composition of the data they have been trained on. When using these systems on the output of an ASR system, performance is generally worse.

This mismatch between training data and spoken language data can be solved either by training the desired NLP system on spoken language transcripts instead of well written text corpora, or by transforming the spoken language transcripts into well written text prior to applying the NLP system. Since acquiring, transcribing and manually labelling spoken language corpora for each specific NLP task is expensive, transforming spoken language into well written text has been the focus of research, and also lies in the focus of this thesis.

In recent years, deep neural networks have gained increasing momentum in machine learning research and state-of-the-art results have been reported in diverse tasks ranging from image recognition to automated speech recognition and natural language processing. This thesis builds upon these successful ideas and presents several approaches to transforming spoken language transcripts into well written text. Since spoken language consists of a sequence of words, recurrent neural networks (RNNs) have been especially successful in many natural language modelling tasks. RNNs are especially designed to model sequential data and represent a powerful tool for sequence labelling tasks. The results presented in Chapter 6 of this thesis show that by transitioning from classic feed forward architectures to recurrent architectures, the performance on two data sets, the meeting data set (Cho, Niehues, & Waibel, 2014) and the switchboard data set (Godfrey et al., 1992), can be

improved.

Whilst the well-known switchboard corpus consists of transcripts of telephone conversation, the meeting data set contains transcripts of multi-party meetings. The latter poses additional challenges for detecting disfluencies and inserting punctuations because in conversations between several people, disfluencies, especially interruptions, occur more often. Furthermore, the meeting data set is considerably smaller than the switchboard data set. The amount of labelled data which is available plays a critical role in a model's ability to generalize to unseen data. In fact, apart from increased computational power, the success of deep neural networks in recent years is largely attributable to the increased availability of large amounts of data. Therefore, in this thesis, two techniques for improving the generalization capabilities of models for the meeting data set are investigated: Multi-task learning and transfer learning.

Multi-task learning embraces the idea that by jointly learning two related tasks concurrently in the same model, one can improve on the task of interest. By *reading* two labels for each input instead of only one, the parameters that are shared between the tasks are more robust and generalization is improved. In previous research efforts on disfluency detection and punctuation prediction, both tasks are combined instead of considering them separately and in sequence. The effect of leveraging the relation between both tasks by learning them jointly in the context of neural networks will be presented in Chapter 7.

Although there are differences between the switchboard data set and the meeting data set, both tasks intuitively share common features. Transfer learning aims to exploit these common factors that are shared between different but adjacent data sets. In Chapter 8, the focus lies on improving the performance of the relatively small meeting data set by using knowledge obtained from the switchboard corpus to train a more robust model, which generalizes better than would be possible using the meeting data set alone.

The contribution of this work therefore lies in evaluating the benefits of introducing recurrent neural networks, evaluating the effect of jointly learning punctuations with disfluencies and the evaluation of different approaches to transfer learning. The remainder of this thesis is structured as follows: Chapter 2 gives an introduction into the tasks that are considered in this thesis and formulates them as a machine learning problem. Chapter 3 discusses related works on disfluency detection and punctuation prediction. The data sets used in this thesis, as well as the technical system used in training models on this data is presented in Chapter 4. This is followed by an introduction of classic multilayer perceptron neural networks and their extension into recurrent architectures in Chapter 5. Chapter 6 evaluates the effect of adding recurrent connections into neural networks on both data sets, while Chapter 7 focusses on multi-task learning. Finally, Chapter 8 discusses and evaluates the benefits of transfer learning, which is followed by a critical discussion of the work and possible directions for future research in Chapter 9.

2. Disfluency Detection and Punctuation Prediction

Humans excel at understanding the meaning contained in another person's spoken utterances, even though, in most cases, these utterances do not formulate grammatically correct sentences and are rich with repetitions and interruptions. However, for machines, the ability to process spoken language is very hard to achieve. For various applications in natural language processing, it is necessary for automated systems to understand spoken language and react to the user's verbal instructions. Today's state of the art speech recognition systems already achieve a very high accuracy in transcribing the speech signal into words (Soltau et al., 2014). With the increased performance of these ASR systems, subsequent processing of the spoken language transcript becomes more interesting and gains more attention. Applications which operate on spoken language transcripts include machine translation systems, information extraction systems, dialogue systems, summarization and named entity recognition. Ideally, an automated NLP system is able to detect the words that are most relevant to it, depending on its purpose.

In order for machines to perform well on natural language processing tasks, they need to possess an understanding of natural language. In recent years, machine learning techniques emerged as a very successful approach in enabling machines to solve problems in NLP, as opposed to hand-crafted rules. In order to teach a machine to understand language, machine learning algorithms process large amounts of textual data and build a model of what they *read*. In the context of machine translation, examples of such data might be pairs of sentences: One sentence is written in the source language, while the second sentence represents the translated version of it.

The amount of available data becomes even more relevant within the field of *deep learning*, a sub-field of machine learning that uses deep neural networks. The achievable performance of these neural network models largely depends on the amount of data they can be trained on. The more data a model has been trained on, the better it can generalize to unseen data points. Since recording, transcribing and annotating large amounts of spoken language is infeasible or, at the least, prohibitively expensive, NLP systems are usually trained on large corpora of well-written, well-structured data that is readily available. During training, these systems build a model of the corpus and thus understand a well-written and generally well-behaved version of natural language. However, since humans in conversational contexts normally do not formulate sentences similar to what one might find on Wikipedia or in newspaper articles, the performance of these systems suffers when applied to spoken language. Cho et al. (2013), for example, show the negative impact of spoken language as input to machine translation, while Jones et al. (2003) show its negative impact on the readability of the transcripts.

Due to the fact that building NLP systems on spoken language transcripts is very hard with a limited amount of data, it is necessary to solve the problem of misaligned training and test data by transforming spoken language into a format resembling written texts. From a system's perspective, words, as they are spoken, are initially recognized and transcribed

by an ASR system. Before passing this stream of words to a NLP system, in order to align the spoken language transcript with the training data, the elements of natural language that lead to reduced performance of the system need to be addressed.

The two main differences between the stream of words that is provided by the ASR system and the well-written training corpora are the lack of sentence boundaries and reliable punctuation marks in the transcribed stream of words (Cho et al., 2015), as well as the aforementioned disfluencies inherent to spontaneous speech. Further differences, especially with better readability of transcripts in mind, are correct capitalization of words and higher level formatting such as paragraphs and sections (Ostendorf et al., 2008). They are, however, not as relevant as punctuation prediction and disfluency detection and therefore out of the scope of this thesis.

A successful alignment of spoken language to the domain of written texts therefore enriches the ASR system's output with punctuation information and identifies speech disfluencies.

2.1. Disfluencies in Spontaneous Speech

Spontaneous speech contains several distinguishable disfluencies, depending on the situation in which the speaker is recorded. A well-prepared speech, such as, for example, the speeches in the lecture corpus discussed in (Cho, Niehues, & Waibel, 2014), contains less disfluencies, whereas a conversation between two or more participants is rich with disfluencies (see Section 4.1.2). Benesty (2008) differentiates spoken language sources along the axes "number of speakers" (one, two and many) and "speaking style" between formal and casual, placing meetings and telephone conversations on the more challenging end of the spectrum.

The terminology in disfluency detection has been coined by the work of E. E. Shriberg (1994). The definitions given in her dissertation have been refined in subsequently created data sets and customized to the targeted modelling of disfluencies. In order to give an overview over common disfluencies, this section introduces notions and terminology found in the annotation schemes of both data sets used in this thesis, based upon E. E. Shriberg (1994).

Disfluencies in the English language consist of *fillers* and *edits*. Fillers are simple disfluencies that mark an interruption in fluent speech. They carry no information about the ideas formulated in the sentence and usually appear independently of the semantic context in which they are embedded. There are two main groups of fillers: *Filled pauses* and *discourse markers*. Meteer et al. (1995) refer to fillers as non-sentence elements and state further differentiations for this type of disfluencies commonly found in spontaneous speech. Examples include coordinating conjunctions such as "but anyway, it's not" and explicit editing terms "you, I mean, he".

Filled pauses are very common form of fillers in spontaneous speech. They encompass words and utterances such as "uh", "uhm" or "oh", which are used to bridge pauses during expressing a train of thought. In general, the speaker is thinking about how to continue or

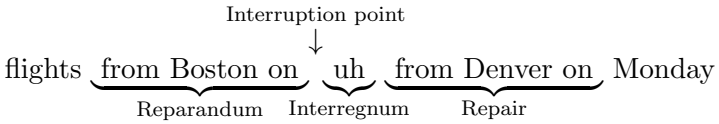


Figure 2.1.: Exemplary edit structure.

finish a sentence and fills the resulting pause with a sound. The sentence "Is it like, uh, oh, what's that called, it's, uh, correspondence school," contains filled pauses at several points, indicating the speaker was trying to find her words. All occurrences of filled pauses have in common that they carry no semantic information and thus can be considered irrelevant for most NLP applications.
Discourse markers are close to the previously discussed filled pauses. However, depending on the context, words such as "well", "you know" and "like" do carry semantic content and are therefore more difficult to distinguish. Consider the sentence "You are like a composer", which carries different meaning than "You are, like, a composer". Only in the second case should "like" be considered a discourse marker.

The second category of disfluencies in spoken language are *edits*. Edits are words the speaker changed her mind about and decided to replace them with a corrected version. A speaker edit consists of several regions: *Reparandum*, *interruption point*, *repair* and an optional *interregnum* (E. E. Shriberg, 1994). Reparanda, also referred to as *edit regions* or simply *edits* in some sources, encapsulate the words the speaker intends to replace with the words contained in the repair. In some cases (see Figure 2.1 for an example), the interruption point between the reparandum and the repair is followed by an interregnum. This interregnum usually consists of a discourse marker or filled pause.

Depending on the relationship between reparandum and repair, different kinds of edits can be differentiated. In cases where the repair is similar or identical to the reparandum, such as in "I think, I think this is good", the edit is called a *repetition*. In the meeting data set, which is introduced in Section 4.1, as well as in the Spontaneous Speech Reconstruction corpus (Fitzgerald & Jelinek, 2008), the elements inside the reparanda, corresponding to repetitions, are labeled as *rough copies*.
Sometimes, a speaker alters spoken words to a larger extend such as in "I know, I strongly believe that" or cancels a sentence completely, thus starting a new train of thought. The sentence "It's also, I used to live in" shows where the speaker restarts her sentence. Consequently the first uttered words carry no meaning in light of the subsequent words. In this case, instead of rough copies, these words are labelled *non copies* and are called *restarts* or *false starts* in some sources (Fitzgerald, Hall, & Jelinek, 2009; Cho et al., 2013).

The meeting data set introduced *interruptions* as another type of edit. Cho, Niehues, & Waibel (2014) felt that the increased amount of interruptions in a multi-speaker environment needs to be explicitly modelled. Interruptions are a special kind of edit, in which the reparandum stands alone and is not followed by a repair structure. Instead, another speaker continues the conversation.

The correct identification of edits in spoken language transcript lies in the focus of current research in disfluency detection (Johnson & Charniak, 2004; Hough & Schlangen, 2015; Qian & Liu, 2013), because detecting edits poses a greater challenge than detecting filler words. Filler words often consist of fixed phrases (Qian & Liu, 2013) and are not as heavily dependent on the surrounding context in which they are embedded in. Furthermore, filler words usually do not carry any meaning. Therefore, simply deleting filler words is sufficient for reconstructing speech fragments containing these elements. Deciding whether or not to label words as reparanda, however, depends on the correct identification of the repair structure. In case of restarts, whether or not words need to be labelled as such, depends on the meaning conveyed in the words following the restart. In the phrase "I don't like, his ideas do not interest me", correctly identifying "I don't like" as a restart is possible only by identifying that the words "his ideas do not interest me" mark the start of a new sentence, leaving the previous sentence incompleted.

More complex relations between repair and reparandum need to be identified in order to reconstruct a meaningful sentence in case information from the reparandum is carried over to the repair (Fitzgerald, Jelinek, & Frank, 2009). In order to transform the sentence "they like video games some kids do" into "some kids like video games", the relationship between "they" and "some kids" as well as the implicit connection between "do" and "video games" needs to be modelled. Correctly identifying the relation between elements inside an edit structure is, however, not part of this thesis. Here, the focus lies on correctly identifying the reparanda.

2.2. Sentence Segmentation and Punctuation Prediction

Sentence segmentation in the context of processing written text is usually called sentence boundary detection, whereas in spoken language processing, it is referred to as identifying *sentence-like units* (Liu et al., 2006). Sentence boundary detection concerns itself with identifying segments where a sentence boundary punctuation mark, i.e. a full-stop, question mark or exclamation mark should be (Batista & Mamede, 2011). In this setting, punctuation marks are present in the input data stream and the task is reduced to deciding whether a punctuation mark actually signifies a sentence boundary. This means, for example, deciding whether a full-stop is part of an abbreviation or whether a comma is used in an enumeration or signifies a clause. Sentence segmentation in the setting of spoken language processing, however, can not rely on the availability of punctuation marks in the input stream. The task of segmenting the input stream into sentence-like units is therefore closely related to the prediction of punctuation. A sentence-like unit may be equivalent to a full grammatical sentence but can also be a smaller unit. It is not unusual in conversational speech for a speaker to utter a semantically complete sentence in a grammatically incomplete form. These utterances may also be interrupted by another speaker or consist of only one-word sentences such as "right" or "oh really" (Meteer et al., 1995). Usually, at the boundary of a sentence-like unit, there is a missing punctuation mark, which needs to be correctly inserted.

Punctuation plays a critical role in understanding written texts. Without correct punctuation, a text is not only more difficult to read, missing punctuation also gives rise to potential ambiguities with respect to the meaning contained in it. Consider the popular example "Let's eat grandpa", which, depending on whether or not a comma is placed before the last word, suggest either to eat the speaker's grandfather, or invites the speaker's grandfather to the table. It is equally important to not place punctuation marks at the wrong location. In the phrase "When I sing well ladies feel sick"[1], placing the comma before the word "ladies", the phrase "When I sing well, ladies feel sick" might not convey what the speaker intended. Instead the speaker might have wanted to attribute the word "well" to "ladies": "When I sing, well ladies feel sick". These ambiguities do not only occur within sentences. Consider the phrase "Tim talks with Peter. You can be sure." with the alternative "Tim talks. With Peter, you can be sure.". This shows how changing the position of the sentence boundary yields two different interpretations of the speaker's intended meaning.

Apart from clarifying otherwise ambiguous word arrangements, punctuation marks can also be responsible for defining the meaning of a sentence. In case a question mark is inserted at the end of the phrase "I am right", it becomes a question and carries a different meaning in comparison to the case where an exclamation mark turns it into a statement. Especially for statistical machine translation, differentiating between these two cases becomes very relevant since questions and exclamatory statements might be formulated differently in the target language.

For spoken language, a wide variety of punctuation marks can be considered relevant for insertion into a transcript. Batista & Mamede (2011) mention full-stop, comma, question mark, exclamation mark, quotation mark, colon, semicolon and quotation marks. However, since the placement of most of these punctuation marks is determined by the human annotator's interpretation or occur comparably rarely and are therefore hard to predict, research has focussed on working with full-stop, comma and, in some cases, question marks.

In many languages, humans intuitively segment a sentence and distinguish between the possible alternatives by incorporating knowledge of the context and by listening to how a sentence was spoken. It therefore follows that, in order for an automated system to distinguish between grammatically correct hypotheses, the use of prosodic (acoustic) features can be helpful (E. Shriberg et al., 1998). When asking a question, a speaker usually rises the tone of her voice. In contrast, expressing something with a forceful tone suggests an exclamatory sentence. Similarly, sentence boundary detection benefits from using prosodic features. Especially the duration of pauses between words seems to be indicative of sentence boundaries (Rao et al., 2007b). In this thesis, however, the focus lies on only using lexical features in the prediction of disfluencies as well as sentence segmentation and punctuation prediction. The inclusion of prosodic features will be part of future research.

Since both, punctuation prediction and disfluency detection are important steps in reconstructing spontaneous speech, traditional preprocessing steps in an NLP pipeline have to

[1] http://www.future-perfect.co.uk/grammar-tip/fun-with-punctuation/, 31.October 2015

decide which task to perform first. In either case, by performing the tasks in sequence, errors from one task can be propagated into the subsequent task, making its performance highly dependent on the first task. W. Wang et al. (2010) performed extensive empirical studies on the two tasks' relationship. They find that jointly modelling both yields better performance than executing each of the respective tasks in sequence.

Jointly learning two related task is, in the machine learning literature, referred to as *multi-task* learning. The underlying hypothesis to this approach is that there are explanatory factors in the data that are useful for both tasks. Therefore there exists some intermediate representation of the data for both tasks that can be sensibly shared between them. Each shared parameter in such a jointly learned model is used for solving both tasks at the same time. By training these parameters jointly, each shared parameter thus sees proportionally more training examples than in the single-task case, granting more statistical strength and leading to better generalization of the model (Bengio et al., 2016). Jointly modelling NLP tasks has proven to be successful for punctuation prediction and disfluency detection for example by W. Wang et al. (2010) and Cho et al. (2015). Therefore, in this thesis, the benefits from jointly modelling both tasks is investigated in the context of neural networks (see Chapter 7).

2.3. Problem Formulation

Although both tasks considered in this thesis are examples for sequence labelling tasks, when talking about the correct identification of disfluencies and the correct placement of punctuations, in the literature the terms *detection* and *prediction* are commonly used. The difference in naming results from the fact that disfluencies are already present in the text and need to be *detected*, while punctuations are not present and need to be *predicted* (X. Wang, 2015). In order to formally introduce the task of sequence tagging, the situation in which the individual observations are assumed to be independent from each other is formulated first. Subsequently, the formulation is generalized to encompass the modelling of dependencies within the observations of a sequence.

Pattern classification

In a standard non-sequence supervised learning setting, there exists a input space $\mathcal{X} = \mathbb{R}^M$ of M-dimensional input vectors and a target space $\mathcal{Y}$, which, in the case of the classification tasks considered in this thesis, is equivalent to the finite set of possible classes $C = \{C_1, \dots C_K\}$. *Learning* in this context means asking a computer to learn from given data the parameter set θ of a model $f_\theta : \mathcal{X} \to \mathcal{Y}$, which specifies the label $C_k \in C$ (or multiple labels) that belongs to a given input $x \in \mathcal{X}$. The learning procedure stops when the parameters θ are optimal with respect to a task-specific error-function or performance measure. The data consists of input-target pairs (x, y), where each pair is assumed to be drawn independently from the same distribution $\mathcal{D}_{\mathcal{X}\times\mathcal{Y}}$ over the input and target space. The data is usually partitioned into three disjoint sets, the *training set* S, the *validation set* and the *test set*. Only the data in the training set is used to fit the model parameters. In order to compare the performance of different hyperparameter choices, a model's performance is evaluated on the validation set. It is important to evaluate a model on data that

has not been used in training the model in order to correctly evaluate the generalization capabilities of the model. Only in maintaining this separation can overfitting (see Section 5.3) be correctly recognized and the comparison between different models' performances is based on the true generalization capabilities of each model. Howecer, since during the process of selecting the best combination of hyperparameters for a model, the performance on the validation set is used to select the best model, a systematic bias towards models that perform well on the validation set is introduced. The best model on the validation set might thus not be the model that has the highest generalization performance and instead be the model that performs best on the validation set. Therefore, the performance of the final model that is reported for the designated task is evaluated on the separate test set, which has not been used in the process until this point.

Although ultimately, for a given input x, the corresponding target label C_k is of interest, most supervised learning algorithms, such as the approach presented in this thesis, model a conditional probability for each possible label $p(C_k|x)$, $C_k \in C$. This has the advantage that, in a setting where to each input x exactly one output label can be associated, a hypothesis $h(x)$ can be computed easily:

$$h(x) = \arg\max_{C_k} p(C_k|x). \tag{2.1}$$

In cases where multiple labels can be associated to an input x, a label is associated with an input if its probability is higher than a custom threshold $t(x)$. This leads to:

$$h(x) = \{\, C_k \,|\, p(C_k|x) > t(x),\ C_k \in C\}. \tag{2.2}$$

The binary classification problem is a special case of (2.1), in which $C = \{C_1, C_2\}$ and $p(C_1|x) = 1 - p(C_2|x)$.

The most common principle by which a model is fitted to the data is by the maximum likelihood (ML) principle (Bengio et al., 2016). In this setting, the estimator θ_{ML} for the optimal parameters θ^* maximises the likelihood of θ given the training data $\mathcal{L}(\theta|S) = p(S|\theta)$:

$$\theta_{ML} = \arg\max_{\theta} p(S|\theta) = \arg\max_{\theta} \prod_{(x,y)\in S} p(y|x,\theta). \tag{2.3}$$

In order to find θ_{ML}, the maximization of the likelihood is formulated as the minimization of a cost function O. The standard approach is to transform the likelihood into the negative log-likelihood and minimize the resulting term:

$$O(\theta; S) = -\ln \prod_{(x,y)\in S} p(y|x,\theta) = -\sum_{(x,y)\in S} \ln p(y|x,\theta). \tag{2.4}$$

Sequence labelling

The previous discussion introduced the concept of pattern classification. Since the tasks of disfluency detection and punctuation prediction are examples of sequence labelling tasks, the previously introduced formulation is now extended to model the tasks at hand. This thesis follows the notation style in Graves et al. (2012) in order to formulate the following. Whereas in the pattern classification case, the input space $\mathcal{X}$ was equivalent to $\mathbb{R}^{\mathbb{M}}$ and $\mathcal{Y}$ was equivalent to C, when considering labelling sequences, the input space is equivalent to the set of all sequences of real valued M-dimensional vectors $(\mathbb{R}^M)^*$, which is being mapped to a sequence of labels C^*. In order to denote the differences between a M-dimensional real-valued vector input x with a corresponding label y and a sequence of M-dimensional real-valued vectors $\mathbf{x}$ and the corresponding sequence of labels $\mathbf{y}$, the latter are notated in bold letters. Although both sequences $\mathbf{x}$ and $\mathbf{y}$ in an element $(\mathbf{x}, \mathbf{y}) \in S$ may be of arbitrary length, the length of the input sequence and the length of the sequence of labels are assumed to be equivalent in modelling disfluency detection and punctuation prediction. Models that perform the task of sequence labelling consequently model $p(\mathbf{y}|\mathbf{x})$ and compute a hypothesis $h(\mathbf{x})$.

In modelling the word sequence explicitly as a sequence, it is possible to capture and model the dependencies between words in a sentence. As has been explained in Section 2.1, this capability is beneficial in modelling the relationships of disfluencies with their surrounding context.

3. Related Work

3.1. Disfluency Detection

The body of research on disfluency detection is hard to analyse with respect to performance on the task. This is because different research groups use different corpora with respective different annotation schemes and evaluation metrics. Additionally, disfluency detection can be viewed as an extension to the speech recognition problem or as a processing step following automatic speech recognition. Consequently, some research efforts rely only on acoustic/prosodic features, while others use lexical information alone or a combination of both. In this analysis of related work, the focus therefore lies on modelling techniques.

In early work on disfluency detection, Heeman & Allen (1999) introduce a statistical language model for speech recognition, which models speech disfluency detection along with further tasks such as part of speech (POS) tagging and the identification of discourse markers, speech repairs and intonational phrases. In addressing these tasks jointly, they receive better results compared to modelling each task separately.

Charniak & Johnson (2001) employ a boosting model of linear classifiers to identify edit regions in the switchboard corpus. Johnson & Charniak (2004) improve upon their results by introducing a noisy channel model based on a Tree-Adjoining-Grammar. A syntactic parser-based language model as introduced by Charniak & Johnson (2001) is used to rescore the output of the noisy channel model. In a noisy channel approach, it is assumed that the fluent data F has passed through a noise-inducing channel which produces the disfluent data D. In order to retrieve the mostly likely input string $\hat{F} = \arg\max_F P(F|D)$, the right-hand side is reformulated according to Bayes' Theorem: $\hat{F} = \arg\max_F P(D|F)P(F)$. Here, $P(F)$ represents a language model over the fluent data and $P(D|F)$ represents the noisy-channel model that needs to be learned from the data. A further noisy channel approach is described in Maskey et al. (2006), in which the authors formulate the task as a phrase-level statistical machine translation problem.

Honal & Schultz (2003) present a noisy channel approach, which is combined with N-gram language model scores to retrieve the best hypothesis. In Rao et al. (2007a), this model has been analysed with respect to its performance in improving spoken language translation. They achieve up to 8% improvement on BLEU scores by removing disfluencies prior to translation. Similar to Rao et al. (2007a), W. Wang et al. (2010) investigate the effect of cleansing the input to a machine translation system prior to translation. They employ a combination of statistical models using Hidden Markov Models, Conditional Random Fields (CRF) and heuristic rule-based systems.

Fitzgerald, Hall, & Jelinek (2009) employ a CRF approach, using several features as input for their model. Beside language model features and lexical features, they use the output of the system proposed by Johnson & Charniak (2004). Zwarts & Johnson (2011) evaluate the model presented by Johnson & Charniak (2004) with respect to the effect of the language model choice on the disfluency prediction task. They experiment with different

language models built from external sources and show that adding large external sources improves the reported F-measure (see Section 4.3) on held-out data. Whether or not the external sources are transcripts of spoken language or written texts does not influence overall performance significantly. Furthermore, the authors experimented with reranking using a minimal expected F-measure loss function, achieving improved results compared to using the negative log conditional likelihood of the training data (see Section 2.3).

Qian & Liu (2013) employ Max-margin Markov Networks with the goal to better balance precision and recall in their system. Their model consists of three consecutive steps, which is an implementation of a stacked-learning approach. In passing over the cleaned-up data again, they can improve their results.
Zwarts et al. (2010) extend the noisy channel model approach presented by Johnson & Charniak (2004) with the notion of incremental disfluency detection. Instead of processing an utterance as a whole in order to provide labels for each element in the utterance, an incremental model marks disfluencies as soon as possible.
In Hough & Purver (2014), the authors distinguish between strong incremental detection and left-to-right operating systems. While the latter also process an utterance left-to-right, only the former are designed and evaluated for their incremental performance. Honnibal & Johnson (2014), for example, introduce structured averaged perceptrons for training coupled with a beam search decoder. According to Hough & Purver (2014), the hypotheses in Honnibal & Johnson (2014) tend to *jitter*, thus resulting in worse incremental detection performance. In Hough & Schlangen (2015), the authors employ Recurrent Neural Networks in order to model the disfluency detection task. Along with a Markov Model decoder, they report results comparable to previous results by Hough & Purver (2014).

In addition to lexical features, prosodic cues extracted from the speech signal have proven useful in detecting speech disfluencies. Savova & Bachenko (2003), for example, performed an extensive study of prosodic cues for four types of disfluencies. E. Shriberg et al. (1997) use decision trees to model prosodic cues assuming known word boundaries. Stolcke et al. (1998) model speech disfluencies and sentence boundaries as hidden events at inter-word boundaries. They combine decision trees as in by E. Shriberg et al. (1997) for the modelling of prosodic cues with a N-gram language model to predict at most one of the hidden events at the word boundaries. Liu et al. (2006) compare Hidden Markov Models, Maximum Entropy Models and Conditional Random Field approaches for modelling disfluencies as well as sentence boundaries, using lexical as well as prosodic features.

3.2. Sentence Segmentation and Punctuation Prediction

Reconstructing punctuation and sentence boundaries into spoken language has been investigated using a variety of different approaches. As in the case of disfluency detection, the different approaches can not sensibly compared in terms of performance because different corpora, different features (prosodic and/or lexical) and different evaluation criteria were employed over the years. This review therefore concentrates on the techniques applied to the task.

Hidden Markov Models (HMM) have been used in a variety of combinations in order to predict different structural events such as sentence boundaries and punctuation marks. In an HMM, words represent the observations and the hidden states model pairs of words and interword events. The transition probabilities between hidden states are estimated using a hidden event N-gram language model, which models the joint distribution of the sequence of words and events. Stolcke et al. (1998) introduced this technique, modelling, amongst others, disfluencies, punctuation marks and sentence boundaries and combining lexical with prosodic features. The disadvantage in using HMM lies in that instead of maximising the posterior probability $p(e|W, F)$ of an event e given the observed words W and features F, they maximise the joint probability $p(W, E)$ of both, words and hidden events. As such, these generative model are not ideal for classification tasks.

Christensen et al. (2001) investigate finite state machines and a neural network based approach for punctuation prediction. In their work, they focus on predicting full stop, comma and question marks. They compare the effectiveness of lexical features and prosodic features in both models, showing that from the range of possible prosodic features, pause duration is the most indicative for punctuations.

Huang & Zweig (2002) introduce a maximum entropy model which directly models the posterior probability and predicts comma, period and question mark on the switchboard corpus. Since in a maximum entropy model the choice of features is essential, the authors report results on a variety of prosodic and lexical features. They find that their lexical feature based model outperforms the prosodic feature based model and report that distinguishing between punctuation and question mark is prone to errors. Maximum entropy models have the disadvantage that they cannot capture long-term dependencies that may be required, especially when predicting question marks. Lu & Ng (2010a) perform an extensive comparison between HMM and maximum entropy models. They find that the HMM model is stronger in incorporating prosodic features than the maximum entropy model and vice versa for lexical features.

In treating punctuation prediction and sentence boundary detection as sequence tagging task, they have been approached using CRF. Liu et al. (2005) employ this technique for sentence boundary prediction, showing that it can outperform both, HMM and maximum entropy approaches on this task using both, acoustic and lexical features. It combines the strength of both, by incorporating sequential information like HMM (constrained by the N-gram language model) and being trainable in a discriminative fashion like the maximum entropy approach. Lu & Ng (2010b) use dynamic random fields for punctuation prediction using only lexical features, outperforming the HMM based approach.

Liu et al. (2006) compare HMM, maximum entropy and CRF approaches for sentence boundary detection with respect to their differences in performance with different features. They showed that adding additional lexical features to words improves performance compared to only using words.

The authors in Xu et al. (2014) combine a deep neural network with a CRF approach. Prosodic features are embedded using a deep neural network, which is then integrated with

lexical features and used as input for the CRF in order to model the posterior distribution for sentence boundaries. Their approach outperforms previous approaches modelling prosodic features with decision trees by a wide margin.

A further neural network based method has recently been presented by Tilk & Alumäe (2015). They use an LSTM recurrent neural network (see Section 6.3) in a two-stage approach. First, they train a LSTM language model on a large corpus of text in the Estonian language. The features represented by this model are then used, along with pause durations, as features in a second LSTM network, which is then fine-tuned to predict full-stops and commas.

4. Data and System

In this thesis, experiments were conducted on two data sets: The switchboard disfluency annotated corpus presented by Godfrey et al. (1992) and a multi-party meeting data set first discussed by Cho, Niehues, & Waibel (2014). This chapter reviews the characteristics of both data sets, discusses preprocessing of the data and the labels used in detecting disfluencies and predicting punctuations.

4.1. Meeting Data Set

In a globally connected society, meetings between people from different countries speaking different languages is certainly no rarity. Whether these people sit around the same table during the meeting or talk via video conferences, their spoken sentences are of interest for several NLP applications. Possible scenarios in a multi-party meeting are the generation of a well-written transcript, the extraction of relevant information or the automated translation between participants not familiar with the same language.

Severe challenges for ASR systems arise in the setting of multi-party meetings. Usually, participants use suboptimal configurations of microphones and there is, depending on the formality of the meeting, a large chance that speakers overlap. The reason for the latter is that humans, when absorbed in a discussion, have a deep understanding of the context of the meeting and can therefore successfully predict and finish another speaker's sentence. In cases where speakers do not have individual microphones, it is very challenging to decide who spoke when, especially when spoken sentences overlap (Ostendorf et al., 2008). The word error rate in such situations is therefore higher, making the application of downstream NLP tasks even more challenging.

Assuming an ASR system is capable of capturing the words spoken during a meeting with a sufficiently low error rate, NLP systems are faced with transcripts that are rich in disfluencies. Cho, Niehues, & Waibel (2014) compare the amount of disfluencies in this data set with a corpus consisting of transcribed university lectures and find that sentences spoken in the meeting data set contain roughly twice the amount of disfluencies. It is therefore of paramount importance to reconstruct meeting transcripts prior to the designated NLP application and to properly identify disfluencies as well as insert punctuation and sentence boundary information.

The data set at hand was designed with the purpose to advance the state of the art in multi-party meeting translation. It consists of recordings of eight meeting sessions held in English. Each meeting involves five to twelve different participants witch are both, native and non-native English speakers. The meeting sessions were transcribed and manually annotated. For future research purposes, the annotators created a grammatically corrected version in addition to labelling disfluencies. They were allowed to reorder, remove and even add words into this corrected version if they felt it to be necessary. By doing so, the annotators created, for each spoken sentence, a clean and fluent reference version without modifying the meaning of the original sentence. See Table 4.1 for an example

original:	I have a comment with error analysis as well as uhm
reference:	I have a comment concerning error analysis, too.

Table 4.1.: Example for an original, disfluent sentence and its cleaned reference version.

of an original sentence and its reference version. Although it is not considered in this thesis, these reference versions of the sentences can be regarded as the *gold standard* for a disfluency cleaned and punctuation enriched spoken language transcript.

4.1.1. Disfluency Annotation Scheme

The annotation scheme for this data set was designed in the style of the work by Johnson & Charniak (2004) and has been used before by Fitzgerald, Hall, & Jelinek (2009) and Cho et al. (2013). Using this scheme, disfluencies are categorized into `filler`, `(rough) copy` and `non-copy`. The latter two categories are more fine-grained classifications of what in the notation of E. E. Shriberg (1994) is called the reparandum. The labels equivalent to the interregnum and repair are missing in this scheme. After reviewing the special characteristics of the meeting data set, the authors in (Cho, Niehues, & Waibel, 2014) decided to extend this scheme by introducing a fourth category, `interruption`, which reflects the fact that in a multi-party meeting setting, speakers interrupt each other frequently.

A word sequence is labelled as a `(rough) copy` if it is followed by an exact or very close copy. In the example "+/it 's/+ it 's also using" the word sequence "it 's" has been surrounded with the symbolds +//+, marking the words as a `(rough) copy`. `Non copy` sequences are used to mark word sequences that were aborted by the speaker. The ensuing word sequence may either be a moderately altered version of the `non copy` sequence and carry the same intended meaning or it may introduce a new idea. In the sequence "-/it really/- most of them", the words "it really" are marked as a `non copy` using the symbols -//-. Words that are labelled as `interruption` are surrounded with the symbols #//#, such as in the following example:

Speaker A:	I 'll have to do that this afternoon #/and beginning of next/#
Speaker B:	Okay, does that mean you can start writing rules?

Table 4.2.: Example for a conversation containing an `interruption`.

The annotators decided that the second part of the first speaker's sentence has been interrupted by the second speaker and therefore marked it as an `interruption`. `Fillers` are marked by enclosing them within the symbols <> such as in the phrase "<uh> let 's see".

4.1.2. Statistics

Following the argumentation in Section 2.3, the meeting corpus was split into a training set, validation set and test set. The eight available meeting sessions were split up in three parts as depicted in Table 4.3. Joined together, the meeting transcripts in the training

set amount to roughly $4k$ sentences consisting of roughly $39k$ words. The validation set consists of circa 600 sentences and roughly $8.5k$ words. With ca $1.7k$ sentences and $15k$ words in the test set, the available data was split into training, validation and test set at a ratio of 62 : 14 : 24. This split follows the procedure by Cho et al. (2015). It is worthy to note that the average sentence length is below ten words per sentence, which is short compared to written text. In counting disfluencies, only words were taken into considerations and punctuations were omitted. For example, the annotated sentence "<well ,> I don't recall if", was counted as containing only one filler word. As it can be seen in Table 4.3, roughly 20% of the words in the transcripts are marked as disfluent. This is a remarkably high percentage compared to the previously mentioned lecture corpus (Cho, Fünfer, et al., 2014) (11%) and comparable to the Switchboard Corpus, in which 21% of the words are disfluent.

		Training		Validation		Test	
	Sessions	m002, m003, m005, m006b, m008		m007, m001a, m006a		m004	
	Sentences	3,987		610		1,772	
Disfluencies	`filler`	2,658	6.8%	566	6.8%	999	6.7%
	`interruption`	2,427	3.5%	481	5.7%	1,165	7.8%
	`non copy`	846	2.0%	153	1.8%	268	1.8%
	`(rough) copy`	1,443	5.7%	258	3.0%	885	5.9%
	`fluent`	31,469	81.0%	7,009	82.8%	11,610	77.8%
	Total	38,843	100.0%	8,469	100.0%	14,927	100,0%
Punctuation	`full stop`	3,354	8.6%	491	5.8%	1576	10.6%
	`comma`	2,357	6.1%	406	4.8%	1,061	7.1%
	`question mark`	632	1.6%	118	1.4%	195	1.3%
	`no punctuation`	32,500	83.7%	7,454	88.0%	12,095	81.0%
	Total	38,843	100.0%	8,469	100.0%	14,927	100,0%

Table 4.3.: Meeting data set statistics.

4.2. Switchboard Data Set

The switchboard data set (Godfrey et al., 1992) consists of several hundred telephone conversations between two people, each about a predetermined topic. In this thesis, the disfluency tagged transcripts as provided by the Linguistic Data Consortium were used. More specifically, all the `*.dff` files from the Penn Treebank III release (Marcus et al., 1999) in the directory `/dysfl/dff/swbd/` were used for training, validation and testing.

As a preprocessing step, all non-disfluency annotated words (such as for noise or unintelligible words) were ignored. In case a whole utterance of a speaker consisted only of these non-disfluency annotated words, the previous and ensuing utterance by the other speaker were joined into one utterance. Additionally, all partial words (words marked by a hyphen) were removed as in other research on this data set (Honnibal & Johnson, 2014; Zwarts & Johnson, 2011; Johnson & Charniak, 2004) because it more realistically reflects a NLP setting (Zwarts & Johnson, 2011). Partial words are considered to be disfluencies

at sub-word level and their identification and elimination is the task of the ASR system (X. Wang, 2015).

4.2.1. Disfluency Annotation Scheme

The switchboard disfluency annotation scheme (Meteer et al., 1995) follows E. E. Shriberg (1994) in the annotation of *edits* and differentiates between a variety of classes for *fillers*.

Fillers are annotated with { } as shown in the exemplary sentence "{D Well, } I have plenty". The type of filler (in this case a discourse marker) is encoded with a single capital letter after the opening bracket. There are five classes for fillers in the switchboard corpus. `filled pause` such as "uh" or "uhm" are encoded by the letter "F". `discourse marker` ("you know", "well"), as shown before, are encoded by the letter "D". The capital letter "E" denotes `explicit editing` terms ("I mean", "sorry") and the letter "C" encodes `coordinating conjunction`, such as "and then" or "but". Finally, the letter "A" encodes the category `aside`, which are word sequences that interrupt the flow of the sentence. Since the last category is very rare, it has been ignored in this thesis.

The sentence "[when I, + {F uh, } before I] got married" contains a complete reparandum, interregnum, repair structure inside the squared brackets. Every word in-between the opening bracket and the plus sign is considered as part of the reparandum and therefore tagged as `RM` in the following. The interregnum, following the plus sign, is always encoded as one of the filler categories mentioned above. However, the interregnum is not mandatory, therefore the plus sign can be followed directly by the repair. As a matter of fact, the repair also is not necessary, in case the speaker does not repeat herself or provides a substitution but instead deletes the spoken words. Every word, either following the interregnum, or directly following the plus sign, until the closing square bracket is considered the repair. Since identifying the interregnum is not of interest, there exists no label for it, except for its label attributed to its role as a filler. Equally, the words inside the repair are not modelled in this thesis and do not have a designated label.

The switchboard corpus contains several examples of nested edit structures as shown in in the following example:

```
label:    RM    RM      0  0  0   0   0
orig.: [ [  I  +  I  ] + I don 't want it ]
```

This example shows the chaining of two edit structures. Since the primary interest lies in identifying reparanda in order to produce a clean, fluent version of the transcript, in situations such as these, all elements before the last interruption point are treated as reparandum, ignoring its inner structure.

```
label:  RM RM          F      RM     0     0     0 0 0
orig.: [  I  liked, + {F uh, } [  I  , + I ] liked ] it a lot
```

In the second example, there is an edit structure in the repair of an edit structure. Similar to the previous example, this situation is disentangled by ignoring its complications. The fact that the words to the right of the first plus sign (until the second closing square

bracket) is part of a repair structure, is ignored and the repeated "I" is treated like a normal repair structure.

As opposed to the meeting corpus, there may be more then one label associated to a word. In the following example, the word "uh" is part of a reparandum, while, at the same time, it needs to be labelled as a filled pause.

label: 0 RM RM F/RM RM 0 0 0
orig.: it [is [not , + {F uh , } not ,] + is not] necessary

Although ultimately one might not care which category of disfluency will be detected, removing one label would hinder the model in learning to predict correctly. If, for example, the word "uh" in the upper example were to be only classified as part of a reparandum, the model would be punished for (correctly) predicting a filled pause. By enforcing the model to decide for one of the two plausible labels, the model would have to learn this more complicated situation in which filled pauses must not be identified as such inside reparanda. This corresponds to a more complicated partitioning of the input space which is making the task harder, considering the small amount of these situations that is available for training. Consequently, learning disfluency detection on the switchboard data set requires modelling the task as a multi-label learning problem.

In the meeting data set, a speaker's turn is either considered complete or marked as an interruption if the speaker's last sentence was not finished. In the switchboard data set, however, the annotation is such that a speaker may finish a turn after the other speaker's interruption. In the following example, the interruption by speaker B is marked with the symbols "--".

A: we did get the wedge cut out by building some kind of --
B: A cradle for it.
A: -- a cradle for it.
(Slash-unit annotation (Section 4.2.2) has been removed)

In order to predict these disfluencies, they are included as a label to the last word before the interruptions. For the previous example, the first sentence is encoded as follows:

0 0 0 0 0 0 0 0 0 0 --
we did get the wedge cut out by building some kind of

Depending on whether an ASR system provides turn information or not, disentangling pieces of conversation that are interrupted in order to correctly infer which words are disfluent, can be a challenging task.

4.2.2. Slash-unit Annotation

Additionally to punctuation marks, the switchboard corpus is annotated with *slash-units*. Slash-units correspond to the previously discussed *sentence-like units* and are marked with the sign "/" at their boundaries. Participants in telephone conversations do not consistently speak in grammatically correct, full sentences, but nevertheless convey meaning in what

they say. The annotators were therefore encouraged to mark word sequences as slash-units if they felt that it was complete. This includes correct sentences, but may also be smaller units, such as continuers and short sentences as in the following dialogue:

A: Yeah, / Right. /
B: A cradle for it. /
A: Uh-huh. / I guess he 's young. /

As is evident by the above example, the beginning of a speaker's turn marks the beginning of a slash-unit. It is, however, also possible for a speaker's slash-unit to continue after the other speaker's turn, if the slash-unit was left uncompleted. In the following example, the speaker was interrupted and finishes her turn after the interruption:

A: we did get the wedge cut out by building some kind of --
B: A cradle for it. /
A: -- a cradle for it. /

For cases, in which a slash-unit is left incomplete by a speaker, the annotators marked the word sequence as an incomplete slash-unit with the symbols "`-/`". In these situations, the speaker either starts a new sentence or does not finish a sentence after being interrupted, as in the following example.

A: They 're always necessary. / If you put enough patience into, -/
B: Yeah, / Just be consistent and diligent --
A: Uh-huh. /
B: -- with it / {C and,} {F um, } -/

Speaker A did not carry on her second sentence after speaker B's first turn. Therefore, the word sequence between the last slash-unit label (it may be a complete slash-unit or an incomplete slash-unit) and the ending of this slash unit is considered incomplete.

For predicting (incomplete) slash-units, the last word of a slash unit carries the corresponding label. The first sentence of the last example is therefore encoded as follows:

0	0	0	/	0	0	0	0	0	-/
They	're	always	necessary.	If	you	put	enough	patience	into,

4.2.3. Statistics

The switchboard data set was divided into a training, validation and test set in the same partitioning as by Johnson & Charniak (2004) and Qian & Liu (2013): The training set consists of the conversations named `sw[23]*.dff`, the validation set is built out of all files named `sw4[5-9]*.dff` and the test set contains all files named `sw4[0-1]*.dff`. With a total of 1.5 million words, the switchboard corpus is about 24 times as large as the meeting data set (see Table 4.4). As in the meeting data set, only words were taken into consideration for counting the number of difluencies. Because the number of aside tokens is very small, they have been omitted, just as exclamation marks have been removed. The

total amount of labels, also counting fluent words, exceeds 100% on each data set because words may be labelled with more than one category. The same holds true for punctuations. In most cases where there is a full stop, a slash-unit boundary is encountered. Also counting those words that are not followed by a punctuation mark or slash-unit boundary, the total amount of punctuation labels (also counting slash-units) can exceed 100%.

From the perspective of some applications, such as machine translation, the total amount of words that need to be removed in order to receive a cleaned version of the transcripts is of interest. For counting the total amount of disfluent words, all words inside an incomplete slash-unit were counted as being disfluent. Every words following a slash-unit boundary up to the incomplete slash-unit label `-/` are thus treated as words to be removed. In case a speaker's incomplete slash-unit is interrupted by the other speaker, only the words following the interruption are counted as being part of the incomplete slash-unit. Although interruptions are listed and treated as disfluencies in Table 4.4, it is assumed that they will not be removed. Incomplete slash-units, however, although they are listed as punctuations due to purpose in sentence segmentation, need to be removed for a fluent speech transcripts.

		Training		Validation		Test	
	Transcripts	`sw[23]*.dff`		`sw4[5-9]*.dff`		`sw4[0-1]*.dff`	
	Words	1,314,636		112,618		65,484	
	Disfluent words total	274,056	20.58%	25,191	22.37 %	14,449	22.06%
Disfluencies	F: `filled pause`	39,680	3.0%	4,050	3.6%	2347	3.6%
	E: `explicit editing`	6452	0.5%	637	0.6%	370	0.6%
	D: `discourse marker`	42,550	3.2%	3,980	3.5%	2,472	3.8%
	C: `coordinating conjunction`	50,131	3.8%	4,240	3.8%	2,158	3.3%
	RM: `reparandum`	67,879	5.2%	6,642	5.9%	3,789	5.8%
	--: `interruption`	10,020	0.8%	1,196	1.1%	860	1.3%
	0: `no disfluency`	1,101,762	83.8%	92,505	82.1%	53,730	82.1%
	Disfluency tokens total	1,318,474	100.3%	113,250	100.6%	65,726	100,4%
Punctuation	`full stop`	100,965	7.7%	8,956	8.0%	4,649	7,1%
	`comma`	204,591	15.6%	16,069	14.3%	10,028	15.3%
	`question mark`	6,977	0.5%	483	0.4%	415	0.6%
	/: `slash-unit`	156,268	11.9%	11.849	10.5%	7,062	10.8%
	-/: `incomplete slash-unit`	19,243	1.5%	1,581	1.4%	961	1.5%
	`no punctuation`	978,362	74.4%	84,313	74.9%	49,312	75.3%
	Punctuation tokens total	1,466,406	111.5%	123,251	109.4%	72,427	110,6%

Table 4.4.: Switchboard data set statistics.

4.3. Performance Evaluation

When reporting the performance on a classification task, the minimum costs that is achievable in minimizing the costfunction of the classifier (see Section 2.3) may not necessarily reflect the classifier's performance with respect to its desired behaviour. Therefore, the performance measure is usually specific to the task that is learned by the classifier. Although improving on the classifier's cost function should correspond to an improvement in its performance measure, both do not necessarily need to be the same. A common

measure for a classifier's performance is *accuracy*. Accuracy simply measures the percentage of examples that the system classified correctly, which is the same as measuring one minus the *error rate*. Accuracy is, however, not an appropriate performance measure for tasks in which the class label distribution over examples is highly skewed, such as in many NLP tasks and in disfluency detection and punctuation prediction in particular. A high overall accuracy might reflect the case in which the classifier achieves excellent results on the majority class while completely ignoring the performance on the minority classes. Ideally, a good classifier performs well on the minority classes while not deteriorating in performance on the majority class. For these reasons, for classification tasks involving imbalanced data sets, other performance measures, such as receiver operating characteristics and the F-measure based on precision and recall, are better suited (He et al., 2009). In the literature on NLP tasks, performance is usually reported in terms of precision, recall and the F-measure, which are therefore going to be the performance measures of choice for reporting the performance on disfluency detection and punctuation prediction in this thesis.

Binary classification

In a binary classification setting, the F-measure, which is also called the F_β-score, is defined as the weighted harmonic mean of precision (PR) and recall (RC):

$$FM_\beta = \frac{(1+\beta^2) \cdot PR \cdot RC}{\beta^2 \cdot PR \cdot RC}, \tag{4.1}$$

with precision and recall defined in terms of true positives (TR), false positives (FP) and false negatives (FN):

$$PR = \frac{TP}{TP+FP}, \tag{4.2}$$

$$RC = \frac{TP}{TP+FN}. \tag{4.3}$$

In an intuitive explanation, precision determines the exactness of the classifier, while precision measures how thorough the classifier is in identifying all positive examples. The F-measure balances precision and recall according to the parameter β, since there is usually a trade-off in optimizing for both. Whenever the term *F-measure* is used, it implies that FM_β with $\beta = 1$ is referred to.

Determining true positives, false positives and false negatives requires transforming the output of a probabilistic classifier, such as a neural network, into a definite hypothesis $h(x)$ about the class label that belongs to a given example. In the case of a binary classification task ($y \in \{C_1, C_2\}$) this is done by imposing a threshold on the probability $p(C_1|x)$ of example x belonging to class C_1 (which is equivalent to $1 - p(C_2|x)$)(see (2.2)).

With this, TP, FP and FN can be determined for, for example, the training set S:

$$TP = \sum_{(x,y)\in S} h(x) \cdot y, \tag{4.4}$$

$$FP = \sum_{(x,y)\in S} h(x) \cdot (1 - y), \tag{4.5}$$

$$FN = \sum_{(x,y)\in S} (1 - h(x)) \cdot y. \tag{4.6}$$

By increasing the threshold that determines $h(x)$, it is possible to directly influence the trade-off between precision and recall in a probabilistic classifier to a preferred ratio.

Multiclass classification

For disfluency detection and punctuation prediction it is not only of interest if a certain word is disfluent or not, or if is followed by a punctuation mark or not. Instead, the classifier's performance on individual disfluency classes and individual punctuation marks is relevant. Detecting disfluencies and punctuation marks in the meeting data set therefore belong to the set multiclass classification tasks. In this setting, an example x is labelled with exactly one of $K > 2$ labels ($y \in C = \{C_1, \ldots, C_K\}$). In case of the meeting data set, a word carries one of the labels {`filler`, `interruption`, `non copy`, `(rough) copy`, `no disfluency`} by encoding a fluent word as carrying the label `no disfluency`. In this multiclass classification setting, there are two possible ways of averaging across labels for a definition of the F-measure (Sokolova & Lapalme, 2009). Both are based on a per-label definition of TP, FN and FP:

$$TP_k = \sum_{(x,y)\in S} h(x)_k \cdot y_k, \tag{4.7}$$

$$FP_k = \sum_{(x,y)\in S} h(x)_k \cdot (1 - y_k), \tag{4.8}$$

$$FN_k = \sum_{(x,y)\in S} (1 - h(x)_k).y_k. \tag{4.9}$$

The micro F-measure averages over labels by first aggregating TP, FP and FN for all labels and then computing PR, RC and FM based on these aggregates:

$$PR_{micro} = \frac{\sum_{k=1}^{K} TP_k}{\sum_{k=1}^{K} (TP_k + FP_k)}, \tag{4.10}$$

$$RC_{micro} = \frac{\sum_{k=1}^{K} TP_k}{\sum_{k=1}^{K} (TP_k + FN_k)}, \tag{4.11}$$

$$FM_{micro,\beta} = \frac{(1 + \beta^2) \cdot PR_{micro} \cdot RC_{micro}}{\beta^2 \cdot PR_{micro} \cdot RC_{micro}}. \tag{4.12}$$

Alternatively, the macro averaged F-measure is defined as the average over label-specific F-measures that are computed as in the binary case:

$$PR_k = \frac{TP_k}{TP_k + FP_k} \qquad PR_{macro} = \frac{1}{K}\sum_{k=1}^{K} PR_k, \tag{4.13}$$

$$RC_k = \frac{TP_k}{TP_k + FN_k} \qquad RC_{macro} = \frac{1}{K}\sum_{k=1}^{K} RC_k, \tag{4.14}$$

$$FM_{k,\beta} = \frac{(1+\beta^2)\cdot PR_k \cdot RC_k}{\beta^2 \cdot PR_k \cdot RC_k} \qquad FM_{macro,\beta} = \frac{1}{K}\sum_{k=1}^{K} FM_{k,\beta}. \tag{4.15}$$

Whereas the micro F-measure favours larger classes, macro F-measure treats all classes equally (Sokolova & Lapalme, 2009; Sorower, 2010). Since in the tasks considered in this thesis, the performance on specific classes is of interest, the focus will lie on comparing the individual $FM_{k,\beta}$. Since for many downstream NLP applications that improve by operating on a disfluency-cleaned transcript, the classes of identified disfluencies is not directly relevant (statistical machine translation, for example, can be improved by simply removing disfluencies), experiments will also report a version of the F-measure F_{disf} that does not distinguish between individual labels. For the disfluency detection task on the meeting data set, for example, the set of labels is aggregated into the two labels {`no disfluency`, `disfluency`}, with

$$h'(x) = \texttt{disfluency} \Leftrightarrow h(x) \neq \texttt{no disfluency}$$

and

$$h'(x) = \texttt{no disfluency} \Leftrightarrow h(x) = \texttt{no disfluency}.$$

Consequently, in addition to the individual labels, presented results will contain F-measure, precision and recall for `disfluency` as well.

Multilabel classification

In the switchboard data set, each word can carry none, one or more disfluency labels. Equally, a word can be followed by no punctuation mark, a comma or, for example, a full stop and a slash-unit boundary at the same time. For this multilabel classification setting, Sorower (2010) proposes the macro-averaged and micro-averaged F-measure as in the multiclass classification setting. Consequently, results on the switchboard data sets will be reported on the individual $FM_{k,\beta}$, as well as an aggregated F-measure for `disfluency` following the same argumentation as in the multiclass classification setting. In contrast to the multi-class classification task, it is not necessary to explicitly model `no disfluency`, because in the multilabel classification setting, an input does not have to correspond to exactly one label.

4.4. System Architecture

This section introduces technical details of the system that was used to preprocess the data sets and to run, as well as evaluate the experiments described in the following chapters. All

code written during this thesis is, at the moment of its publication, available on github under the directory `https://github.com/matthiasreisser/Disfluency_detection`. All paths to files or directories in the following sections are relative to this github directory. No data files have been synchronized with github because of their size and the fact that the meeting dataset is not publicly accessible.

4.4.1. Preprocessing and Feature Generation

Meeting data set

Reproduction of results for the meeting data set are possible by repeating the following steps. In `/data/meeting`, execute the following bash scripts:

1. `preprocess.sh`: This script traverses the raw text files for the training, validation and test data sets in `/data/meeting/00_raw` and generates corresponding cleaned files in `/data/meeting/00_clean`. Cleaning the data involves the removal of notes, repair of annotators' mistakes that have been found during the data preparation, the removal of superfluous annotations and splitting annotations that span multiple words into one label per word. This process was implemented using `sed` commands and scripts provided in `/data/meeting/perlScripts`. These scripts, as well as the scripts referred to in the rest of this section, reuse code that has been used for generating the results by Cho et al. (2015).

2. `createFeatures.sh`: This script traverses the cleaned text files to produce a variety of files, one for each feature. These files can be found in `/data/meeting/02_finished`. Finally, it concatenates these features into one file for each data set. Each file contains the data matrix in which each row corresponds to a data point and each column corresponds to a feature. These files can be found in `/data/meeting/03_original`. The `createFeatures.sh` script calls several perl scripts that can be found in `/data/meeting/perlScripts`.

3. `vectorsCreate.sh`: This script produces the final data in `/data/meeting/04_featureVectors` that is used in training the models. The logic in this script involves replacing words with their embeddings, POS tags with one-hot encodings as well as centering of features. Feature vectors with alternative word embedding dimensions can be found in `/data/meeting/04_featureVectors200` and `/data/meeting/04_featureVectors50`

Switchboard

The steps for recreating the feature vectors used in training on the switchboard data set essentially follow the same logic as for the meeting data set. All files related to the switchboard data set can be found in `/data/switchboard`.

1. `preprocess.sh`: This script traverses the raw text files, whose links lie in `/data/switchboard/00_raw` and removes the headers of every text file contained in the switchboard corpus, storing the raw speech transcripts in `/data/switchboard/text`. Subsequently, it removes superfluous annotations, executes logic that ensures that subsequent lines of text belong to alternating speakers and reformats spacing and

punctuation. The cleaned text files are then stored in `/data/switchboard/02_clean/individual`.

2. `correctdatasetSplit.sh`: This script merges the cleaned text files from the previous step into train, validation and test data sets, storing them in `/data/switchboard/02_clean`.

3. `createFeatures.sh`: This step is analogous to the corresponding step for the meeting data set, whose output is stored in `/data/switchboard/03_finished` and `data/switchboard/04_original`.

4. `vectorsCreate.sh`: This step is equal to the corresponding step for the meeting data set. Its output is stored in `/data/switchboard/05_featureVectors`.

The scripts for preprocessing and feature generation for the switchboard data set have been optimized to parallelize computational expensive steps for the different data set splits. Due to the size of the switchboard data set, reproducing the above steps can, depending on the available infrastructure, take several hours. The cause of the high computational costs lies in that some lexical and language model features require, for a given word, the traversal of the following words in this current, highly inefficient, implementation.

4.4.2. Model Training

The implementation of the experiments was done using Blocks (van Merriënboer et al., 2015), which is strongly integrated with Fuel (van Merriënboer et al., 2015) for data handling. Blocks is a library which encapsulates the Theano (Bergstra et al., 2010) computational graph and allows for easy specification of recurrent neural network architectures. The logic for data handling (i.e. feature selection, conversion into mini-batches and implementing shift (see 6.2)) lies in `/data`. These files are used in the respective experiments in `/main`, such as, for example `/main/Meeting/meetingDisf.py` for training a recurrent disfluency detection model on the meeting data set. Blocks supplies functionalities for plotting training progress as well as saving and loading models. Where necessary, classes in the Blocks and Fuel framework have been extended for the specific requirements that arose in the progress of this thesis. These extensions can be found in `/utils` for Blocks extensions and in `/data/init.py` as well as in the respective files in `/data` for Fuel extensions.

Individual experiments and their resulting models can be found in `/models`. In this folder, all bash scripts, which set up the local environment and control experiments by passing the desired arguments as arguments to the designated python scripts, can be found. Upon installing Blocks and Fuel along with their dependencies, it is these bash scripts that need to be adapted for the local available hardware on which the models are to be trained.

4.4.3. Model Evaluation

During training, the performance of the model is regularly evaluated on the validation set. Every time a new best model has been found, its parameters are saved in a folder named according to the experiment's parameters in `/models`. In case there exists a previous best

model for an experiment, it is overwritten (See Section 5.3 for an argumentation for this procedure). The arguments specifying the concrete configuration of the experiment are saved in a configuration file in the same folder, allowing reloading and restarting of the exact experiment at a later stage. In order to evaluate the performance of a model on, for example, the test set, there exist evaluation scripts in `/models`. Upon specification of the data set and the specification of the folder name corresponding to the model to be evaluated, these evaluation scripts load the trained model and return performance measures on the data set.

5. Neural Networks for Sequence Modelling

The term *artificial neural networks*, or *neural networks* for short, encompasses a set of models and algorithms that have enjoyed increasing success and popularity over the last decade. Neural networks have been applied to a large variety of tasks, ranging from supervised learning tasks in speech recognition (e.g. (Hannun et al., 2014)), natural language processing (e.g. (Collobert et al., 2011)) to image (e.g. (Krizhevsky et al., 2012)) and video recognition (e.g. (Le et al., 2011)). They have been successfully applied in unsupervised learning settings (e.g. (D. P. Kingma & Welling, 2013)) because of their ability to learn good representations of their inputs and have enjoyed success in research on reinforcement learning (e.g. (Mnih et al., 2013)). Their recent success is largely due to the availability of larger data sets as well as the increasing performance of computers, especially the availability of powerful GPUs, which allowed for models to become bigger and increase their capacity (Bengio et al., 2016). In terms of neural networks, models with more layers, ergo deeper models, became feasible and their increasing depth coined the popular phrase *deep learning*. Originally, neural networks were proposed as a model to understand the human mind (Rumelhart et al., 1986). Although it is clear that the processes in the human brain are far more complex than what neural networks can express, it served as a source of inspiration and shaped the term *artifical neural networks*.

Since, in this thesis, the focus lies on supervised classification and sequence labelling respectively, this chapter introduces neural networks in this setting. First, classic feedforward multilayer perceptrons will be examined and afterwards, their generalization to recurrent neural networks for sequence modelling will be presented. Since the literature on this topic is extensive, this chapter presents only the core concepts of the associated theory and the reader is referred to, for example, Bengio et al. (2016) for a very recent and thorough discussion of neural networks and Graves et al. (2012) for a very comprehensive examination of recurrent neural networks in supervised sequence labelling. Both sources serve as basis for what is presented in the following sections and emphasise is given to the concepts and ideas that will be explored in the following chapters.

5.1. Multilayer Perceptron

Standard feedforward neural networks, in the form that they are commonly used today, are also called multilayer perceptron (MLP) (Rumelhart et al., 1985). In a MLP, several layers composed of single units are connected to form a network of layers. Each layer in such a neural network computes a function over a vector, which is either the input to the network or the output of the previous layer. Finally, the model computes an output vector that is interpreted as $p(y|x)$. Joined together, these individual layers compute a function that is parametrized by the connection strength between units of connected layers. These connections are usually referred to as the parameters of the neural network, or, more commonly, as its weights. A unit of a neural network layer, also sometimes referred to as

neuron, computes a simple non-linear function on the weighted sum of its inputs. Even though each unit computes only a simple function, joined together, the function that is modelled by the neural network can become arbitrarily complex.

5.1.1. Forward Pass

In the following, superscript indices denote layers while subscript indices indicate units in a specific layer.
The output of a neural network is computed by providing an input vector at the input layer and passing the resulting activations of the first hidden layer as input to the next. The activations of each layer are successively passed through the network until the activations of the last layer are regarded as the output of the network. Consider a MLP with I nodes in its input layer. In the following, let $x \in \mathbb{R}^I$ be an input to the network. Unit j is one unit of the first hidden layer. With w_{ij} denoting the weight from from unit i to unit j and b_j a bias term, the activation a_j is computed as:

$$a_j = b_j + \sum_{i=1}^{I} w_{ij} x_j. \tag{5.1}$$

This activation is then passed through a non-linear activation function f to compute the output h_j of unit j:

$$h_j = f(a_j). \tag{5.2}$$

More generally, for each hidden layer $l \in \{1, \ldots, L\}$, the output h^l of the l^{th} layer is computed as

$$h^l = f\left(b^l + W^l h^{l-1}\right), \tag{5.3}$$

with $h^0 = x$ at the input layer. W^l is the weight matrix connecting the layer $l-1$ to layer l and b^l is the corresponding bias vector.

A variety of choices for the activation function f have been described in the literature. The only constraint on f is that it must be differentiable in order to enable the network to be trained using gradient decent. Although linear functions are possible, the expressive power of neural networks stems from the fact that non-linear transformations are used as activation function. Popular choices for f are the sigmoid function $\sigma(x)$ and the hyperbolic tangens:

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{5.4}$$

$$\tanh(x) = 2\sigma(2x) - 1. \tag{5.5}$$

Since the hyperbolic tangens is a linear transformation of the sigmoid function, both are essentially equivalent except for differences in learning dynamics due to their different slopes. Another popular choice for activation function which has been favoured in more recent literature is the rectifier linear unit (ReLU):

$$\phi(x) = \max(0, x). \tag{5.6}$$

The advantage of using ReLU instead of tanh or the sigmoid function lies in that ReLU units do not saturate for large positive inputs. As a consequence, the gradient of rectified linear units does not diminish and learning is simplified in these networks.

5.1.2. Output Layer and Cost Function

Let

$$a^O = b^O + W^O h^K \tag{5.7}$$

be the input to the output layer of an MLP prior through transformation by the output layer's activation function. Depending on the task that is modelled and the associated costfunction, the activation function for the output layer must be chosen specifically.

In the simple **binary classification** setting ($y \in \{C_1, C_2\}$), the output layer contains only one unit, which models $p(C_1|x)$ and should therefore be active if the input x carries the target label C_1. The sigmoid function is the logical choice for the activation function, because it *squashes* a^O into the interval $(0, 1)$ and can therefore be directly interpreted as conditional probability of the class label:

$$\begin{aligned} p(C_1|x) &= f_\theta(x) = \sigma\left(a^O\right) \quad \text{and} \\ p(C_2|x) &= 1 - \sigma\left(a^O\right). \end{aligned} \tag{5.8}$$

$f_\theta(x)$ is defined as the function that the neural network computes, parametrized by θ, which is the collection of all weight matrices and bias vectors in the network. By encoding the target vector y such that $y = 1$ if C_1 is the correct class label for an example x and $y = 0$ otherwise, then y is Bernoulli distributed with $p = f_\theta(x)$:

$$p(y|x) = p^y(1-p)^{1-y}. \tag{5.9}$$

The above generalizes to the Multinoulli distribution (a special case of a Multinomial distribution with $n = 1$) in case an example x may belong to one of $K > 2$ classes ($y \in \{C_1, \ldots, C_K\}$). In order to correctly obtain the conditional probabilities for each class label in this **multiclass classification** task, the softmax activation function is used as the activation function at the output layer. The desired behaviour of the output layer is such that the k^{th} unit in the K-dimensional output layer corresponds to the conditional probability of the example x belonging to the k^{th} class. This functionality is exactly what the softmax function provides:

$$p(C_k|x) = f_\theta(x)_k = f(a^O)_k = \frac{e^{a_k^O}}{\sum_{k'=1}^{K} e^{a_{k'}^O}}, \tag{5.10}$$

in which f denotes the softmax function. The output of the network thus is a K-dimensional vector. The target vector y in such a setting is encoded as a one-hot vector: If, for example, K = 6 and the true label for an example x is C_3, y is encoded as $(0, 0, 1, 0, 0, 0)$.

In using this encoding scheme, y is Multinoulli distributed with $p_k = f_\theta(x)_k$ and

$$p(y|x) = \prod_{k=1}^{K} p_k^y. \tag{5.11}$$

In the third setting that is relevant to the thesis, an example may belong to any number of K classes. In this **multilabel classification** task, the target vector y can therefore be considered a tuple of random variables $(y_1, \ldots, y_K)$. If, for example, K = 6 and the true labels for an example x are C_2 and C_4, the corresponding one-hot encoded vector is $(0, 0, 1, 0, 1, 0, 0)$. In the simplest form of expressing this situation, it is feasible to assume that the probability of one class is conditionally independent from the other classes. The joint distribution over labels can thus be expressed as

$$p(y|x) = p(y_1, \ldots, y_k|x) = \prod_{k=1}^{K} p(y_k|x). \tag{5.12}$$

Now it is possible to decompose this learning task into learning K different neural networks because each task is assumed independent from each other given the input vector. However, since it is, in many cases, justified to assume that the labels y_k share common factors given x, sharing parameters of the model is a more powerful approach.

Cross Entropy

The optimization of a MLP follows the argumentation given in Section 2.3. Following the maximum likelihood principle for finding the parameters θ of the model, by substituting (5.9) into (2.4) and simplifying the logarithm, the cost function for the binary classification task is given by:

$$\begin{aligned} O(\theta; S) &= -\sum_{(x,y)\in S} \ln\left[(f_\theta(x))^y (1 - f_\theta(x))^{1-y}\right] \\ &= -\sum_{(x,y)\in S} y \ln\left(f_\theta(x)\right) + (1-y)\ln\left(1 - f_\theta(x)\right) \end{aligned} \tag{5.13}$$

The resulting costfunction is also called the *cross entropy* cost function. The same argument can be applied to the multiclass classification problem. By substituting (5.11) into (2.4), the cost function becomes:

$$\begin{aligned} O(\theta; S) &= -\sum_{(x,y)\in S} \ln\left[\prod_{k=1}^{K} (f_\theta(x)_k)^{y_k}\right] \\ &= -\sum_{(x,y)\in S} \sum_{k=1}^{K} y_k \ln(f_\theta(x)_k). \end{aligned} \tag{5.14}$$

y_k in the above refers to the k^{th} entry in a one-hot encoding of the target vector.

For optimizing a multi-label MLP, a task-specific cost function has been proposed in the past by Zhang & Zhou (2006). It uses a pairwise ranking loss, which minimizes the number of miss-orderings between a pair of true labels and false labels for a given example. Nam et

al. (2014) propose an alternative method, which is an extension to the previously described cost function for the binary classification setting and will be used in this thesis. The costs associated with one example (x, y) is simply the sum over the binary costs associated with each label. Consequently the cost function to minimize in the multi-label case is given by:

$$\begin{aligned} O(\theta; S) &= -\sum_{(x,y)\in S}\sum_{k=1}^{K} \ln\left[(f_\theta(x)_k)^{y_k}(1-f_\theta(x)_k)^{1-y_k}\right] \\ &= -\sum_{(x,y)\in S}\sum_{k=1}^{K} y_k \ln\left(f_\theta(x)_k\right) + (1-y_k)\ln\left(1-f_\theta(x)_k\right). \end{aligned} \tag{5.15}$$

F-measure

Fitting a neural network through maximization of the cross entropy is not the only possibility. Often, especially in classification settings, the final performance of the model is not reported in terms of the cost function that is used in fitting the model. Instead, as argued in Section 4.3, the F-measure better reflects how good a classifier solves the classification task on an unbalanced data set. Since ultimately, the classifier's performance in terms of F-measure is of interest, it would be desirable to optimize for the F-measure directly. Following this argumentation, the effects of using a modification of the F-measure as cost function is investigated in Section 6.4. This approach has been previously presented in the context of neural networks in (Pastor-Pellicer et al., 2013) for the binary classification case and is here extended for the multi-class and multi-label case.

Computing the F-measure involves choosing the most probable label (2.1) or those labels whose conditional probability lies above a threshold (2.2). However, in choosing a definite hypothesis for the calculation of the F-measure, all the additional information contained in the probabilistic interpretation of the neural network's output is lost. Additionally, the arg max function (2.1) is not differentiable. Both, the loss of information, as well as the practical issue of non-differentiability which is required for its applicability in gradient decent empower the argument to not use the F-measure as it is defined for the evaluation of a classifier's performance. Instead, for training a neural network, the probabilistic version of the F-measure is determined by redefining TP, FP and FN, using $f_\theta(x)$ instead of the hypothesis $h(x)$:

$$TP_{prob,k} = \sum_{(x,y)\in S} f_\theta(x)_k \cdot y_k, \tag{5.16}$$

$$FP_{prob,k} = \sum_{(x,y)\in S} f_\theta(x)_k \cdot (1-y_k), \tag{5.17}$$

$$FN_{prob,k} = \sum_{(x,y)\in S} (1-f_\theta(x)_k) \cdot y_k. \tag{5.18}$$

Following the argumentation in Section 4.3, $TP_{prob,k}$, $FP_{prob,k}$ and $FN_{prob,k}$ for each possible label k are averaged according to the macro-averaging fashion. In order to perform gradient descent on this probabilistic version of the F-measure, the cost function is defined

as the negative of the F-measure:

$$O(\theta; S) = -\frac{1}{K}\sum_{k=1}^{K}\frac{(1+\beta^2)\cdot PR_k \cdot RC_k}{\beta^2 \cdot PR_k \cdot RC_k}$$
$$= -\frac{1}{K}\sum_{k=1}^{K}\frac{(1+\beta^2)\cdot \sum\limits_{(x,y)\in S} f_\theta(x)_k \cdot y_k}{\sum\limits_{(x,y)\in S}(f_\theta(x)_k + \beta^2 \cdot y_k)}. \tag{5.19}$$

by substituting (5.16), (5.17) and (5.18) into (4.13), (4.14) and consequently into (4.15).

5.1.3. Training Neural Networks

Gradient Decent

The most popular method for minimizing the cost function of a neural network is *gradient descent* (Cauchy, 1847). The idea behind gradient descent is to compute the gradient of the cost function with respect to the weights of the network in order to find the direction of steepest descent. Subsequently, the weights are adjusted such that one moves into the direction of steepest descent of the cost function. The update rule for the neural network's weights is therefore given by:

$$\theta \leftarrow \theta - \alpha \nabla_\theta O(S; \theta). \tag{5.20}$$

The cost function O, which is to be minimized, is a function of both, the model parameters θ and the data in the training set S. The parameter α is the learning rate, which determines the size of the step into the direction of the steepest descent. Most cost functions, including the previously discussed cross entropy, can be expressed as a sum over the costs associated with each data point $L(x, y; \theta)$ in the training set S:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \sum_{(x,y)\in S} L(x, y; \theta). \tag{5.21}$$

Evaluating $\nabla_\theta O(S; \theta)$ can be very expensive, since it involves computing the gradient of the costs associated with each data point in S. Although this computation of the gradient is exact, this so-called *batch gradient descent* is rarely used. The alternative approach, called *stochastic gradient descent* (SGD), is computationally feasible compared to batch gradient descent. In SGD, instead of computing the exact (deterministic) gradient $\nabla_\theta O(S; \theta)$, an approximate (stochastic) gradient is computed on a subset $S^m \in S$ of the training data:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \sum_{(x,y)\in S^m} L(x, y; \theta). \tag{5.22}$$

The variant using $m > 1$ examples per subset is called *mini-batch stochastic gradient descent* (as opposed to online stochastic gradient descent for $m = 1$). Applying mini-batch SGD (referred to as SGD in the following) involves repeatedly sampling mini-batches and computing the weight update according to (5.22) until a stopping criterion is met.

The choice of the stopping criterion will be discussed in Section 6.1.2. Besides being computationally tractable for arbitrarily sized data sets, SGD converges faster and its stochasticity helps in escaping local minima (Graves et al., 2012).

A critical choice to the success of applying SGD is the correct choice of the hyperparameter α, which is called the *learning rate*. There exist several sophisticated approaches for tuning the learning rate that will be discussed in Section 6.1.2.

Backpropagation

Using gradient descent requires computing the derivative of the cost function with respect to each weight of the network. Since neural networks are explicitly designed with this differentiability in mind, they can be trained using gradient descent. The technique used to efficiently compute this gradient in a neural network is called *backpropagation*, which is a repeated application of the chain rule. The reason why backpropagation is so efficient lies in that the partial derivatives of O with respect to each weight w_{ij} can be decomposed recursively. This decomposition is possible by recognizing the fact that the activation of units in the network aggregate the influence of previous weights (closer to the input).

The reader is reminded that superscript indices denotes layers, while subscript indices denote specific units in a layer. In order to generalize the derivates to an arbitrary hidden layer $l \in \{1, \dots, L\}$, the function given in (5.1) and (5.2) are reformulated:

$$a_j^l = b_j^l + \sum_{i \in l-1} w_{ij}^l h_i^{l-1}, \tag{5.23}$$

$$h_j^k = f(a_j^k). \tag{5.24}$$

In the above, w_{ij}^l is the weight connecting one unit i in layer $l-1$ to unit j in layer l. The sum in (5.23) is over all units i in layer $l-1$ (assuming a fully connected network) and f is a layer-specific activation function.

As is evident in (5.22), the gradient of the mini-batch costs with respect to one specific weight w_{ij} of the network decomposes as the sum of the gradients over each data point in S^m. Therefore, in the following, the focus will lie on deriving $\partial L(x, y; \theta)/\partial w_{ij}$ for one data point (x, y).

Whereas in the forward pass, the network is traversed from input layer to output layer (i.e. the input is propagated forward), in computing the effect of changes of each weight in the network on the cost function (i.e. the gradient), the network is traversed in the opposite direction (the error is propagated backwards). Therefore, the derivation of the gradients starts at the output layer, and then recursively considers every hidden layer l before arriving at the input layer.

Following the chain rule, the gradient of $L(x, y; \theta)$ with respect to the weights of the output layer can be decomposed by acknowledging that the output of the neural network is equal to the output of the output layer: $h^O = f_\theta(x)$. Furthermore, x influences the cost function through $f_\theta(x)$, the function computed by the neural network. The cost function is therefore reformulated as $L(f_\theta(x), y)$. The gradients for the weights of the output layer are then

given by:

$$\frac{\partial L}{\partial w_{ij}^O} = \frac{\partial L}{\partial a_j^O} \frac{a_j^O}{\partial w_{ij}^O} \tag{5.25}$$

The form of the previous equation depends on the choice of cost function, as well as the corresponding activation function f in the output layer. Assuming a **multi-class classification** task with a softmax activation function (5.10) in the output layer together with the cross entropy cost function given in (5.14), this leads to:

$$\begin{aligned}
\frac{\partial L}{\partial a_j^O} &= \frac{\partial}{\partial a_j^O} - \sum_k y_k \ln(f(a_k^O)) \\
&= -\sum_k y_k \frac{\partial \ln(f(a_k^O))}{\partial a_j^O} \\
&= -\sum_k y_k \frac{1}{f(a_k^O)} \frac{\partial (f(a_k^O)}{\partial a_j^O} \\
&= -y_j(1 - f(a_j^O)) - \sum_{j \neq k} y_k \frac{1}{f(a_k^O)} (-f(a_k^O) f(a_j^O)) \qquad (5.26) \\
&= -y_j + y_j f(a_j^O) + \sum_{j \neq k} y_k f(a_j^O) \\
&= -y_j + f(a_j^O) \underbrace{\sum_k y_k}_{=1} \\
&= f(a_j^O) - y_j. \qquad (5.27)
\end{aligned}$$

(5.26) follows from the fact that for the softmax function f:

$$\frac{\partial f(a_k)}{\partial a_j} = \begin{cases} f(a_j)(1 - f(a_j)) & , \text{ if } \quad j = k \\ -f(a_j) f(a_k) & , \text{ otherwise.} \end{cases} \tag{5.28}$$

For the **binary classification** setting, as well as for the **multi-label classification** setting, the cost functions in (5.13) and (5.14) along with the sigmoid activation function f of (5.4) yield the following intermediate step for the gradient:

$$\frac{\partial L}{\partial f(a_j^O)} = \frac{f(a_j^O) - y_j}{f(a_j^O)(1 - f(a_j^O))}. \tag{5.29}$$

Since, according to the chain rule,

$$\frac{\partial L}{\partial a_j^O} = \frac{\partial L}{\partial f(a_j^O)} \frac{\partial f(a_j^O)}{\partial a_j^O} \tag{5.30}$$

and with the following convenient form for the gradient of the sigmoid activation function,

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)), \tag{5.31}$$

it follows that

$$\frac{\partial L}{\partial a_j^O} = f(a_j^O) - y_j. \tag{5.32}$$

In the binary classification setting, this is equivalent to $f(a^O) - y$, since in this case, there is only one unit in the output layer.

Computing these gradients for the **probabilistic F-measure** as a cost function has an additional requirement that is different from computing the gradient for the cross entropy. Using the F-measure as a cost function requires optimizing the network with mini-batch SDG because the F-measure is not defined for a single data point. Except for this difference, the gradient with respect to the network's weights is computed as in the cross entropy case by propagating the error backwards through the network. In order to specify $\partial O(\theta; S^m)/\partial f(a_j^O)$ for one specific data point (x, y) in the mini-batch, it is helpful to first reformulate (5.19) into a sum of costs over each data point in S^m:

$$\begin{aligned} O(\theta; S^m) &= -\frac{1}{K}\sum_{k=1}^{K}(1+\beta^2)\sum_{(x,y)\in S^m}\frac{f_\theta(x)_k \cdot y_k}{\sum\limits_{(x',y')\in S^m}\left(f_\theta(x')_k + \beta^2 \cdot y'_k\right)} \\ &= \sum_{(x,y)\in S^m} -\frac{1}{K}(1+\beta^2)\sum_{k=1}^{K}\frac{f_\theta(x)_k \cdot y_k}{\sum\limits_{(x',y')\in S^m}\left(f_\theta(x')_k + \beta^2 \cdot y'_k\right)} \quad = \sum_{(x,y)\in S^m} L(x, y; \theta). \end{aligned} \tag{5.33}$$

Consequently, it follows:

$$\frac{\partial L}{\partial f(a_j^O)} = -\frac{1}{K}(1+\beta^2)\left[\frac{y_j}{\sum\limits_{(x',y')\in S^m}(f(a_j^O)' + \beta^2 y'_j)} - \frac{f(a_j^O)\cdot y_j}{\left(\sum\limits_{(x',y')\in S^m}(f(a_j^O)' + \beta^2 y'_j)\right)^2}\right], \tag{5.34}$$

by using the relation

$$\frac{\partial f(a_j^O)'}{\partial f(a_j^O)} = \begin{cases} 1, \text{ for } (x,y) = (x',y') \\ 0, \text{otherwise,} \end{cases}$$

in the above derivation.

Depending on whether the F-measure is used in a multi-class or multi-label case, $f(a_j^O)$ is either the softmax activation function (5.28) or the sigmoid function (5.31).

In order to compute the gradient with respect to the weights of the output layer and to derive the gradients of the hidden layers, it is helpful to first redefine the gradients with respect to a unit's activation:

$$\delta_j^l \overset{def}{=} \frac{\partial L}{\partial a_j^l}. \tag{5.35}$$

Therefore, using (5.35) in (5.25) for any layer l:

$$\frac{\partial L}{\partial w_{ij}^l} = \delta_j^l \frac{\partial a_j^l}{\partial w_{ij}^l} = \delta_j^l f(a_i^{l-1}) \tag{5.36}$$

In the previous, $f(a_i^{l-1})$ is the activation of unit i in the previous layer. The form of δ_j^l depends on whether l represents the output layer or a hidden layer. For $l = O$, the form of δ_j^O has been discussed previously in the context of different classification settings (see (5.27) and (5.32)). In order to specify the gradients with respect to the output of a unit in a hidden layer $f(a_j^l)$, it is necessary to consider how the error caused by $f(a_j^l)$ has influenced all nodes that follow on the way to the output layer. This consideration can be expressed as another application of the chain rule. In the following, the derivative is calculated by summing the error over all the nodes i in the following layer $l+1$ that is attributed to them:

$$\frac{\partial L}{\partial f(a_j^l)} = \sum_i \frac{\partial L}{\partial f(a_i^{l+1})} \frac{\partial f(a_i^{l+1})}{\partial a_i^{l+1}} \frac{\partial a_i^{l+1}}{\partial f(a_j^l)} \tag{5.37}$$

$$= \sum_i \delta_i^{l+1} \frac{\partial a_i^{l+1}}{\partial f(a_j^l)} \tag{5.38}$$

$$= \sum_i \delta_i^{l+1} w_{ji}^{l+1} \tag{5.39}$$

In (5.37), the first two derivatives were replaced with the error of the next layer δ_i^{l+1}, thus recursively specifying the gradient for every hidden layer in terms of the error in the next hidden layer until, for the last hidden layer, the error is specified depending on the classification setting. Computing the gradient with respect to the weights of the hidden layers now is straightforward by using (5.38) in (5.30) to receive δ_j^l

$$\delta_j^l = \frac{\partial f(a_j^l)}{\partial a_j^l} \sum_i \delta_i^{l+1} w_{ji}^{l+1} \tag{5.40}$$

and substitute it in (5.36).

5.2. Recurrent Neural Networks

Recurrent neural networks (RNN) are an extension to the multilayer perceptron, in which recurrent connections in layers are allowed. In this section, the focus lies on presenting a simple recurrent architecture which contains one layer that connects to itself (see Figure 5.1) and, in Section 5.2.2, an extension to the simple architecture that simplifies learning in recurrent networks. By allowing recurrent connections, a neural network is enabled to learn from sequential data because at each time step of the sequence, the RNN has access to a compressed version of the entire history of the sequence it has processed so far. Thus it can capture temporal relations in sequential data. In the following, each element of a sequence will be referred to as an element in a temporal sequence. RNNs are, however, not restricted to processing data with temporal relations. Recent literature applied RNNs

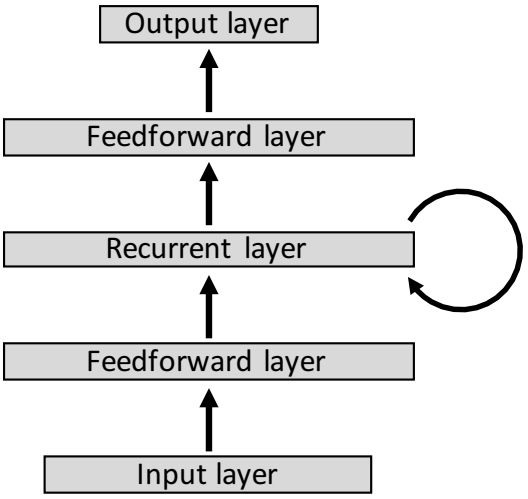


Figure 5.1.: Schematic of a recurrent neural network with two feedforward layers and one recurrent layer. Each arrow indicates a weight matrix and a bias; own representation.

to non-sequential data by presenting it to an RNN in a sequential fashion. Gregor et al. (2015), for example, successfully modelled images as (two-dimensional) sequences by traversing the pixels of an image sequentially. A core idea of recurrent neural networks that allows their application on sequences of arbitrary lengths is the idea of parameter sharing, which has been presented by Waibel, Hanazawa, et al. (1989) in a network-structure called *time-delayed neural networks*. If one would have a separate set of parameters for each time-step of a sequence, one would have to specify parameters for each possible length of a sequence. By re-using parameters across time-steps, the model is able to learn and generalize for sequences of arbitrary length.

5.2.1. Training Recurrent Neural Networks

Forward pass

The forward pass in a recurrent neural network can be explained by extending (5.23) to

$$a_j^t = \sum_{i \in I} w_{ij} x_i^t + \sum_{j' \in H} w_{j'j} h_{j'}^{t-1}. \tag{5.41}$$

In the previous, x_i^t depicts the i^{th} element of the input vector x^t at time t of the input sequence $\mathbf{x}$. For ease of notation, the fact that $a_j t$ is a unit of the first hidden (recurrent) layer in the network, has been omitted. As in a standard feed forward layer, the input to a unit j in a hidden layer contains the weighted sum over all input units. These weights are not dependent on time, since, for each step in the sequence $\mathbf{x}$, these weights are shared. Additionally to the standard feed forward layer, the input to a unit j in a recurrent layer contains the second sum. For each unit j' in the hidden layer, its output at the previous time step $h_{j'}^{t-1}$ is added to the input of unit j at the next time step t. For ease of notation, the bias terms in each sum have been omitted. Instead, it is possible to assume that x_i^t and h_j^{t-1} contain an extra dimension, whose values are always equal to one. As a consequence, the bias term is included in the respective weight matrices. The initial values $h_{j'}^0$ present

an additional parameter vector that needs to be correctly set or learned by the network through SGD. Usually, these initial values are set to zero. The above notation generalizes to the case in which there are more than one hidden layer or the recurrent connections occur not in the first, but in later hidden layers.

Apart from the differences in the input to a recurrent layer, the elements of a recurrent neural network are equal to those in an MLP. Consequently, the same activation functions, the same output layers and the same cost functions can be used as in the MLP case. For a sequence $(\mathbf{x}, \mathbf{y})$, the associated costs can usually be decomposed as the sum over the costs of misclassifying the individual elements at each time step:

$$L(\mathbf{x}, \mathbf{y}; \theta) = \sum_t L(x^t, y^t; \theta). \tag{5.42}$$

Backpropagation through time (BPTT)

In the following, for ease of notation, the superscript in a_j^t denotes time and thus lacks notation for different layers. Therefore, a simple RNN with one recurrent (hidden) layer between the input and output layer is considered. However, the equations presented here generalize to a multi-layer recurrent neural network. For a MLP, δ_j captures how unit j contributes to the error of what follows in the network. Since in a recurrent neural network, a unit not only influences the output layer, but also influences the next time step of the network, a unit in the recurrent layer needs not only consider the error with respect to the units in the output layer, but also needs to include the unit's influence on the error in future time steps. The combined gradient for the output of a recurrent layer's unit j therefore is an extension of (5.40):

$$\delta_j^t = \frac{\partial f(a_j^t)}{\partial a_j^t} \left(\sum_{i \in O} \delta_i^t w_{ji} + \sum_{j' \in H} \delta_{j'}^{t+1} w_{jj'} \right). \tag{5.43}$$

In the above, the first summand considers the error as it is backpropagated from each unit i in the output layer O. Since $L(\mathbf{x}, \mathbf{y}; \theta)$ is the sum over all $L(x^t, y^t; \theta)$, δ_i^t is computed straightforward as in the MLP case, dependent on the classification task. In the way the output of a recurrent network is computed by sequentially propagating x_1 until x_T forward through the network, the error is backpropagated from $t = T$ to $t = 1$. Since beyond $t = T$ there is no error, $\delta_{j'}^{T+1}$ is set to zero for all j'. The second summand in (5.43) therefore aggregates the gradient information from future time steps.

In order to finally compute the gradient with respect to the weights of the recurrent layer, the respective gradients in each time-step are summed, since the weights are shared across time:

$$\frac{\partial L}{\partial w_{ij}} = \sum_t^T \delta_j^t f(a_i^t). \tag{5.44}$$

5.2.2. LSTM

Although backpropagation through time gives a theoretical convenient way to train recurrent network architecture, practical issues arise in reliably learning long-term dependencies. Particularly, the so-called vanishing or exploding gradient problem (Hochreiter, 1991; Hochreiter et al., 2001) hinder the optimization procedure. See Bengio et al. (2016), Chapter 8, for an in-depth discussion of the problem. Applying the chain rule in computing the gradient in BBTT involves multiplying the jacobian matrices of all future time steps. This product tends to either vanish in cases where the individual elements are smaller than one, or explode otherwise. Since, through the sharing of parameters, the matrices are highly related to each other, it is more likely for the gradients to explode or vanish than for the product to be balanced. While the vanishing of the gradient results in that the gradient at a time step contains no error information from future time steps, the explosion of the gradient leads to extreme changes to the network weights, potentially resulting in a much more expensive situation in parameter space and ruining the so far achieved training progress.

The destructive effects of exploding gradients can be easily mitigated heuristically by capping the maximum size of the gradients (gradient clipping). In order to ease the learning of long-term dependencies, several approaches have been made, the most popular of which are the Long Short-Term Memory (LSTM) architectures (Hochreiter & Schmidhuber, 1997). LSTM layers today are used widely in machine learning architectures that include recurrent architectures. Although different variants of the LSTM architecture have been proposed over the years, this discussion will introduce the most widely used *vanilla* architecture, which is an extension of the original architecture and includes changes by F. A. Gers et al. (2000) and F. Gers et al. (2000) to include the forget gate, peephole connections and its training through BPTT. All experiments reported in this thesis involving LSTM layers use this variant of LSTM. For a recent comparison of different LSTM variants' performances, see Greff et al. (2015), in which the authors find that the standard LSTM performs comparably with other variants.

In an LSTM layer, the standard form of a unit's computation (5.41) is extended to include gates which provide it with the capability to remove or add information to its current state. The units' states can be interpreted as the LSTM layer's memory and offers a way for the network to explicitly store information. The output of a LSTM unit thus not only depends on the recurrent layer's output of the previous time step and the current input as in the standard recurrent layer, but also depends on the information encoded in each of the LSTM units' internal memory cell. At each time step, an LSTM unit updates its internal memory cell's state by manipulation of three explicit gating functions. In Graves et al. (2012), each LSTM unit contained several memory cells that were manipulated jointly by the same gating function. However, in the literature, units containing one cell only are the established version of the LSTM architecture.

A graphical overview of an LSTM cell (adapted from Greff et al. (2015)) can be seen in Figure 5.2. Bold arrows represent weighted connections, while dashed arrows represent

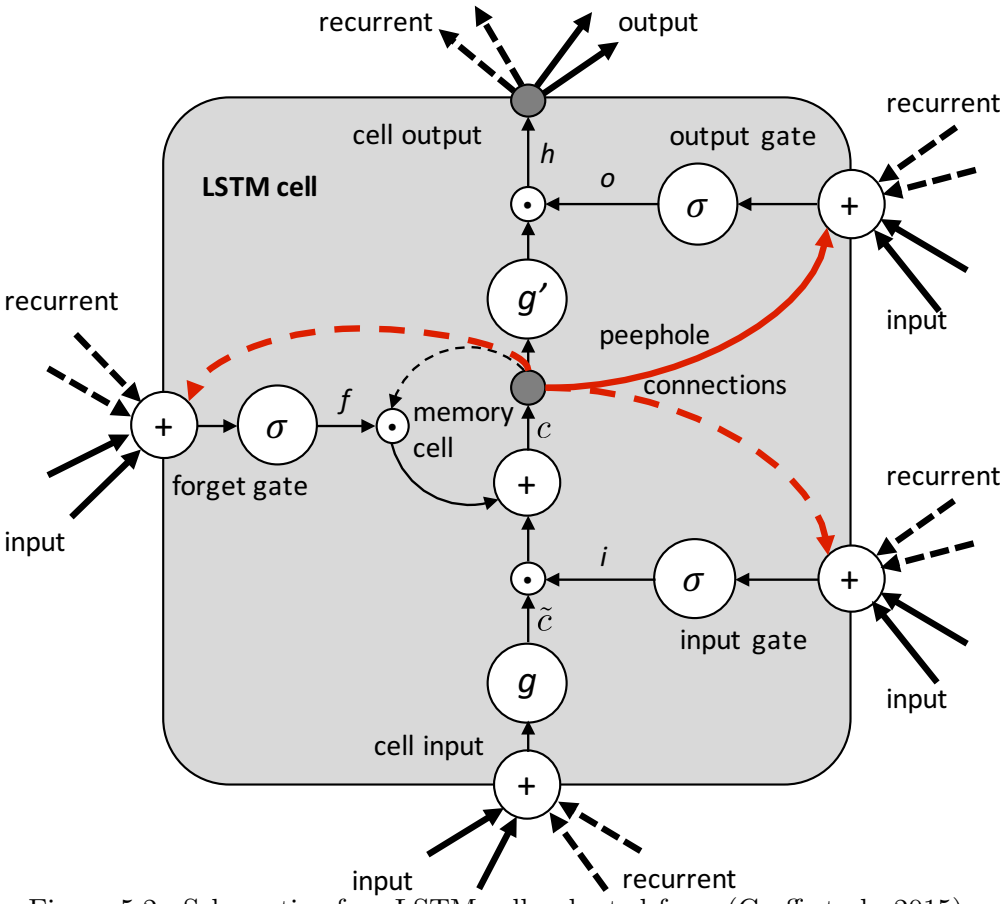


Figure 5.2.: Schematic of an LSTM cell, adapted from (Greff et al., 2015).

information from a previous time step. Red arrows represent the peephole connections, through which the cell state influences the individual gating functions.

Upon receiving the input x_i^t from all units i in the input layer I (or the previous hidden layer, respectively) as well as the LSTM layer's output h_j^{t-1} for each hidden unit j from the previous time step, a proposal $\tilde{c}^t$ for the new cell state is formed by passing the input through the non-linearity g:

$$\tilde{c}^t = g\Big(\sum_{i \in I} w_{ic} x_i^t + \sum_{j \in H} w_{jc} h_j^{t-1}\Big). \tag{5.45}$$

The degree to which this new proposal is integrated into the cell's memory is dependent on the degree to which the cell forgets its current state, as well as to the degree to which the proposal is allowed to influence the current state. The update of the cell's state is thus regulated by the input gate i^t as well as the forget gate f^t. Both functions aggregate the cell's input and the weighted current cell state c^{t-1} through the peephole connections and pass them through the sigmoid function. The sigmoid function squashes its input between 0 and 1 and thus specifies the extend to which the LSTM cell will forget and update its state into c^t respectively:

$$f^t = \sigma\Big(\sum_{i \in I} w_{if} x_i^t + \sum_{j \in H} w_{jf} h_j^{t-1} + w_{cf} c^{t-1}\Big), \tag{5.46}$$

$$i^t = \sigma\big(\sum_{i' \in I} w_{i'i} x_i^t + \sum_{j \in H} w_{ji} h_j^{t-1} + w_{ci} c^{t-1}\big), \tag{5.47}$$

$$c^t = f^t c^{t-1} + i^t \tilde{c}^t. \tag{5.48}$$

Finally, the unit's output h^t is determined by passing the updated cell state through a second non-linearity g', regulated by the output gate o^t. Analogously to the input and forget gate, the output gate aggregates the input to the LSTM unit with the weighted cell state. The peephole connection to the output gate, however, uses the updated cell state c^t as opposed to the other gating functions:

$$o^t = \sigma\big(\sum_{i \in I} w_{io} x_i^t + \sum_{j \in H} w_{jo} h_j^{t-1} + w_{ci} c^t\big), \tag{5.49}$$

and finally,

$$h^t = o^t g'(c^t). \tag{5.50}$$

The key *ingredient* to the LSTM architecture's effectiveness in capturing long-term dependencies lies in the way the LSTM cell retains information in its internal state. Inside the *inner loop* of the LSTM cell, the current state is retained with a constant weight of 1. The extend to which information contained in this current state is kept over time is explicitly controlled through the forget gate, as opposed to being completely overwritten (i.e. forgotten) by the input to the unit as in a simple RNN architecture (see (5.41)).

Although the LSTM architecture is more complicated than a simple RNN layer, the function it computes is still differentiable with respect to all weights (and the initial state c^0). The equations for the gradients of the network's loss function with respect to each of the parameters in a LSTM layer can be found in, for example, Graves et al. (2012).

It is worthy to mention that the presented LSTM architecture and its variants are not the only option for combating the vanishing gradient problem. Other methods, including second-order optimization techniques as well as careful initialization of the weights in recurrent layers have been successful in learning long-term dependencies (Sutskever et al., 2013). However. LSTM layers remain the most popular choice.

5.3. Overfitting

Maximizing the likelihood with SGD eventually will lead to a model which is very good at modelling the data it has seen during training but suffers from bad performance on the validation data set. Since the goal of any machine learning task is to fit a model that generalizes well to unseen data, overfitting the model to the training data must be avoided. Since the generalization capabilities lie at the heart of every machine learning problem, the development of effective regularization techniques is a wide and active area of research. Especially in the deep learning field, in which the number of parameters of a model can go into the billions, applying effective regularization techniques is important. The most

effective technique for avoiding overfitting is, of course, training on more data. Since data is limited, this approach usually involves introducing artificial training examples by applying noise to existing data. There is, however, no sensible interpretation of noise in the domain of textual data that could be applied in the setting studied in this thesis. Chapter 7 as well as Chapter 8 discuss approaches to learning from additional data (additional labels in the case of Multi-Task learning and additional data from a related data source in the Transfer Learning setting). Regularization techniques that are not task related but more generally applicable to the training of neural networks are *early stopping*, *dropout* and *weight decay*. The latter operates by applying a prior distribution on the weights of the network to obtain their maximum a-posteriori estimator. Placing a prior on the weights acts as a regularizing penalty term for undesirable, extreme parameter values.

Early stopping

Early stopping is, due to its simplicity and ease of implementation, one of the most widely used forms of regularization. The basic idea is to evaluate the model's performance regularly on the validation set and to *keep* the parameter setting on which highest performance on the validation set is reached. By stopping the optimization procedure before reaching the maximally achievable likelihood on the training data, the kept parameters instead maximize generalize performance on the validation set. Early stopping is a very popular regularization technique since it does not involve any changes to the learning procedure and does not prohibit using further regularization techniques. Additionally, by conditioning the optimality of the model parameters for the validation set not necessarily on the cost function but on the actual performance metric of interest to the task at hand, it is possible to implicitly mitigate the effect of using a surrogate cost function for optimization. This notion is further discussed in Section 4.3. Early stopping is implemented practically by comparing the model's performance on the validation set with the previously best performance and writing the current model to disk if it outperforms the previously best model.

Dropout

Dropout is a very effective and commonly used regularization technique introduced by Srivastava (2013). The original idea behind this technique is to stochastically set unit activations in the layers of a network to zero with a predetermined probability. During training of the network, the network is thus forced to make sense of the given example using only a subset of its units. This effectively prevents units in a layer from co-adapting and results in several different representations of the data being learned. At test-time, the model can be interpreted as a large ensemble of networks that share weights. Recently, dropout has been interpreted from a bayesian perspective (Gal & Ghahramani, 2015) and ongoing research investigates why dropout acts as such an effective regularization technique. Zaremba et al. (2014) investigated the use of dropout in recurrent neural networks and found that adding dropout to the recurrent connections harms performance. Consequently, when dropout is applied, it is applied solely to the non-recurrent connections in the network.

Due to the fact that the models presented in this thesis are studied not with respect to

achieving highest possible accuracy on their respective tasks, network sizes relative to the amount of available data are not pushed to be as large as possible. *Possible* in this context is to be interpreted as that dropout was not required to regularize a large network that would heavily overfit if dropout were absent. On the contrary, network sizes are relatively small, such that by employing early stopping in combination with an effective learning rate schedule (see Section 6.1.2), overfitting was not an issue.

6. Capturing Time Dependencies through Recurrent Architectures

This chapter describes the setup and results of experiments on both, the meeting data set and the switchboard data set. It will be shown that, by extending the MLP architecture with a recurrent architecture, the model can improve upon the feed forward baseline. Hyperparameters for training the MLP as well as the RNN architecture are given and the specific choices for these parameters are explained. Furthermore, results on experiments with the LSTM architecture will be analysed, as well as results on experiments with the probabilistic F-measure as cost function.

6.1. MLP Architecture as Baseline

In order to establish a baseline, this section first introduces experiments that aim to reproduce the results on the meeting data set reported by Cho et al. (2015). In their work, the authors combine a standard feed forward MLP with a conditional random field approach, and report results for the multi-task learning approach of jointly learning disfluency detection and punctuation prediction on the meeting data set. Here, the focus will lie on reproducing the neural network part of this work, first training each task separately, followed by the reproduction of the actual results in Chapter 7. Contrary to their setup, in this thesis, the network is not pre-trained with denoising autoencoders (Vincent et al., 2010). In the following, the features extracted from the word transcripts as they are presented to the neural network will be described first, followed by the description of the neural network architecture and the choice of hyperparameters.

6.1.1. Features

Using words in a neural network classifier is best done by embedding words in a low dimensional feature space. Instead of building a model whose input layer is as wide as the number of words in the dictionary, each word is mapped to a lower dimensional vector, which is then presented to the network. By doing so, similar words in this vector space lie closer together than dissimilar words and the network can easier learn and generalize. A second advantage in using embedded words instead of a one-hot encoding of the vocabulary is that it is possible to use large quantities of out-of-domain text data for training the word embedding matrix. So instead of being limited to using the relatively small amount of speech transcripts, robust embeddings can be created by including external data. The word embedding matrix that maps each word in the vocabulary to an embedded word vector was created using word2vec (Mikolov et al., 2013) on ~ 340 *million* words, including the words of both, the meeting and the switchboard data set, and a vocabulary of $\sim 182k$ words. Punctuations have been removed in the training corpus and all words have been converted to lower-case. As in Cho et al. (2015), the dimensionality of the embedded vectors is 100. Additionally to the embedded words, POS tags were extracted from the data and included in the feature vector in a one-hot encoding. Since a feed forward neural

network does not have access to the history of the word sequence, the feature vector for each word includes the word embeddings and POS encodings of the two previous and the two following words in the sentence, thus creating a context of words surrounding the current word of interest. Although Cho et al. (2015) use a variety of lexical and language model features, the following experiments are conducted using only word encodings and POS tags in order to establish a baseline from which the effect of adding additional features can be evaluated.

6.1.2. Hyperparameters

In Cho et al. (2015), a MLP with three hidden sigmoid layers consisting of 500 units each is trained on the meeting data set. The details of the optimization procedure were not reported in their work, which is why, after preliminary experiments, the hyperparameters were chosen to be as in Table 6.1. In the following, these parameters will be discussed.

Parameter	Chosen value	Range of experimented values
initial learning rate	0.002	$\{0.1, 0.01, 0.002, 0.001\}$
learning rate decay	0.7	$\{0.5, 0.6, 0.7, 0.8, 0.9\}$
optimizer	RMSProp	{standard SGD, Adam, RMSProp}
parameter initialization	first layer: $\mathcal{N}(0, 0.01)$ other layers: $\mathcal{N}(0, 0.1)$	
# epochs	30	
batch size	100 (meeting) 300 (switchboard)	

Table 6.1.: Hyperparameters for the MLP experiment.

Performance criterion

As has been argued in Section 4.3, the cost function by which a neural network is trained is not necessarily the same as the performance criterion of interest for the specific task. Although one would not expect a huge difference between the resulting network performance when choosing between the overall F-measure for disfluencies and the F-measure for one specific disfluency, performance was consistently better when choosing the F-measure of `rough copies` in the meeting data set and the F-measure for `reparandum` in the switchboard data set over the aggregate F-measure for all disfluency classes. Consequently, the performance criteria in Table 6.2 were chosen in the respective setting for early stopping as well as the learning rate schedule. `punctuation` stands for the aggregated F-measure of all punctuation classes.

task	disfluency detection	punctuation prediction
meeting data set	`(rough) copy`	`punctuation`
switchboard data set	`reparandum`	`punctuation`

Table 6.2.: Performance criteria for different setting.

RMSProp

Preliminary experiments with different optimizers showed a clear advantage of using adaptive algorithms versus using standard stochastic gradient descent in terms of stability of

the results with respect to changing architectures, learning rates, etc.. The advantage was especially significant when training recurrent networks, in which standard SGD often failed to converge. Differences between Adam (D. Kingma & Ba, 2014) and RMSProp (Tieleman & Hinton, 2012) were minimal, however RMSProp showed slightly better results in initial experiments. Adam, as well as RMSProp (which can be understood as an extension to Adagrad (Duchi et al., 2011)) modify the learning rate for each parameter in the network separately, as opposed to standard SGD, which multiplies the gradient for each parameter with the same learning rate (see (5.20)). For RMSProp, the optimizer used in this thesis, the update rule for a specific weight w therefore changes to:

$$r_w^t = \gamma r_w^{t-1} + (1-\gamma)\left(\frac{\partial L}{\partial w}\right)^2 \tag{6.1}$$

$$w \leftarrow w - \frac{\alpha}{\sqrt{r_w^t}}\frac{\partial L}{\partial w}. \tag{6.2}$$

In the above, γ is called the decay factor and α is the global learning rate. RMSProp maintains a moving average of the squared gradient and exhibits very robust performance in mini-batch gradient descent. RMSProp is, as of yet, unpublished, but referencing to the lecture slides by Tieleman & Hinton (2012) seems to be an accepted form of reference to RMSProp in the literature.

Learning rate schedule

In order to converge to a local optimum in SGD, it is necessary to anneal the learning rate over time. This is because sampling from the training data introduces noise that does not become zero, even when the parameters near or reach a local optimum. Formally, a sufficient guarantee for convergence is, that $\sum_{k=1}^{\infty} \alpha_k = \infty$ while $\sum_{k=1}^{\infty} \alpha_k^2 < \infty$, in which k indexes the batches during training over time. In practice, the learning rate is reduced heuristically, usually conditioned on the performance of the validation set, and training is terminated after a limited amount of mini-batches. In preliminary experiments, reducing the learning rate by multiplying it with 0.7 has shown best performance in this task. The learning rate is reduced if the performance on the validation set does not improve between two consecutive epochs, after the end of the second epoch at the earliest. An epoch is said to have passed every time the training procedure has processed the training data completely. Depending on the data set, its size and the complexity of the model along with the characteristics of the machine learning task, the data is usually processed several times over several epochs. Interestingly, conditioning the learning rate reduction on the validation set costs has shown worse performance than conditioning it on the validation set F-measure. At a certain time during the optimization procedure, the model starts to overfit, the learning rate is reduced more often and converges towards zero. Since with a sufficiently small learning rate, the change in parameters (5.22) becomes very small, it makes no sense to continue training the model after this point. This convergence usually is reached before the training procedure has processed the data 30 times, which is the termination criterion.

The initial learning rate of 0.002, as well as a mini-batch size of 100 words for the meeting

data set and 300 words for the switchboard data set were chosen after initial experiments. Higher mini-batch sizes lead to more efficient gradient calculations by taking advantage of fast matrix-matrix versus relatively slow vector-matrix multiplications (Bengio, 2012). However, increasing the batch size might also lead to longer training times in terms of training epochs, since there will be less gradient updates per epoch.

Weight initialization

The weights of the network are initialized randomly by drawing from the normal distribution $\mathcal{N}(\mu, \sigma^2)$. Initial experiments showed that drawing from $\mathcal{N}(0, 0.01)$ for the first hidden layer and drawing from $\mathcal{N}(0, 0.1)$ for all other layers showed good results and this initialization scheme was therefore maintained throughout this thesis.

6.1.3. Meeting Data Set

Results for the experiment run with the above configuration are given in Table 6.3for the meeting data set. A detailed inspection of the different labels of the disfluency classification task for the meting data set reveals that the model is not able to learn the concept of `non-copy` and is very conservative in its prediction of `interruptions`, leading to an overall low F-measure. As expected, the model is good at identifying `filler` since they encompass a limited variety of words and, in most cases, do not depend heavily on the context in which they are embedded. Equally to the discussion by Cho et al. (2015), the MLP exhibits overall conservative behaviour with lower recall and higher precision.

	F-measure	Precision	Recall
disfluency	**49.85**	**79.24**	**36.36**
`filler`	71.77	77.19	67.07
`interruption`	6.37	53.57	3.39
`non copy`	0	0	0
`(rough) copy`	40.95	60.37	30.99
`no disfluency`	90.30	84.25	97.28

Table 6.3.: Results for disfluency detection on the meeting data set.

Predicting punctuations with the above architecture and configuration leads to the results given in Table 6.4. The overall higher F-measure on `punctuation` is higher than `full-stop` and `comma` respectively, which shows that the model is not able to reliably distinguish between both. For the purpose of sentence boundary detection, this inability to distinguish is not a hindrance, however for other purposes, this might be an inconvenience. Due to the very small amount of training data, as well as due to the strong dependence on the first words in a sentence, the model fails to understand the concept of question marks. Results on the recurrent architectures, as well as results with the switchboard data set will show that the lack of training data is indeed the main contributing factor to the bad performance on question marks.

6.1.4. Switchboard Data Set

The same architecture and configuration given above were used to train a model for disfluency detection and a model for punctuation prediction on the switchboard data set. The

	F-measure	Precision	Recall
punctuation	**63.98**	**69.59**	**59.22**
`full stop`	41.66	59.32	32.11
`comma`	33.77	28.39	41.66
`question mark`	0	0	0
`no punctuation`	92.33	90.77	92.94

Table 6.4.: Results for punctuation prediction on the meeting data set.

results given here serve as a baseline for experiments in later sections. They show that the MLP architecture has difficulties identifying the label `interruption` and does not excel in identifying `reparandum` structures. Since interruptions make up only 0.8% of the words and are highly variable in terms of which words they are composed of, as well as in terms of which words follow, these results are not surprising. Edits, on the other hand, which make up only 0.5% of the words in the training data, are highly structured in that a small amount of words are labelled as edits. Consequently, although they are very sparse, the model is very good at identifying them. `reparandum` structures, which exhibit the same difficulties as with `interruption` in that they are highly variable, exist comparably often with 5.2% of the words. Therefore, the model is able to learn the concept of a `reparandum`, while failing to understand `interruption`.

	F-measure	Precision	Recall
disfluency	**83.61**	**92.14**	**76.52**
`interruption`	7.15	91.42	3.72
`coordinating conjunction`	88.25	84.85	91.93
`discourse marker`	93.06	95.29	90.93
`explicit editing`	94.41	97.68	91.35
`filled pause`	96.68	95.20	98.21
`reparandum`	63.09	84.97	50.17

Table 6.5.: Results for disfluency detection on the switchboard data set.

It is interesting to compare the model's performance for punctuation prediction on the different data sets. From comparing Table 6.4 with Table 6.6, it can be surmised that the greater availability of training data leads to the significantly higher performance in these comparable tasks. Also, from Table 6.6 it can be seen that the model's ability to identify `incomplete slash-unit`, as well as `question mark` is relatively low.

	F-measure	Precision	Recall
punctuation	**87.57**	**89.15**	**86.05**
`full stop`	67.89	71.70	64.47
`comma`	76.63	79.08	74.32
`question mark`	37.89	77.61	25.06
`slash-unit`	82.51	81.99	83.04
`incomplete slash-unit`	33.24	53.85	24.04

Table 6.6.: Results for punctuation prediction on the switchboard data set.

6.2. Simple Recurrent Architecture

The power of RNN architectures lies in their ability to capture the information contained in a sequence of events and use the context information it needs from the history of observed events. This is an advantage compared to MLP approaches, for which the observable context is fixed and determined beforehand by concatenating words before and after the current word of interest. However, in presenting the network with a context-window of two words as in the above experiments, the MLP is able to use two words after its current word. The capability to access future context is essential in order to identify reparanda, since their role as such becomes apparent only in the face of the words that follow.

Consider the example sentence "I do think I strongly believe that". In the moment that the model processes the words "I do think", it is impossible to determine that these three words are part of a reparandum structure. Assuming that it is possible to classify the first three words of this sentence upon having processed the fourth word "I", the MLP, with a context window of two words, cannot possibly know that the first word in the sentence is part of a reparandum. Depending on the sentence, the assumption that it is possible to identify the reparandum upon having processed the interregnum or the first word of the repair structure might even be too optimistic.

In order to further extend on this notion, for the recurrent architecture, the input-target sequence pairs have been shifted s steps in time, such that the label for an input $\mathbf{x}_t$ corresponds to the label $\mathbf{y}_{t+s}$ that the network sees at time $t+s$. Therefore, in addition to the past context, all the words in the sub-sequence $\{\mathbf{x}_t, \ldots, \mathbf{x}_{t+s}\}$ along with their additional features (e.g. POS tags) can be considered by the network before formulating a hypothesis for the label corresponding to $\mathbf{x}_t$. In the above exemplary sentence "I do think I strongly believe that", upon shifting the target sequence for two steps in time, the learning task would look like this:

y:	-	-	RM	RM	RM	0	0	0	0
x:	I	do	think	I	strongly	believe	that	-	-

Assuming that for each input word x^t, the two surrounding words are still included in the input vector, the network now has the possibility to consider the words "I do think I strongly", before having to predict the label for the first word in the sentence. The first experiments with recurrent architectures therefore focus on identifying the step size s that leads to the best performance.

Mini-batch training in a recurrent neural network requires cutting the training data into smaller sequences that can be presented to the network concurrently. In modelling an ASR output word stream that contains no sentence boundaries, there is no assumption about the start and end of a sequence. Therefore, the text is cut arbitrarily into equal length sub-sequences. Out of these sub-sequences, batches are formed and presented to the network. The disadvantage of this approach is that the network, at the beginning of each sequence, starts without knowledge of the words that lie before the first word that is presented to the network. Likewise, the knowledge accumulated by the end of a sequence is lost. The easiest way to alleviate this problem is to increase the length of the individual

sequences in a mini-batch. This, however introduces an increased computational burden because the RNN needs to be rolled out across a longer period of time, through which the gradients must be propagated through. Also, the assumption that a spoken word influences the decision of whether another word, say, 100 time steps in the future, is disfluent or not, seems not to be justified in light of the characteristics of spoken language.

Instead, in future improvements on this system, the hidden states of the network at the end of a mini-batch sequence will be used to initialize the hidden state at the beginning of the next mini-batch sequence. In order to achieve this, the training data will be cut into a number of approximately equal length sequences. The number of sequences is equal to the desired batch size. These individual sequences are then cut into sub-sequences of the desired length that one would want to propagate the errors through. In order to sensibly associate the knowledge accumulated in the hidden state at the end of one such mini-batch sub-sequence, the sub-sequences are presented to the network in sequence.

Since a recurrent neural network is able to encode the past context it has seen, in combination with the above introduced shift parameter, the network should be able to learn the correct mapping between words and labels without the need for explicitly adding past and future words to the word at a time step. However, experimenting with a reduced feature set lead to significantly worse performance compared to using the full context window as in the MLP setting. Therefore, also for reasons of comparability of the results, the experiments with recurrent architectures were run with the same feature vectors as for the MLP architecture.

The space of possible recurrent neural network architectures is unlimited. For simplicity and comparability with the MLP setting, an architecture with just one recurrent layer was chosen for these experiments. The resulting architecture consists of one hidden non-recurrent sigmoid layer before the recurrent layer with sigmoid activation, as well as one hidden sigmoid non-recurrent layer after the recurrent layer. Figure 5.1 gives a schematic overview of this architecture. Compared to the MLP architecture, the middle hidden layer was replaced with a recurrent layer, which leads to an increase in the total number of model parameters by 500^2. Although increasing the number of model parameters increases the model's expressive power on its own, it is reasonable to argue that the increased performance of the recurrent model is due to the addition of the recurrent layer and not only because of the increased number of parameters. This argument is reasonable because the MLP architecture is already exhibiting overfitting behaviour on the meeting data set when the learning rate is not annealed.

6.2.1. Hyperparameters

Sequence length

The length of individual sequences that are processed by the network determines the amount of past context that is considered by the network for a given word late in the sequence. The longer the sequence, the more information is available to the network for determining the correct label of a word. However, as it has been motivated at the beginning of this section, longer sequences lead to an increased computational burden. At the same

Parameter	Chosen value	Range of experimented values
initial learning rate	0.002	$\{0.1, 0.01, 0.002, 0.001\}$
learning rate decay	0.7	$\{0.5, 0.6, 0.7, 0.8, 0.9\}$
optimizer	RMSProp	{standard SGD, Adam, RMSProp}
parameter initialization	first layer: $\mathcal{N}(0, 0.01)$ other layers: $\mathcal{N}(0, 0.1)$	
# epochs	30	
batch size	5 (meeting) 5 (switchboard)	
padding	5	{0,5,10,20}
sequence length	20 (meeting) 100 (switchboard)	{20,50,100}
shift	2 (meeting) 3 (switchboard)	$\{0, 1, 2, 3, 4, 5\}$

Table 6.7.: Hyperparameters for recurrent architectures.

time, since weights in a recurrent network are shared across time, longer sequences have a similar effect as an increased batch size with respect to time to convergence. Increasing the sequence length may therefore lead to longer training times. In order to alleviate the fact that in a recurrent network a gradient update involves sequence length times batch size number of words, instead of only the batch size number of words, the batch size for recurrent architectures was reduced to five. Since data in the switchboard data set is not as scarce as in the meeting data set, the sequences were chosen to have a length of 100 words, while words in the meeting data set are processed in sequences of 20 words.

Padding

At the beginning of Section 6.2, it has been explained that the activation of the recurrent layers at the end of a word sequence is not transferred to the initial state of the recurrent layer at the beginning of the next sequence. Therefore, at the beginning of each sequence, the network has no knowledge of the words that came before, and it is not surprising that performance for the first words in a sequence is worse than for words that come later in the sequence. Without the knowledge about previous context, the advantage of using recurrent architectures is lost compared to MLP architectures. At the same time, by introducing the shift parameter s, the labels for the s last words at the end of a sequence are lost. For a sequence of length T, the last input word x_T is presented to the network along with the label y_{T-s}. Therefore, the beginning and the end of a sequence are padded with words, whose labels are masked by zeros and are thus not considered for the calculation of the error of the sequence. These padded words only serve the purpose for introducing sufficient history such that the network can correctly classify the beginning of the actual sequence of interest and they allow for the consideration of the last words in the sequence of interest. While the number of additional words at the end of a sequence is equal to s, the number of words at the beginning was determined to be five in preliminary experiments.

Shift

In order to determine a suitable value for the shift parameter, a grid search over the values $\{0, 1, 2, 3, 4, 5\}$ was done for the tasks disfluency detection, punctuation prediction, as well

as the multi-task setting (see Chapter 7) for both, the meeting as well as the switchboard data set. Table 6.8 shows the differences in F-measure performance for varying values of the shift parameter for disfluency detection on the switchboard data set. As it can be seen, the impact of the shift value on the performance is not significant. The same conclusion about the influence of the shift parameter on network performance can be drawn for all other settings, on both data sets. The results for these settings can be found in Appendix A.

Shift	0	1	2	3	4	5
`interruption`	36.26	35.74	27.43	47.42	48.37	39.72
`coordinating conjunction`	90.55	90.12	90.97	91.20	90.90	90.91
`discourse marker`	94.63	94.62	94.37	94.28	93.94	94.28
`disfluency`	85.49	86.04	86.55	85.83	85.46	86.19
`explicit editing`	96.36	95.85	95.63	95.80	96.48	96.41
`filled pause`	97.47	97.63	97.47	97.63	97.62	97.61
`reparandum`	65.20	68.26	69.71	69.34	68.75	68.84

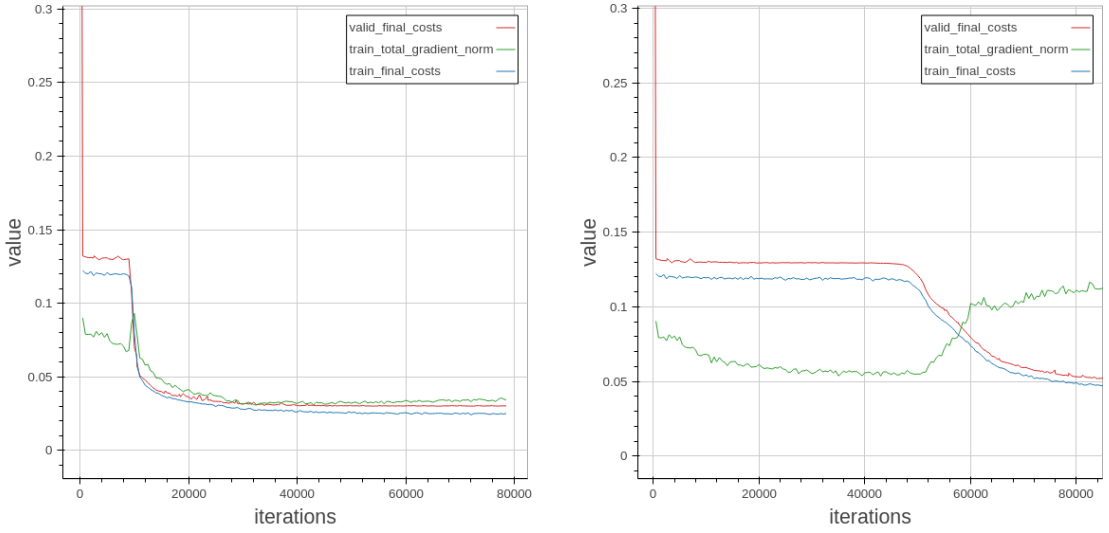
Table 6.8.: Disfluency detection on the switchboard data set with varying shift value and full context window. The table shows F-measure on the validation set.

Since any given input in the sequence x_t is the concatenation of the current word with two previous and two following words in the sentence, it stands to reason that enough future context is captured in x_t for the network to perform reasonably well even for $s < 2$. By shifting the target sequence for more than $s > 2$, the model apparently does not gain in performance, because there are not enough situations in the training data where this combined level of foresight (shift plus context in the input vector) is necessary. In order to confirm this intuition, experiments were conducted, in which the input vector x_t was reduced to only contain the word embedding and POS tag for the current word of interest. The results of these experiments can be found in Appendix B. In comparison to the model trained on context windows, the model trained on the reduced feature vectors (Table 6.9) does indeed benefit from the introduction of the shift parameter. The detection of every disfluency, except for `filled pause`, benefits significantly from shifting the labels by one time step. Increasing the shift to two time steps still results in improved performance on all labels, however not as significantly. Those disfluency classes whose detection performance heavily rely on the context in which they are embedded, namely `interruption` and `repadandum`, profit again by increasing the shift parameter to three time steps.

By separating the input word from its corresponding target label in time, the task of correctly identifying the relationship between them becomes increasingly harder. Consequently, the speed of convergence decreases, and, for $s > 3$, the model is still improving after the termination criterion of 30 epochs has been reached. For a shift of more than three time steps, training generally did not converge within 30 epochs. In order to study the influence of higher shift values, training was allowed to continue for 150 epochs for the case of disfluency detection on the switchboard data set. However, even after this prolonged training time, the models were still improving. Consequently, a shift of more than three time steps is impractical for the switchboard data set. As it can be seen in the

Shift	0	1	2	3	4*	5*
`interruption`	1.16	25.05	27.97	31.96	27.40	23.59
`coordinating conjunction`	82.85	89.01	90.59	90.75	90.39	90.06
`discourse marker`	85.13	94.12	94.58	94.36	94.01	93.56
`disfluency`	71.49	83.14	85.91	85.74	84.61	83.93
`explicit editing`	63.77	96.17	96.20	96.10	95.72	94.85
`filled pause`	97.50	97.60	97.62	97.70	97.59	97.52
`reparandum`	1.75	55.49	66.32	67.45	64.57	61.95

Table 6.9.: Disfluency detection on the switchboard data set with varying shift value and reduced context window. The table shows F-measure on the validation set. * Training was run for 150 epochs.



(a) Shift of three time steps

(b) Shift of four time steps

Figure 6.1.: Training and validation costs over mini-batch iterations for disfluency detection on the switchboard dataset. Values correspond to an average over the last 500 mini-batches. The total gradient norm is plotted in green.

comparison between Figure 6.1a and Figure 6.1b, it takes the model increasingly longer to make initial headway in identifying any relationship between input and label at all. In Figure 6.1b it becomes apparent that the model was allowed to continue to train and still improves after 30 epochs (circa 78,000 mini-batches).

Similar deductions can be made from the other settings on the switchboard data set (Appendix B). For the task of punctuation prediction (Table B.5), increasing the shift value from zero to one improves the performance significantly. Further increasing the shift value continues to improve the results until a peak seems to have been reached at a shift of three or four time steps. Results for the concurrent training of disfluency detection and punctuation prediction show similar results (Table B.6).

In analysing the results for the meeting data set, the aforementioned difficulties in relating a shifted label to the corresponding input word become apparent. There is not enough training data for the model to be learn the concept of `(rough) copy` in a recurrent model without a context window. Since the early stopping criterion is related to the performance

of `(rough) copy`, except for $s = 0$, no model has been saved to disk. For predicting punctuations on the meeting data set, the results (Table B.2) show similar behaviour as on the switchboard data set. In increasing the shift parameter from zero to one, the model improves significantly. For a shift of two time steps, the model improves on `full stop`. For other shift values however, the model fails to converge. For the concurrent training of both tasks (Table B.3), the model is unable to learn about `(rough) copy` and `comma` for a shift value of zero. In increasing the shift to one time step however, the model is able to learn these concepts. The prediction of `full stop` also benefits from introducing a time shift.

In a live productive environment, increasing the shift parameter increases the latency of the system because a label for a given word is received only after s further words have been processed. The decision about the optimum shift parameter therefore needs to take latency requirements to the system into account. As it can be seen from, e.g. the results in Table 6.9, the gains in performance from larger shift parameters diminish. This is because situations in which far foresight is required for predictions are less common than *easier* situations. For the following experiments involving the switchboard data set, a value of three for the shift parameter has been chosen as a compromise between performance, convergence speed and latency. For the meeting data set, taking into account the small amount of data and that the models have difficulties in converging for larger values, a shift value of two will be used.

6.2.2. Meeting Data Det

The results on the test set for training a recurrent neural network with the architecture and hyperparameters discussed above are depicted in Table 6.10. The input vectors for this experiment, as well as all further experiments in this thesis, include a context window of two past and two future words, as opposed to the experiments related to the tuning of the shift parameter in the previous section.

	F-measure		Precision		Recall	
disfluency	**52.27**	(+2.43)	**71.77**	(−7.46)	**41.10**	(+4.75)
`filler`	70.26	(−1.52)	76.81	(−0.38)	64.73	(−2.34)
`interruption`	11.96	(+5.59)	34.59	(−18.98)	7.23	(+3.84)
`non copy`	0.68	(+0.68)	4.17	(+4.17)	0.37	(+0.37)
`(rough) copy`	41.41	(+0.46)	49.12	(−11.25)	35.79	(+4.81)
`no disfluency`	89.90	(−0.40)	85.01	(+0.75)	95.38	(−1.90)

Table 6.10.: Results for disfluency detection on the meeting test set with the recurrent architecture. Differences to own results on the MLP architecture (see Table 6.3) are given in parentheses. Results are reported on the meeting test set.

Overall, the model gains in F-measure performance compared to the MLP architecture by better balancing precision and recall. Especially for `(rough) copy` and `interruption`, the differences in this balance to the MLP architecture are significant. Especially for `interruption`, the trade-off results in an overall increased F-measure performance. Overall, however, the introduction of a recurrent layer has not led to a hugely improved performance.

	F-measure		Precision		Recall	
punctuation	**67.17**	(+3.18)	**73.53**	(+3.94)	**61.82**	(+2.60)
`full stop`	52.56	(+10.89)	57.21	(−2.11)	48.60	(+16.50)
`comma`	35.11	(+1.34)	35.45	(+7.06)	34.78	(−6.88)
`question mark`	0	(+0.00)	0	(+0.00)	0	(+0.00)
`no punctuation`	93.06	(+0.73)	91.38	(+0.61)	94.79	(+0.85)

Table 6.11.: Results for punctuation prediction on the meeting test set with the recurrent architecture. Differences to own results on the MLP architecture (see Table 6.4) are given in parentheses. Results are reported on the meeting test set.

The results for punctuation prediction with the recurrent architecture (Table 6.11) show an improvement versus the MLP architecture. Especially for `full stop`, the model has significantly higher recall, leading to a significantly increased F-measure. Increased precision in predicting `comma` at the cost of reduced recall leads to an overall increased F-measure for this label. For `question mark`, the model is still not able to learn because of the lack of enough training data.

6.2.3. Switchboard Data Set

The results for disfluency detection on the switchboard data set with the recurrent architecture (Table 6.12) show a significant improvement for those labels that were suspected to improve most through the introduction of the recurrent layer in the model. Both, `interruption` as well as `reparandum` show significantly higher recall, leading to an overall improved F-measure. The correct identification of both disfluency classes relies more heavily on the context in which they are embedded. By providing the model with context of past words, combined with the foresight induced by shifting the labels, the model is capable of better detecting these labels. Overall it seems that introducing a recurrent layer results in higher recall, a better balance between recall and precision and consequently a higher F-measure.

	F-measure		Precision		Recall	
disfluency	**85.36**	(+1.75)	**85.75**	(−6.40)	**84.98**	(+8.46)
`interruption`	54.84	(+47.68)	53.67	(−37.75)	56.05	(+52.33)
`coordinating conjunction`	89.58	(+1.33)	86.01	(+1.15)	93.47	(+1.53)
`discourse marker`	92.64	(−0.43)	93.97	(−1.33)	91.34	(+0.40)
`explicit editing`	94.68	(+0.27)	95.59	(−2.10)	93.78	(+2.43)
`filled pause`	97.49	(+0.81)	97.28	(+2.08)	97.70	(−0.51)
`reparandum`	68.64	(+5.54)	71.92	(−13.06)	65.64	(+15.47)

Table 6.12.: Results for disfluency detection on the switchboard data set with the recurrent architecture. Differences to own results on the MLP architecture (see Table 6.5) are given in parentheses. Results are reported on the switchboard test set.

The introduction of a recurrent layer improve the results for punctuation prediction on the switchboard data set (Table 6.13) significantly in the identification of `question mark`. Since the correct placement of a question mark often depends on the first words in a sentence, having access to the history of past words at the end of a sequence is expected to

help in marking questions correctly. Furthermore, the results for punctuation prediction show that an overall increased recall for all labels better balances recall with precision. Except for `comma`, this relation holds and increases the overall F-measure for these labels.

	F-measure		Precision		Recall	
punctuation	**88.42**	(+0.84)	**86.25**	(−2.91)	**90.71**	(+4.66)
`full stop`	72.00	(+4.11)	70.47	(−1.23)	73.60	(+9.13)
`comma`	75.74	(−0.89)	69.05	(−10.03)	83.86	(+9.53)
`question mark`	64.58	(+26.70)	70.25	(−7.36)	59.76	(+34.70)
`slash-unit`	85.10	(+2.59)	82.93	(+0.94)	87.39	(+4.36)
`incomplete slash-unit`	39.51	(+6.28)	44.58	(−9.27)	35.48	(+11.45)

Table 6.13.: Results for punctuation prediction on the switchboard data set with the recurrent architecture. Differences to own results on the MLP architecture (see Table 6.6) are given in parentheses. Results are reported on the switchboard test set.

6.3. LSTM

The LSTM architecture, as it that has been introduced in Section 5.2.2, is the most widely used recurrent architecture in the recent literature. The advantage of LSTM architectures is that long-range dependencies in the data can be easier learned. Consequently, it is interesting to evaluate how a recurrent architecture with a LSTM layer instead of a standard recurrent layer performs. The implicit hypothesis in using a LSTM architecture for disfluency detection and punctuation prediction is therefore that performance is higher on those labels that exhibit long-term dependencies.

As can be see in Table 6.14, introducing the LSTM layer leads to an overall small improvement in performance for disfluency detection on the meeting data set. There are, however, no significant improvements to be gained. The increased precision for `non copy` can not be considered interesting in light of the extremely small recall for this label. On the switchboard data set (see Table 6.16), disfluency detection performance suffered slightly compared to the simple recurrent architecture. The F-measure for every label in lower than the F-measure for the alternative, simpler structure.

Performance for punctuation prediction with an LSTM architecture increased slightly compared to using a simple recurrent architecture on the meeting data set (see Table 6.15). The model has built a rudimentary understanding of `question mark`, however since training data is scarce, the F-measure is still very low. On the switchboard data set, performance does not differ significantly from the baseline.

The hypothesis that disfluency or punctuation labels with long-term dependencies might exhibit higher performance when using a LSTM architecture needs to be rejected in light of the results presented here. Performance is compareable and the increased complexity of a LSTM architecture can not be justified. Apparently, the tasks considered here do not exhibit long-term dependencies that the normal recurrent architecture is not able to learn. In the following chapters, experiments are therefore reported on the simpler, faster to train recurrent architectures.

	F-measure		Precision		Recall	
disfluency	**52.90**	(+0.63)	**72.29**	(+0.52)	**41.71**	(+0.60)
`filler`	70.48	(+0.23)	75.57	(−1.24)	66.03	(+1.30)
`interruption`	12.09	(+0.13)	31.88	(−2.71)	7.46	(+0.23)
`non copy`	1.43	(+0.74)	16.67	(+12.50)	0.75	(+0.37)
`(rough) copy`	42.17	(+0.76)	50.97	(+1.86)	35.97	(+0.17)
`no disfluency`	90.00	(+0.10)	85.14	(+0.14)	95.43	(+0.05)

Table 6.14.: Results for disfluency detection on the meeting data set with the LSTM architecture. Differences to results with the standard recurrent architecture (see Table 6.10) are given in parentheses. Results are reported on the meeting test set.

	F-measure		Precision		Recall	
punctuation	**68.81**	(+1.65)	**69.65**	(−3.88)	**68.00**	(+6.18)
`full stop`	56.32	(+3.77)	52.28	(−4.92)	61.04	(+12.44)
`comma`	32.96	(−2.14)	37.35	(+1.90)	29.50	(−5.28)
`question mark`	7.86	(+7.86)	12.79	(+12.79)	5.67	(+5.67)
`no punctuation`	92.81	(−0.25)	92.55	(+1.17)	93.06	(−1.73)

Table 6.15.: Results for punctuation prediction on the meeting data set with the LSTM architecture. Differences to results with the standard recurrent architecture (see Table 6.11) are given in parentheses. Results are reported on the meeting test set.

	F-measure		Precision		Recall	
disfluency	**85.15**	(−0.22)	**89.11**	(+3.36)	**81.52**	(−3.46)
`interruption`	30.17	(−24.67)	75.70	(+22.03)	18.84	(−37.21)
`coordinating conjunction`	89.41	(−0.18)	85.84	(−0.17)	93.28	(−0.19)
`discourse marker`	92.62	(−0.01)	93.34	(−0.62)	91.91	(+0.57)
`explicit editing`	93.69	(−0.99)	93.07	(−2.53)	94.32	(+0.54)
`filled pause`	97.30	(−0.19)	96.95	(−0.33)	97.66	(−0.04)
`reparandum`	68.12	(−0.52)	75.58	(+3.66)	62.00	(−3.64)

Table 6.16.: Results for disfluency detection on the switchboard data set with the LSTM architecture. Differences to results with the standard recurrent architecture (see Table 6.12) are given in parentheses. Results are reported on the switchboard test set.

	F-measure		Precision		Recall	
punctuation	**88.50**	(+0.09)	**86.21**	(−0.03)	**90.92**	(+0.22)
`full stop`	71.47	(−0.53)	69.46	(−1.02)	73.60	(+0.00)
`comma`	76.13	(+0.40)	69.70	(+0.65)	83.88	(+0.02)
`question mark`	66.13	(+1.55)	74.40	(+4.14)	59.52	(−0.24)
`slash-unit`	84.85	(−0.25)	82.67	(−0.26)	87.15	(−0.24)
`incomplete slash-unit`	40.75	(+1.24)	46.95	(+2.37)	36.00	(+0.52)

Table 6.17.: Results for punctuation prediction on the switchboard data set with the LSTM architecture. Differences to results with the standard recurrent architecture (see Table 6.13) are given in parentheses. Results are reported on the switchboard test set.

6.4. F-measure as Cost Function

Instead of maximizing the log-likelihood of the data (see (2.4)) through minimization of the cross entropy cost function, other cost functions for training neural networks are possible. As has been argued in Section 5.1.2, since ultimately a model's performance in terms of F-measure is relevant, it is interesting to evaluate the use of a probabilistic version of the F-measure (see (5.19)) as a cost function. Since the F-measure is especially suited for evaluating unbalanced data sets, the implicit hypothesis is that a network, when trained with the probabilistic F-measure cost function on the unbalanced meeting or switchboard data sets, performs better than when trained with the cross entropy cost function.

This hypothesis, however, showed to be difficult to evaluate, because in the available time for hyperparameter tuning in the course of this thesis, only the task of disfluency detection on the switchboard data set was successfully trained to convergence with the F-measure cost function. With a learning rate of 0.0002 and otherwise equivalent hyperparameters and architecture setting as described in Section 6.2.1, the results for training disfluency detection on the switchboard data set are given in Table 6.18.

	F-measure		Precision		Recall	
disfluency	84.30	(−1.06)	88.05	(+2.30)	80.86	(−4.12)
`interruption`	44.39	(−10.45)	40.15	(−13.53)	49.62	(−6.42)
`coordinating conjunction`	90.42	(+0.84)	88.81	(+2.79)	92.10	(−1.36)
`discourse marker`	93.28	(+0.65)	97.09	(+3.12)	89.76	(−1.58)
`explicit editing`	95.56	(+0.88)	96.63	(+1.04)	94.51	(+0.72)
`filled pause`	97.62	(+0.13)	98.30	(+1.01)	96.96	(−0.74)
`reparandum`	64.48	(−4.15)	79.41	(+7.49)	54.28	(−11.35)

Table 6.18.: Results for disfluency detection on the switchboard data set with the F-measure cost function. Differences to results with the cross entropy cost function (see Table 6.12) are given in parentheses. Results are reported on the switchboard test set.

Within the 30 epochs of training, the model is still slightly improving on `reparandum`. Overall, the performance of training with the F-measure cost function is worse than with cross entropy on the more difficult labels `interruption` and `reparandum`. It is interesting to see how the trade-off between precision and recall emerges in this setting compared to cross entropy. The overall lower recall suggests that the F-measure trained model is more conservative in its predictions. The higher precision on the individual labels also suggests the same. Instead of producing false positives, the model accepts that it will miss some positive labels.

6.5. Summary

In this chapter, several experiments in applying neural networks to disfluency detection and punctuation prediction were presented. Introducing recurrent architectures showed different levels of improvements for the different labels involved in these tasks. Overall, the performance for the aggregated labels `disfluency` and `punctuation` improved, which

is an important distinction for label-invariant downstream NLP tasks. As has been argued in the respective sections, the F-measure improvements can be, in most cases, attributed to a higher recall for the individual labels. The introduction of the LSTM architecture showed no significant improvement compared to the simple recurrent architecture. It stands to reason that the tasks considered here do not pose a significant challenge with respect to long-term dependencies that the simple recurrent architecture cannot cope with.

Future work on applying recurrent neural networks to disfluency detection and punctuation prediction will include studying a wider variety of neural network architectures. Whereas the focus of this work lay in showing the advantages of recurrent architectures over MLP architectures, actually maximizing the predictive accuracy on the respective task will involve experimenting with deeper and wider architectures, more sophisticated initialization schemes and the study of other possible features. Introducing deeper architectures will also require studying the effect of dropout and further regularization techniques.

Due to time constraints, experiments with the probabilistic F-measure as cost function were not as extensive as the other experiments in this thesis. The results on disfluency detection on the switchboard data set show promising results with respect to the different balance of precision and recall, compared to the cross entropy cost function. Further research in this direction will include studying the gradient dynamics induced by using different cost functions that lead to these differences in balance. It is to be assumed that the results from these studies will give insight into the dynamics of the hyperparameter space that prevented the models to converge on the other tasks. One possible approach would be to combine two cost functions in order to benefit from each function's advantages. Furthermore it is also interesting to study the effects of building ensembles of models with different precision and recall balance and investigate their predictive performance.

7. Regularization through Multi-Task Learning

Learning punctuation prediction and disfluency detection jointly can be expected to improve the generalization capabilities of the neural network, if it is justified to assume that both tasks can be sensibly learned together. In Cho et al. (2015) and X. Wang et al. (2014), this approach was successful for a variety of models. By introducing a second output layer in a neural network, the costs for each task can be calculated separately in a joint model (see Figure 7.1 for a schema of a recurrent multi-task network). Both output layers might be connected to the same last hidden layer in the network, or there might be several task-dependent hidden layers between the last joined layer and the respective output layer. The model is thus separated into task-specific parameters closer to the output layer and shared parameters closer to the input layer. Because the parameters shared between tasks see more input-target pairs than if each task was modelled by a dedicated network, those shared parameters have higher statistical strength, and can generalize better. This is true, however, only, if the not otherwise specifiable closeness assumption with respect to punctuation prediction and disfluency detection is true in this setting. By training both tasks simultaneously, the model will generate, in the shared layers, a representation of the data that is serviceable to both tasks.

The cost function, which this network is trained to minimize, results from adding the costs from both individual layers. This distinction between layers is only relevant in the multi-class environment, such as with the meeting data set. In this case, each output layer can have only one positive label for a given input. Consequently, each output layer computes the softmax function over its activations (see Section 5.1.2) and no activations of the respective other output layers must be included.

In the multi-label environment, the logistic function constitutes the activation function of the output layers. It is, as opposed to the softmax function, not a vector function and distinguishing between two task-specific output layers is equivalent to considering one larger joint output layer. The remaining network architecture settings used in this chapter, as well as the corresponding training hyperparameters, are equivalent to the settings in Section 6.1.2, unless otherwise noted.

In comparing the influence of a joint treatment of both tasks versus their individual performance, the differences between the recurrent architecture and the MLP architecture does not lie in the foreground. However, since the comparison of recurrent architectures versus MLP is a central part of this thesis, experiments on both, the MLP architecture as well as the recurrent architecture introduced in the previous chapter will be investigated.

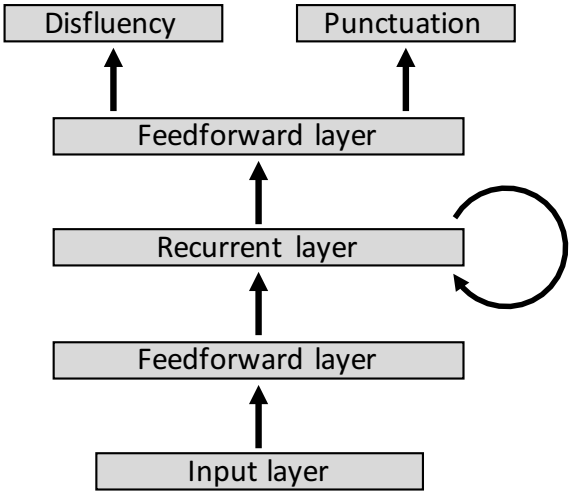


Figure 7.1.: Schematic of a multi-task recurrent neural network. Each arrow indicates a weight matrix and a bias; own representation.

7.1. MLP Architecture

7.1.1. Meeting Data Set

In Cho et al. (2015), the authors report results of jointly learning the disfluency detection task with the punctuation prediction, using the hyperparameters introduced in Section 6.1. Their results, as well as the results of the MLP architecture's performance that have been recorded in this thesis, are depicted in Table 7.1. Considering that the results reported here were achieved without pre-training and using only word embeddings and POS tags as features, it can be assumed that the differences arise from the use of differently chosen hyperparameters.

The generalization capabilities of the model in detecting disfluency increases by including the information from the punctuation prediction task. In both, precision and recall, there is an improvement compared to the performance of the separately trained models. On the other side, punctuation prediction also benefits from the joint learning task. The model improves its prediction capabilities of `full stop` and starts to form a concept of `question mark`.

7.1.2. Switchboard Data Set

Jointly training disfluency detection and punctuation prediction on the switchboard data set leads to a disastrous and inexplicable decrease in performance on both tasks for the MLP architecture. Although the code for producing these results has been thoroughly analysed for errors, a mistake in programming can, of course, not be ruled out. Assuming, however, that no coding errors were made, the results indicate that the model is stuck in a bad local optimum without being able to escape. The fact that the performance of the recurrent network architecture (see Table 7.4) on the joint learning task does not show similar detrimental results gives reason to belief that no property inherent to the data set is responsible for these bad results. Further experimentation on network architecture and

	F-measure		Precision		Recall	
disf. (Cho et al., 2015)	**49.31**		**81.08**		**35.43**	
disfluency	**51.33**	(+1.48)	**79.37**	(+0.13)	**37.93**	(+1.57)
`filler`	71.25	(−0.52)	76.91	(−0.27)	66.37	(−0.7)
`interruption`	9.13	(+2.75)	44.55	(−9.02)	5.08	(+1.69)
`non copy`	0	(+0.0)	0	(+0.0)	0	(+0.0)
`(rough) copy`	43.25	(+2.30)	62.26	(+1.89)	33.13	(+2.15)
`no disfluency`	90.44	(+0.14)	84.57	(+0.32)	97.18	(−0.09)
punc. (Cho et al., 2015)	**52.82**		**65.31**		**44.35**	
punctuation	**67.51**	(+3.53)	**76.84**	(+7.25)	**60.20**	(+0.99)
`full stop`	56.28	(+14.61)	57.88	(−1.44)	54.76	(+22.65)
`comma`	30.59	(−3.17)	39.23	(+10.85)	25.07	(−16.59)
`question mark`	8.98	(+8.98)	22.0	(+22.0)	5.64	(+5.64)
`no punctuation`	93.38	(+1.05)	91.13	(+0.36)	95.75	(+1.81)

Table 7.1.: Results for joint training of punctuation prediction and disfluency detection on the meeting data set with the MLP architecture. Differences to the MLP performance when separately trained (see Table 6.3 and Table 6.4) are included in parentheses. Results are reported on the meeting test set.

hyperparameters might lead to insights about these results, but were out of scope for this thesis.

	F-measure		Precision		Recall	
disfluency	**30.19**	(−53.42)	**17.80**	(−74.35)	**99.49**	(+22.96)
`interruption`	23.24	(+16.09)	14.62	(−76.81)	56.65	(+52.93)
`coordinating conjunction`	77.24	(−11.01)	64.79	(−20.07)	95.63	(+3.69)
`discourse marker`	67.29	(−25.78)	51.33	(−43.97)	97.66	(+6.72)
`explicit editing`	95.03	(+0.61)	95.59	(−2.13)	94.51	(+3.15)
`filled pause`	87.94	(−8.74)	78.68	(−16.53)	99.68	(+1.47)
`reparandum`	12.45	(−50.65)	6.68	(−78.30)	91.55	(+41.38)
punctuation	**38.91**	(−48.66)	**24.88**	(−64.28)	**89.28**	(+3.23)
`full stop`	0.07	(−67.82)	100.0	(+28.30)	0.03	(−64.43)
`comma`	28.53	(−48.10)	17.20	(−61.88)	83.61	(+9.29)
`question mark`	10.87	(−27.01)	87.50	(+9.89)	5.80	(−19.26)
`slash-unit`	77.48	(−5.03)	78.92	(−3.07)	76.09	(−4.94)
`incomplete slash-unit`	3.61	(−29.63)	2.10	(−51.75)	12.85	(−11.19)

Table 7.2.: Results for joint training of punctuation prediction and disfluency detection on the switchboard data set with the MLP architecture. Differences to the MLP performance when separately trained (see Table 6.5 and 6.6) are depicted in parentheses. Results are reported on the switchboard test set.

7.2. Simple Recurrent Architecture

7.2.1. Meeting Data Set

The performance gains for the recurrent network architecture are comparable to the results reported for the MLP in the previous section. The F-measure for `disfluency` increases due to an overall higher precision and recall. Notably, the model also starts to form a concept of `question mark`. Interestingly, the advantage of the recurrent architecture compared to

the MLP in punctuation prediction seems to have been lost. Compared to the results in Section 7.1.1, the F-measure is slightly lower.

	F-measure		Precision		Recall	
disfluency	**53.82**	(+1.54)	**73.96**	(+2.19)	**42.30**	(+1.19)
`filler`	71.11	(+0.86)	78.44	(+1.63)	65.04	(+0.31)
`interruption`	12.91	(+0.95)	25.24	(−9.36)	8.67	(+1.44)
`non copy`	2.34	(+1.65)	16.13	(+11.96)	1.26	(+0.89)
`(rough) copy`	42.43	(+1.02)	54.34	(+5.22)	34.80	(−1.00)
`no disfluency`	90.17	(+0.27)	85.22	(+0.22)	95.72	(+0.33)
punctuation	**66.35**	(−0.82)	**76.18**	(+2.65)	**58.76**	(−3.05)
`full stop`	55.10	(+2.54)	59.47	(+2.26)	51.33	(+2.73)
`comma`	31.59	(−3.52)	39.44	(+4.00)	26.35	(−8.43)
`question mark`	10.19	(+10.19)	13.66	(+13.66)	8.12	(+8.12)
`no punctuation`	93.15	(+0.10)	90.77	(−0.62)	95.66	(+0.87)

Table 7.3.: Results for joint training of punctuation prediction and disfluency detection on the meeting data set with the recurrent architecture. Differences to the recurrent performance when separately trained (see Table 6.10 and 6.11) are depicted in parentheses. Results are reported on the meeting test set.

7.2.2. Switchboard Data Set

As opposed to the performance differences with the MLP architecture, results on the recurrent network architecture (Table 7.4) are more in line with expectations. Overall performance for disfluency detection increases slightly, except for the detection of `interruption`. The F-measure for this label is reduced drastically, which is caused by an unfavourable trade-off between precision and recall compared to the separately trained models. Equally, for punctuation prediction the performance increases slightly for `punctuation`, while exhibiting an unfavourable trade-off between precision and recall for `full stop` and `incomplete slash-unit`.

	F-measure		Precision		Recall	
disfluency	**86.12**	(+0.75)	**91.85**	(+6.10)	**81.05**	(−3.93)
`interruption`	39.00	(−15.83)	77.14	(+23.47)	26.10	(−29.95)
`coordinating conjunction`	90.53	(+0.94)	88.90	(+2.89)	92.21	(−1.26)
`discourse marker`	93.21	(+0.57)	96.07	(+2.11)	90.51	(−0.84)
`explicit editing`	95.24	(+0.56)	96.77	(+1.18)	93.75	(−0.03)
`filled pause`	97.53	(+0.04)	98.26	(+0.98)	96.81	(−0.89)
`reparandum`	69.72	(+1.08)	80.93	(+9.01)	61.24	(−4.40)
punctuation	**88.61**	(+0.19)	**90.84**	(+4.59)	**86.48**	(−4.23)
`full stop`	70.52	(−1.48)	78.15	(+7.68)	64.24	(−9.35)
`comma`	78.02	(+2.28)	79.96	(+10.91)	76.17	(−7.68)
`question mark`	66.67	(+2.08)	77.98	(+7.73)	58.22	(−1.54)
`slash-unit`	85.56	(+0.45)	86.72	(+3.79)	84.43	(−2.97)
`incomplete slash-unit`	31.24	(−8.27)	57.78	(+13.21)	21.41	(−14.08)

Table 7.4.: Results for joint training of punctuation prediction and disfluency detection on the switchboard data set with the recurrent architecture. Differences to the recurrent performance when separately trained (see Table 6.12 and 6.13) are depicted in parentheses. Results are reported on the switchboard test set.

7.3. Summary

In this chapter, the effect of jointly learning disfluency detection and punctuation prediction in one shared model was investigated. Results for experiments on the meeting data set and the switchboard data set for the MLP architecture as well as a recurrent neural network architecture were reported and analysed. Except for an inexplicable extreme performance loss for the MLP architecture on the switchboard data set, jointly training both tasks showed a positive influence on the performance of both tasks. This is in line with previously reported results in which both tasks were combined. Especially considering the fact that the additional computational costs of training the weights of not one, but two output layers is comparably small, including punctuation information, if available, into the task of disfluency detection, is a sensible choice.

Future work on multi-task learning will include experimenting with more sophisticated network structures. Decoupling the task-specific layers further towards the input layers might improve results if both tasks are not as related as one might think. In designing an architecture with a recurrent layer for each specific task, it is possible to experiment with different optimal shift values for each task.

8. Transfer Learning across Data Sets

One major drawback for learning on the meeting data set is that it is comparably small for training a deep neural network. As a consequence, more complex models tend to overfit easily, while smaller models might not be capable to express the relation between input and targets to a satisfying degree. Apart from the previously introduced multi-task learning setting, the most straight-forward solution to the problem of not having enough data is to acquire more data. This, however, can be a costly endeavour. Since the meeting data set is limited and acquiring more data is not possible, in this chapter it will be investigated if, instead of getting more meeting data, it is possible to improve generalization by incorporating knowledge from a similar task on a similar data set, namely detecting disfluencies on the switchboard data set. In this so called *transfer learning* setting, it is assumed that some of the relations that are learned on the switchboard data set are relevant for the meeting data set and can be exploited to improve generalization performance.

In the following, two possible approaches to combining knowledge from the switchboard data set into disfluency detection and punctuation prediction on the meeting data set will be evaluated. Firstly, two possibilities to reuse a model, which has been trained on the switchboard data set, are investigated. This is followed by an investigation of the regularizing power of the switchboard data set on the meeting data set during parallel training of both data sets in one shared model.

8.1. Fine-tuning of a Switchboard Model

Since disfluency detection and punctuation prediction are very similar on the switchboard data set compared to the tasks on the meeting data set, it stands to reason that the knowledge contained in the layers of a switchboard-trained model should provide a good starting point for the training of the meeting data set. Except for the weights of the output-layer, which are label-specific for each data set, the representation of the data in the lower layers can be considered *task-specific*. In order to form an intuition about this distinction between label-specific knowledge in the output layer and more general, task-specific knowledge in the lower, hidden layers, it is reasonable to consider the role of word-embeddings. These word-embeddings that are used as input features to the network for all experiments reported in this thesis, can be considered *specific* for any task in natural language processing. If the word embedding matrix encodes the meaning of words such that similar words lie closer together than words with dissimilar meanings, then one can imagine the higher layers in a trained network to represent intermediate concepts between the meaning of words and the concept of, for example, an `interruption` or a `(rough) copy`. Depending on how close the task, on which the initial model was trained on, is to the task, on which the model will be fine-tuned, the more useful the representation in the upper layers will be for the new task. Azizpour et al. (2014), for example, give an in-depth discussion of the transferability of visual features. In the visual domain, intermediate representations can be easily visualized and therefore allow for an easier interpretation

of this intuition. For the natural language domain there is, however, no intuitive way to interpret the intermediate representations, and assessing their usefulness requires empirical verification. The idea of training recurrent neural networks in a modular way is not a new concept. (Waibel, Sawai, & Shikano, 1989) combines networks trained on individual datasets encompassing different consonant classes for phoneme recognition by fine-tuning select parts of each model. Since the size of networks that are able to model all consonants at the same time were prohibitively expensive to train at the time, techniques to combine select models into one larger model were born out of necessity. In their work, the authors compare different approaches to combining two separate network and report best results when all parameters of the joint model were fine-tuned.

Due to the relative small depth of the network architectures considered here (only considering the number of layers, not the fact that the recurrence in the network can be considered deep), experiments are conducted in two settings. All layers, except for the output layer, are used to initialize the network designated for training on the meeting data set. In one setting, the error is propagated through the whole network. In the second setting, the error is propagated through the output layer only, ignoring the error resulting from imperfect weights in the lower layers. The assumption in this setting is that the representation learned in the lower layers is sufficiently good for the meeting disfluency detection and punctuation prediction task to perform well, without the need to compute gradients and updates for the lower layers. Especially with respect to the burden of computing the gradients for a network unrolled in time, this simplification leads to a large reduction in computational effort. In the first setting, the whole network is fine-tuned to provide the best possible task-specific representation for the meeting data set.

Since recurrent architectures have been shown to outperform MLP architectures on the considered learning tasks, the experiments were conducted on the recurrent architecture presented in Section 6.2.1. Since the lower layers already converged, the learning rate was chosen to be 0.0002 after initial experiments. The output layer was initialized by drawing from $\mathcal{N}(0, 0.1)$. It is important to note that, since the switchboard model was trained with a shift of three time steps, for the fine-tuning, the meeting data target labels were also shifted by three time steps. This is in contrast to the shift of two time steps for previous experiments on the meeting data set. All other architecture settings as well as the training hyperparameters were chosen to be identical to previous experiments in Section 6.2.

In each of the following experiments, the model is initialized with the weights of a model that has been trained on the switchboard data set until convergence. For each of the respective tasks, the initial switchboard models correspond to the models presented in Section 6.2 and Section 7.2 for the joint training of punctuation prediction and disfluency detection.

8.1.1. Fine-tuning all Layers

As it can be seen in Table 8.1, the performance in detecting disfluencies improves significantly when fine-tuning all layers of a model that is trained on the switchboard data set. Especially the performance on `(rough) copy` increased to a large extent. Recall increased

by a large margin, indicating that more words were labelled as `(rough) copy`; at the same time precision increased as well, showing that more of the increased number of positive labels were, in fact, correctly identified. All other labels also profited and resulted in a high increase in performance of `disfluency`.

	F-measure		Precision		Recall	
disfluency	**56.58**	(+4.31)	**76.50**	(+4.73)	**44.89**	(+3.78)
`filler`	71.22	(+0.97)	80.71	(+3.90)	63.73	(−1.00)
`interruption`	13.17	(+1.21)	36.79	(+2.19)	8.02	(+0.79)
`non copy`	6.86	(+6.18)	8.88	(+4.71)	5.60	(+5.22)
`(rough) copy`	54.72	(+13.31)	67.42	(+18.30)	46.05	(+10.25)
`no disfluency`	90.71	(+0.81)	85.92	(+0.92)	96.06	(+0.68)

Table 8.1.: Results for disfluency detection on the meeting data set when fine-tuning a model pre-trained on the switchboard data set. Changes in performance to the model trained on the meeting data set alone (see Table 6.10) are given in parentheses. Results are reported on the meeting test set.

Results on punctuation prediction (Table 8.2) also show a large increase in performance. Especially `full stop` and `question mark` improve drastically.

	F-measure		Precision		Recall	
punctuation	**71.72**	(+4.55)	**81.45**	(+7.92)	**64.06**	(+2.25)
`full stop`	60.93	(+8.37)	64.21	(+7.00)	57.97	(+9.36)
`comma`	35.74	(+0.63)	44.06	(+8.61)	30.07	(−4.71)
`question mark`	32.12	(+32.12)	55.00	(+55.00)	22.68	(+22.68)
`no punctuation`	94.23	(+1.18)	91.99	(+0.61)	96.59	(+1.79)

Table 8.2.: Results for punctuation prediction on the meeting data set when fine-tuning a model pre-trained on the switchboard data set. Changes in performance to the model trained on the meeting data set alone (see Table 6.11) are given in parentheses. Results are reported on the meeting test set.

In the same was as for the seperately trained tasks, performance on the meeting data set increases when jointly training disfluency detection and punctuation prediction. Results for this experiment can be seen in Table 8.3.

8.1.2. Fine-tuning the Output Layer

Results for the experiments in this section are given in relation to the performance results of Section 8.1.1. As it can be seen for the task of disfluency detection (Table 8.4), training only the output layer achieves worse performance than fine-tuning the whole model. In fact, performance is even slightly worse compared to the results from using a randomly initialized model. Apparently, the representation learned from the switchboard data set at the last hidden layer is suboptimal for disfluency detection for the meeting data set.

For punctuation prediction, however, the representation learned with the switchboard data set proves to be very helpful for punctuation prediction on the meeting data set (see Table 8.5). The improvement in performance over the fully fine-tuned model make this experiment the best performing configuration for `punctuation` on the meeting data set.

	F-measure		Precision		Recall	
disfluency	**56.13**	(+2.31)	**81.59**	(+7.63)	**42.78**	(+0.48)
`filler`	71.64	(+0.53)	76.54	(−1.91)	67.33	(+2.3)
`interruption`	7.40	(−5.51)	57.38	(+32.14)	3.95	(−4.72)
`non copy`	0.00	(−2.34)	0.00	(−16.13)	0.00	(−1.26)
`(rough) copy`	55.23	(+12.81)	68.05	(+13.71)	46.48	(+11.68)
`no disfluency`	91.06	(+0.89)	85.61	(+0.39)	97.24	(+1.53)
punctuation	**71.19**	(+4.84)	**83.04**	(+6.86)	**62.30**	(+8.89)
`full stop`	60.24	(+5.14)	64.18	(+4.71)	56.76	(+5.43)
`comma`	34.34	(+2.75)	44.90	(+5.46)	27.80	(+1.46)
`question mark`	24.72	(+14.53)	45.21	(+31.54)	17.01	(+8.89)
`no punctuation`	94.27	(+1.12)	91.66	(+0.90)	97.02	(+1.36)

Table 8.3.: Results for joint training of punctuation prediction and disfluency detection on the meeting data set when fine-tuning a model pre-trained on the switchboard data set. Changes in performance to the model trained on the meeting data set alone (see Table 7.3) are given in parentheses. Results are reported on the meeting test set.

	F-measure		Precision		Recall	
disfluency	**50.77**	(−5.81)	**83.66**	(+7.15)	**36.44**	(−8.45)
`filler`	68.90	(−2.32)	74.71	(−6.00)	63.93	(+0.20)
`interruption`	0.0	(−13.17)	0.00	(−36.79)	0.00	(−8.02)
`non copy`	0.0	(−6.86)	0.00	(−8.88)	0.00	(−5.60)
`(rough) copy`	51.65	(−3.07)	76.78	(+9.36)	38.92	(−7.13)
`no disfluency`	90.66	(−0.05)	84.37	(−1.55)	97.97	(+1.90)

Table 8.4.: Results for disfluency detection on the meeting data set when fine-tuning the output layer of a model pre-trained on the switchboard data set. Changes in performance to the complete fine-tuned model (Table 8.1) are given in parentheses. Results are reported on the meeting test set.

Without training the lower layers, the model is able to better balance precision and recall, leading to an overall improved performance.

	F-measure		Precision		Recall	
punctuation	**73.62**	(+1.91)	**79.18**	(−2.27)	**68.80**	(+4.73)
`full stop`	61.18	(+0.25)	63.03	(−1.18)	59.43	(+1.46)
`comma`	38.62	(+2.88)	42.57	(−1.50)	35.34	(+5.28)
`question mark`	31.36	(−0.76)	48.39	(−6.61)	23.20	(+0.52)
`no punctuation`	94.32	(+0.09)	92.92	(+0.93)	95.77	(−0.82)

Table 8.5.: Results for punctuation prediction on the meeting data set when fine-tuning the output layer of a model pre-trained on the switchboard data set. Changes in performance to the complete fine-tuned model (Table 8.2) are given in parentheses. Results are reported on the meeting test set.

In the joint learning of punctuation prediction and disfluency detection without fine-tuning of the lower layers, the model can improve upon the results for disfluency detection as well as punctuation prediction (see Table 8.6). In this configuration, the performance for `disfluency` is the best achieved compared to the other experiments in this thesis.

	F-measure		Precision		Recall	
disfluency	**56.98**	(+0.86)	**82.81**	(+1.22)	**43.44**	(+0.66)
`filler`	72.73	(+1.09)	77.46	(+0.93)	68.54	(+1.20)
`interruption`	10.99	(+3.59)	47.41	(−9.96)	6.21	(+2.26)
`non copy`	0.73	(+0.73)	16.67	(+16.67)	0.37	(+0.37)
`(rough) copy`	53.95	(−1.28)	69.75	(+1.70)	43.99	(−2.49)
`no disfluency`	91.23	(+0.17)	85.78	(+0.17)	97.42	(+0.18)
punctuation	**72.27**	(+1.08)	**82.01**	(−1.03)	**64.59**	(+2.30)
`full stop`	61.27	(+1.03)	62.52	(−1.65)	60.06	(+3.30)
`comma`	34.23	(−0.12)	48.36	(+3.46)	26.48	(−1.32)
`question mark`	27.96	(+3.24)	34.07	(−11.13)	23.71	(+6.70)
`no punctuation`	94.34	(+0.07)	92.11	(+0.44)	96.68	(−0.34)

Table 8.6.: Results for joint training of punctuation prediction and disfluency detection on the meeting data set when fine-tuning the output layer of a model pre-trained on the switchboard data set. Changes in performance to the complete fine-tuned model (Table 8.3) are given in parentheses. Results are reported on the meeting test set.

8.2. Joint Training of the Meeting and the Switchboard Data Set

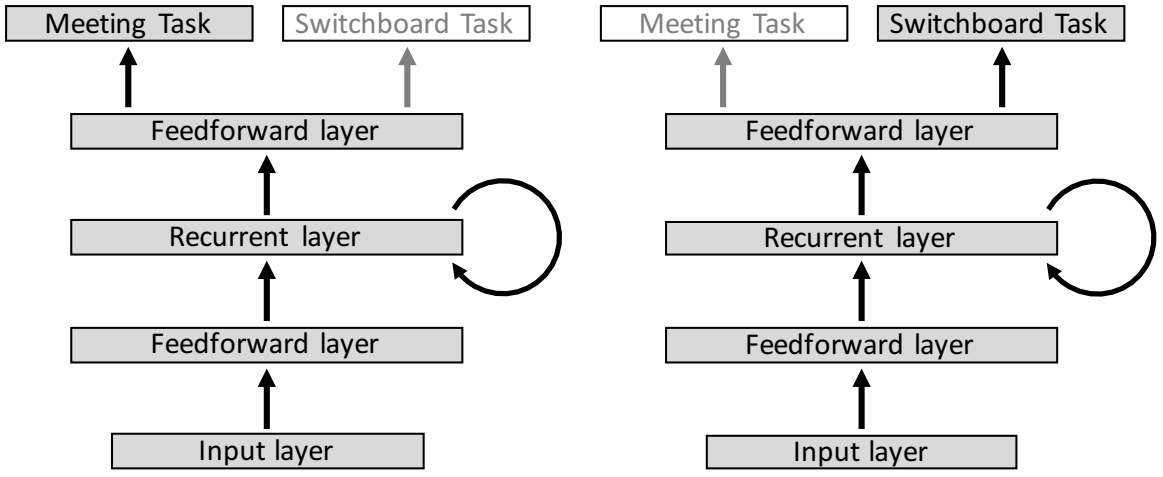


Figure 8.1.: Schematic of jointly training a recurrent neural network. During training, batches of meeting data (left) and switchboard data (right) are alternated.

Instead of separating the learning of the switchboard data set from the learning of the meeting data set and performing one after the other, it is possible to jointly train both tasks in one shared model with randomly initialized parameters. In order to achieve this, each batch of switchboard data is followed by a batch of meeting data and vice versa. As is depicted in Figure 8.1, there is a separate output layer for each data set. No gradients with respect to the parameters of the data set-specific output layer are computed when a mini-batch of data from the other data set is trained.

In order to have an error signal that is balanced between both output layers, the meeting

data set tasks are treated as a multi-label learning task. Consequently, in contrast to previous experiments, the output layer for the meeting data computes the logistic function as opposed to the softmax function. Unfortunately, evaluating the influence of this setting on the performance of training with the meeting data set alone was out of scope of this thesis. Apart from the addition of a separate output layer and the corresponding change in the cost function, the results presented here were achieved with the same architecture settinga and hyperparameters as presented in Section 6.2.1.

The results depicted in Table 8.7 show how the jointly trained model results in a higher F-measure for disfluency detection than the model that is solely trained on the meeting data set. Especially for `(rough) copy`, the performance increase is significant. However, compared to the results in the previous section, the increase in F-measure is not as high.

	F-measure		Precision		Recall	
disfluency	**54.37**	(+2.09)	**71.25**	(−0.53)	**43.95**	(+2.85)
`filler`	70.99	(+0.73)	78.59	(+1.78)	64.73	(+0.00)
`interruption`	14.55	(+2.58)	27.08	(−7.52)	9.94	(+2.71)
`non copy`	4.87	(+4.18)	5.98	(+1.81)	4.10	(+3.73)
`(rough) copy`	52.18	(+10.77)	67.30	(+18.18)	42.61	(+6.82)

Table 8.7.: Results for disfluency detection on the meeting data set when trained concurrently with disfluency detection on the switchboard data set. Changes in performance to the model trained on the meeting data set alone (Table 6.10) are given in parentheses. Results are reported on the meeting test set.

Similarly, for punctuation prediction (see Table 8.8), the results compare favourably to training on the meeting data set alone. Especially for `question mark` and for `full stop`, performance increases. As in the case for disfluency detection however, performance increases not as much as in the experiments presented in the previous section.

	F-measure		Precision		Recall	
punctuation	**68.57**	(+1.41)	**75.57**	(+2.05)	**62.76**	(+0.94)
`full stop`	56.08	(+3.53)	58.52	(+1.32)	53.84	(+5.24)
`comma`	34.16	(−0.95)	39.70	(+4.25)	29.97	(−4.81)
`question mark`	17.89	(+17.89)	23.53	(+23.53)	14.43	(+14.43)

Table 8.8.: Results for punctuation prediction on the meeting data set when trained concurrently with disfluency detection on the switchboard data set. Changes in performance to the model trained on the meeting data set alone (Table 6.11) are given in parentheses.

8.3. Summary

In this chapter, several approaches to transfer knowledge from the bigger switchboard data set to the smaller meeting data set were investigated. In the first approach, the knowledge about the switchboard data that is encoded in the parameters of a trained model was used to initialize the model for the meeting data set. Instead of starting from randomly initialized model parameters, the model already converged to solve a different, but very closely related task on the switchboard data set. Subsequent training on the meeting

data set either fine-tuned the whole model in order to solve the target task or used the features from the last hidden layer of the switchboard model as input features for fitting a task-specific output layer. Experiments were conducted for disfluency detection and punctuation prediction alone, as well as for the joint learning of both tasks.

Future research will include studying different degrees to which the target model is initialized with the weights from the source model. For example, it might be more successful to only use the first hidden layer and randomly initialize the rest of the target model. In studying deeper architectures, the choices for integrating source and target models become more plentiful. Also, when fine-tuning a pre-trained model, the choice of learning rate per layer becomes more relevant. Whereas the randomly initialized output layer might converge faster with a higher learning rate, lower layers might start to overfit fast if the learning rate is not appropriately tuned. The fact that results could be improved by keeping the lower layers of the pre-trained model fixed suggest that this was the case and using a lower learning rate for the lower layers might indeed lead to increased performance. Alternatively to adapting the learning rate, it is also interesting to investigate if performance can again be increased when first, the output layer is trained separately and, after convergence, the whole model is fine-tuned again.

Besides incorporating the knowledge contained in the switchboard data set by initializing with a pre-trained model, the effect of jointly training on the meeting data set and the switchboard data set was studied in this chapter. Although the results were not as convincing as the results from fine-tuning a switchboard model, performance increases were reported compared to training the meeting data set tasks on a randomly initialized model. Future work on this technique will involve studying more sophisticated approaches in weighing the influence of meeting data gradients versus the gradients of the switchboard data set. This might include passing several batches of meeting data for every batch of switchboard data, or vice-versa. Studying the combined effect of jointly learning on two data sets at the same time with jointly learning both tasks at the same time was out of scope for this thesis, but will be part of future research.

Finally, since fine-tuning a pre-trained model gave very good results, fine-tuning the word embedding matrix is a promising extension to this paradigm. The word-embedding matrix was fitted on a very large corpus of data and thus provides a very robust representation of the words of the English language. However, task-specific changes to this embedding might result in a better suited representation. Future work will therefore include experiments in which gradients with respect to the word embeddings are included into the training procedure.

9. Conclusion and Future Work

This thesis investigated the application of recurrent neural network architectures in speech disfluency detection and punctuation prediction. Different approaches to applying these models in different settings have been motivated, described and evaluated in experiments. Furthermore, the influence of jointly learning punctuation prediction with disfluency detection in a shared model was studied and evaluated. In order to account for the limitations of the meeting data set with respect to the amount of available data, this thesis furthermore investigated different approaches to transferring knowledge from the larger switchboard data set to the meeting data set.

Extending a standard MLP architecture to include a recurrent layer showed improvements in performance between 0.8 and 3.2 in F-measure on the aggregated labels `disfluency` and `punctuation` for both, disfluency detection and punctuation prediction on the considered data sets. Specific disfluency classes and punctuation labels improved by an impressive margin, demonstrating the advantage of using a recurrent architecture. Replacing the simple recurrent architecture with a LSTM architecture showed mixed results. This leads to the conclusion that the considered tasks do not exhibit significant long-term dependencies that the simple recurrent architecture cannot learn.

One focus of this thesis lay in showing the differences between an MLP architecture and a comparable (in numbers of parameters) recurrent architecture. The space of possible recurrent architectures, however, is huge and future work will encompass the study of alternative architectures in order to improve the models' predictive performance. In order to alleviate the issues related to the shift parameter, using a bidirectional recurrent network is especially interesting. In a bidirectional recurrent network, the network is given access to all future words in addition to all past words at a given time. For a given subsequence of words, the network thus can learn how future words influence the decision of whether this subsequence is a reparandum. However, along with this powerful advantage of bidirectional architectures comes the disadvantage that such a system cannot be used in a low latency productive environment. For practical purposes, it is desirable to identify disfluencies and punctuation marks as soon as possible, and not after having processed the whole sequence. Depending on the use-case in which such a system is embedded, bidirection architectures might be very promising and useful models.

Combining punctuation prediction and disfluency detection into one task results in a further increase in F-measure performance for the aggregated label `disfluency` for both data sets. Apart from an inexplicable performance loss for the joint learning of the switchboard dataset with the MLP architecture, this multi-task learning approach improved the performance independent of the considered architectures. Especially the results on the meeting dataset, in which the small amount of available training data is a limiting factor, the joint learning led to an improvement of 1.5 point of the F-measure. By supplying the network with two labels for each input word, it was able to generate more robust features for both tasks, respectively. For the switchboard data set, the performance increase was not as

significant. Due to the large amount of available training data, the internal network representations are already very robust. In order for the advantages of the multi-task approach to be prominently visible in the switchboard data set, deeper and wider architectures will be considered in future work.

In addition to considering different architecture choices for the combination of punctuation prediction and disfluency detection, experimenting with the cost function will be part of future research. In the presented experiments, the costs for misclassifying punctuation is equal to the costs for misclassifying disfluencies. By weighing their importance, it is possible to direct the learning task towards the more relevant task while retaining the benefits of training on two tasks at the same time.

Transferring knowledge from the switchboard data set to the learning tasks on the meeting data set showed significant improvement compared to training on the meeting data set alone. By initializing the model for the meeting data set tasks with the weights of the corresponding trained switchboard data set model, the F-measure on punctuation prediction and disfluency detection increased between 4.3 and 6.5 points on the aggregated labels `disfluency` and `punctuation`. Learning on both data sets concurrently showed to be not as effective as fine-tuning the switchboard model. Although the annotation schemes differ between data sets, the experiments in this thesis show how it is possible to combine them in a successful way. Future work on transfer learning will include experimenting with layer-specific learning rates, different architectures and also include fine-tuning of the word-embedding matrix during training.

The input vectors for the models used in this thesis consisted of 100 dimensional word embeddings as well as POS tags in a one-hot vector encoding. Depending on system latency demands it might be feasible to introduce additional features, which simplifies the learning task. As in the work by Cho et al. (2015), possible features include language model features as well as further lexical features. Speaker information, when available, as well as turn information should improve results, especially in the multi-party meeting data set. Additionally, acoustic features have proven to be helpful in detecting both, punctuations as well as disfluencies, and their introduction will be part of future research.

A challenge that is inherent to disfluency detection and punctuation prediction is that the data sets are extremely unbalanced. Individual classes make up as little as 0.5% of all words in the data set. In this thesis, the effects of this skewed label distribution were investigated briefly by experimenting with a probabilistic F-measure cost function for training the network. The results are interesting with respect to the precision and recall balance compared to networks trained with the cross entropy cost function. Future work in this direction shall address further possibilities for dealing with unbalanced data sets. Promising approaches in the literature include sub-sampling the overrepresented word sequences or super-sampling the underrepresented word sequences. Furthermore, by using cost-sensitive learning approaches, the learning progress can be steered towards the underrepresented labels.

On a more abstract level, applying bayesian machine learning techniques to the meeting

data set is a promising way to increase performance. Especially in the domain of limited data, bayesian approaches give a principled way to avoid overfitting by putting higher probability on simpler models. In combination with deep neural networks, bayesian neural networks (Neal, 2012) are an active area of research and applying them to disfluency detection on the meeting data set is an interesting avenue for future work.

The focus of the this thesis lay in showing the influence of recurrent architectures, multi-task learning and transfer learning. Experiments showed the strengths and shortcomings of these approaches and thus investigated principled ways for increasing the performance of these tasks in the deep learning domain. As such, the results reported here leave a lot of room for improvement by performing more intensive hyperparameter tuning, introducing more sophisticated architectures in combination with regularization techniques and more sophisticated optimization techniques. For maximizing the performance of a disfluency detection and punctuation prediction system for its application in a downstream NLP task, the whole *bag of deep learning tricks* that are available should be investigated. This thesis gave an experimental evaluation of more principled approaches and as such serves as a basis for informed future performance maximization.

10. Declaration

Ich versichere hiermit wahrheitsgemäß, die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie (KIT) zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 15. 11. 2015 _____

Matthias Reisser

References

Azizpour, H., Razavian, A. S., Sullivan, J., Maki, A., & Carlsson, S. (2014). From generic to specific deep representations for visual recognition. *CoRR, abs/1406.5774*.

Batista, F., & Mamede, N. (2011). *Recovering capitalization and punctuation marks on speech transcriptions* (Unpublished doctoral dissertation). PhD thesis, Instituto Superior Técnico.

Benesty, J. (2008). *Springer handbook of speech processing*. Springer Science & Business Media.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade* (pp. 437–478). Springer.

Bengio, Y., Goodfellow, I. J., & Courville, A. (2016). *Deep learning.* Retrieved from `http://goodfeli.github.io/dlbook/` (Book in preparation for MIT Press)

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., ... Bengio, Y. (2010). Theano: a cpu and gpu math expression compiler. In *Proceedings of the python for scientific computing conference (scipy)* (Vol. 4, p. 3).

Cauchy, A. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, *25*(1847), 536–538.

Charniak, E., & Johnson, M. (2001). Edit detection and parsing for transcribed speech. In *Proceedings of the second meeting of the north american chapter of the association for computational linguistics on language technologies* (pp. 1–9).

Cho, E., Fünfer, S., Stüker, S., & Waibel, A. (2014). A corpus of spontaneous speech in lectures: The kit lecture corpus for spoken language processing and translation. *LREC, Reykjavik, Iceland*.

Cho, E., Ha, T.-L., & Waibel, A. (2013). Crf-based disfluency detection using semantic features for german to english spoken language translation. In *Proceedings of the international workshop for spoken language translation (iwslt), heidelberg, germany.*

Cho, E., Kilgour, K., Niehues, J., & Waibel, A. (2015). Combination of nn and crf models for joint detection of punctuation and disfluencies. In *Sixteenth annual conference of the international speech communication association.*

Cho, E., Niehues, J., & Waibel, A. (2014). Machine translation of multi-party meetings: Segmentation and disfluency removal strategies.

Christensen, H., Gotoh, Y., & Renals, S. (2001). Punctuation annotation using statistical prosody models. In *Isca tutorial and research workshop (itrw) on prosody in speech recognition and understanding.*

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, *12*, 2493–2537.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, *12*, 2121–2159.

Ferreira, F., & Bailey, K. G. (2004). Disfluencies and human language comprehension. *Trends in cognitive sciences*, *8*(5), 231–237.

Fitzgerald, E., Hall, K., & Jelinek, F. (2009). Reconstructing false start errors in spontaneous speech text. In *Proceedings of the 12th conference of the european chapter of the association for computational linguistics* (pp. 255–263).

Fitzgerald, E., & Jelinek, F. (2008). Linguistic resources for reconstructing spontaneous speech text. In *Lrec.*

Fitzgerald, E., Jelinek, F., & Frank, R. (2009). What lies beneath: Semantic and syntactic analysis of manually reconstructed spontaneous speech. In *Proceedings of the joint conference of the 47th annual meeting of the acl and the 4th international joint conference on natural language processing of the afnlp: Volume 2-volume 2* (pp. 746–754).

Gal, Y., & Ghahramani, Z. (2015). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*.

Gers, F., Schmidhuber, J., et al. (2000). Recurrent nets that time and count. In *Neural networks, 2000. ijcnn 2000, proceedings of the ieee-inns-enns international joint conference on* (Vol. 3, pp. 189–194).

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural computation*, *12*(10), 2451–2471.

Godfrey, J. J., Holliman, E. C., & McDaniel, J. (1992). Switchboard: Telephone speech corpus for research and development. In *Acoustics, speech, and signal processing, 1992. icassp-92., 1992 ieee international conference on* (Vol. 1, pp. 517–520).

Graves, A., et al. (2012). *Supervised sequence labelling with recurrent neural networks* (Vol. 385). Springer.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2015). Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*.

Gregor, K., Danihelka, I., Graves, A., & Wierstra, D. (2015). Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... others (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

He, H., Garcia, E., et al. (2009). Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, *21*(9), 1263–1284.

Heeman, P. A., & Allen, J. F. (1999). Speech repairs, intonational phrases, and discourse markers: modeling speakers' utterances in spoken dialogue. *Computational Linguistics*, *25*(4), 527–571.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*.

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.* A field guide to dynamical recurrent neural networks. IEEE Press.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Honal, M., & Schultz, T. (2003). Correction of disfluencies in spontaneous speech using a noisy-channel approach. In *Interspeech.*

Honnibal, M., & Johnson, M. (2014). Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics*, *2*, 131–142.

Hough, J., & Purver, M. (2014). Strongly incremental repair detection. *arXiv preprint arXiv:1408.6788*.

Hough, J., & Schlangen, D. (2015). Recurrent neural networks for incremental disfluency detection. In *Sixteenth annual conference of the international speech communication association.*

Huang, J., & Zweig, G. (2002). Maximum entropy model for punctuation annotation from speech. In *Interspeech.*

Johnson, M., & Charniak, E. (2004). A tag-based noisy channel model of speech repairs. In *Proceedings of the 42nd annual meeting on association for computational linguistics* (p. 33).

Jones, D. A., Wolf, F., Gibson, E., Williams, E., Fedorenko, E., Reynolds, D. A., & Zissman, M. A. (2003). Measuring the readability of automatic speech-to-text transcripts. In *Interspeech.*

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Le, Q. V., Zou, W. Y., Yeung, S. Y., & Ng, A. Y. (2011). Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis. In *Computer vision and pattern recognition (cvpr), 2011 ieee conference on* (pp. 3361–3368).

Liu, Y., Shriberg, E., Stolcke, A., Hillard, D., Ostendorf, M., & Harper, M. (2006). Enriching speech recognition with automatic detection of sentence boundaries and disfluencies. *Audio, Speech, and Language Processing, IEEE Transactions on*, *14*(5), 1526–1540.

Liu, Y., Stolcke, A., Shriberg, E., & Harper, M. (2005). Using conditional random fields for sentence boundary detection in speech. In *Proceedings of the 43rd annual meeting on association for computational linguistics* (pp. 451–458).

Lu, W., & Ng, H. T. (2010a). Better punctuation prediction with dynamic conditional random fields. In *Proceedings of the 2010 conference on empirical methods in natural language processing* (pp. 177–186).

Lu, W., & Ng, H. T. (Eds.). (2010b). *Better punctuation prediction with dynamic conditional random fields.* Association for Computational Linguistics.

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A., & Taylor, A. (1999). Treebank-3, ldc99t42. *CD-ROM. Philadelphia, Penn.: Linguistic Data Consortium*.

Maskey, S., Zhou, B., & Gao, Y. (2006). A phrase-level machine translation approach for disfluency detection using weighted finite state transducers.

Meteer, M. W., Taylor, A. A., MacIntyre, R., & Iyer, R. (1995). *Dysfluency annotation stylebook for the switchboard corpus.* University of Pennsylvania.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nam, J., Kim, J., Mencía, E. L., Gurevych, I., & Fürnkranz, J. (2014). Large-scale multi-label text classification—revisiting neural networks. In *Machine learning and knowledge discovery in databases* (pp. 437–452). Springer.

Neal, R. M. (2012). *Bayesian learning for neural networks* (Vol. 118). Springer Science & Business Media.

Ostendorf, M., Favre, B., Grishman, R., Hakkani-Tur, D., Harper, M., Hillard, D., ... others (2008). Speech segmentation and spoken document processing. *Signal Processing Magazine, IEEE*, *25*(3), 59–69.

Pastor-Pellicer, J., Zamora-Martínez, F., España-Boquera, S., & Castro-Bleda, M. J. (2013). F-measure as the error function to train neural networks. In *Advances in computational intelligence* (pp. 376–384). Springer.

Qian, X., & Liu, Y. (2013). Disfluency detection using multi-step stacked learning. In *Hlt-naacl* (pp. 820–825).

Rao, S., Lane, I., & Schultz, T. (2007a). Improving spoken language translation by automatic disfluency removal: Evidence from conversational speech transcripts. *Training*, *6370*(46300), 6–50.

Rao, S., Lane, I. R., & Schultz, T. (2007b). Optimizing sentence segmentation for spoken language translation. In *Interspeech* (pp. 2845–2848).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep.). DTIC Document.

Rumelhart, D. E., McClelland, J. L., Group, P. R., et al. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1-2. *Cambridge, MA*.

Savova, G., & Bachenko, J. (2003). Prosodic features of four types of disfluencies. In *Isca tutorial and research workshop on disfluency in spontaneous speech.*

Shriberg, E., Bates, R. A., & Stolcke, A. (1997). A prosody only decision-tree model for disfluency detection. In *Eurospeech* (Vol. 97, p. 23832386).

Shriberg, E., Stolcke, A., Jurafsky, D., Coccaro, N., Meteer, M., Bates, R., ... Van Ess-Dykema, C. (1998). Can prosody aid the automatic classification of dialog acts in conversational speech? *Language and speech*, *41*(3-4), 443–492.

Shriberg, E. E. (1994). *Preliminaries to a theory of speech disfluencies* (Unpublished doctoral dissertation). Citeseer.

Snedeker, J., & Trueswell, J. (2003). Using prosody to avoid ambiguity: Effects of speaker awareness and referential context. *Journal of Memory and language*, *48*(1), 103–130.

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, *45*(4), 427–437.

Soltau, H., Saon, G., & Sainath, T. N. (2014). Joint training of convolutional and non-convolutional neural networks. *to Proc. ICASSP*.

Sorower, M. S. (2010). A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*.

Srivastava, N. (2013). *Improving neural networks with dropout* (Unpublished doctoral dissertation). University of Toronto.

Stolcke, A., Shriberg, E., Bates, R. A., Ostendorf, M., Hakkani, D., Plauche, M., ... Lu, Y. (1998). Automatic detection of sentence boundaries and disfluencies based on recognized words. In *Icslp.*

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (icml-13)* (pp. 1139–1147).

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, *4*.

Tilk, O., & Alumäe, T. (2015). Lstm for punctuation restoration in speech transcripts. In *Sixteenth annual conference of the international speech communication association.*

van Merriënboer, B., Bahdanau, D., Dumoulin, V., Serdyuk, D., Warde-Farley, D., Chorowski, J., & Bengio, Y. (2015). Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619*.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, *11*, 3371–3408.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, *37*(3), 328–339.

Waibel, A., Sawai, H., & Shikano, K. (1989). Modularity and scaling in large phonemic neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, *37*(12), 1888–1898.

Wang, W., Tur, G., Zheng, J., & Ayan, N. F. (2010). Automatic disfluency removal for improving spoken language translation. In *Acoustics speech and signal processing (icassp), 2010 ieee international conference on* (pp. 5214–5217).

Wang, X. (2015). *Advances in punctuation and disfluency prediction* (Unpublished doctoral dissertation). National University of Singapore.

Wang, X., Sim, K. C., & Ng, H. T. (2014). Combining punctuation and disfluency prediction: An empirical study.

Xu, C., Xie, L., Huang, G., Xiao, X., Chng, E. S., & Li, H. (2014). A deep neural network approach for sentence boundary detection in broadcast news. In *Proc. interspeech.*

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zhang, M.-L., & Zhou, Z.-H. (2006). Multilabel neural networks with applications to functional genomics and text categorization. *Knowledge and Data Engineering, IEEE Transactions on*, *18*(10), 1338–1351.

Zwarts, S., & Johnson, M. (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-volume 1* (pp. 703–711).

Zwarts, S., Johnson, M., & Dale, R. (2010). Detecting speech repairs incrementally using a noisy channel approach. In *Proceedings of the 23rd international conference on computational linguistics* (pp. 1371–1378).

Appendices

A. Results for Shift Experiments, full Context Window

Appendix A contains the results for training the recurrent neural network model introduced in Section 6.2 with different shift values. Input vectors contain the word embeddings and POS tags for the current word plus a two word context window of past and future context. Results are reported on the validation set.

	Shift	0	1	2	3	4	5
F-measure	disfluency	55.07	56.22	55.93	55.50	0	0
	filler	77.74	78.17	76.57	77.41	0	0
	interruption	0.75	1.44	5.63	1.52	0	0
	non copy	0	0	1.22	0	0	0
	no disfluency	92.82	93.02	92.71	93.14	90.59	90.59
	(rough) copy	35.32	37.08	38.08	36.41	0	0
Precision	disfluency	73.49	75.29	71.26	78.21	0	0
	filled pause	82.16	81.61	80.43	79.19	0	0
	interruption	12.50	10.00	14.52	40.00	0	0
	non copy	0	0	9.09	0	0	0
	no disfluency	89.24	89.41	89.53	89.14	82.80	82.80
	(rough) copy	41.46	45.73	44.92	52.55	0	0
Recall	disfluency	44.04	44.86	46.03	43.01	0	0
	filled pause	73.77	75.00	73.06	75.70	0	0
	interruption	0.39	0.78	3.49	0.78	0	0
	non copy	0	0	0.65	0	0	0
	no disfluency	96.69	96.93	96.13	97.50	1.00	1.00
	(rough) copy	30.77	31.19	33.06	27.86	0	0

Table A.1.: Disfluency detection on the meeting data set with varying shift value, validation set performance.

	Shift	0	1	2	3	4	5
F-measure	comma	31.10	30.53	37.12	25.29	30.29	9.13
	no punctuation	94.69	95.05	94.97	94.81	94.65	94.96
	full stop	38.54	37.09	34.78	38.82	37.41	39.93
	punctuation	60.17	60.77	61.49	61.49	61.35	55.20
	question mark	17.28	2.92	0	1.67	0	0
Precision	comma	31.95	31.58	32.59	26.87	26.16	38.89
	no punctuation	94.45	94.24	94.52	94.70	94.84	93.02
	full stop	36.56	37.24	38.33	34.44	37.68	35.20
	punctuation	61.37	65.20	63.77	62.05	60.50	67.82
	question mark	31.82	10.53	0	50.00	0	0
Recall	comma	30.30	29.56	43.10	23.89	35.96	5.17
	no punctuation	94.94	95.87	95.41	94.93	94.47	96.99
	full stop	40.73	36.94	31.84	44.49	37.14	46.12
	punctuation	59.01	56.90	59.37	60.95	62.23	46.55
	question mark	11.86	1.69	0	0.85	0	0

Table A.2.: Punctuation prediction on the meeting data set with varying shift value, validation set performance.

	Shift	0	1	2	3	4	5
F-measure	comma	29.02	31.65	29.30	30.61	34.94	21.52
	disfluency	55.37	57.22	58.46	56.32	57.65	49.34
	filler	78.80	78.35	78.47	76.07	79.16	80.15
	interruption	6.68	7.66	10.57	8.06	9.69	0
	non copy	0	0	0	0.90	0	0
	no disfluency	92.98	92.97	93.29	92.77	93.06	92.98
	no punctuation	94.86	95.03	95.15	95.00	94.25	95.03
	full stop	35.70	36.53	35.47	36.48	34.28	38.83
	punctuation	61.14	62.23	59.86	61.23	62.09	51.75
	question mark	13.27	12.50	13.16	9.57	0	0
	(rough) copy	31.13	36.33	37.58	36.88	33.78	14.56
Precision	comma	29.56	29.41	31.45	29.53	27.05	44.72
	disfluency	74.98	73.03	76.39	71.52	74.54	85.93
	filled pause	80.24	77.97	83.40	80.06	80.86	85.42
	interruption	21.74	17.82	17.09	20.99	23.26	0
	non copy	0	0	0	7.69	0	0
	no disfluency	89.30	89.78	89.88	89.61	89.73	87.80
	no punctuation	94.53	94.69	93.91	94.32	95.47	92.32
	full stop	34.21	37.70	38.89	38.41	35.82	41.28
	punctuation	62.81	63.96	67.16	64.85	57.25	72.38
	question mark	29.41	33.33	22.39	28.13	0	0
	(rough) copy	45.38	48.57	51.53	42.86	43.07	46.09
Recall	comma	28.50	34.25	27.42	31.77	49.35	14.17
	disfluency	43.89	47.03	47.35	46.44	47.00	34.60
	filled pause	77.41	78.73	74.08	72.46	77.53	75.50
	interruption	3.95	4.88	7.65	4.99	6.12	0
	non copy	0	0	0	0.48	0	0
	no disfluency	96.97	96.40	96.96	96.15	96.65	98.81
	no punctuation	95.20	95.36	96.42	95.69	93.05	97.91
	full stop	37.33	35.43	32.61	34.73	32.87	36.64
	punctuation	59.55	60.60	53.99	57.98	67.82	40.27
	question mark	8.57	7.69	9.32	5.77	0	0
	(rough) copy	23.69	29.02	29.58	32.37	27.79	8.65

Table A.3.: Punctuation prediction and disfluency detection on the meeting data set with varying shift value, validation set performance.

	Shift	0	1	2	3	4	5
F-measure	interruption	36.26	35.74	27.43	47.42	48.37	39.72
	coordinating conjunction	90.55	90.12	90.97	91.20	90.90	90.91
	discourse marker	94.63	94.62	94.37	94.28	93.94	94.28
	disfluency	85.49	86.04	86.55	85.83	85.46	86.19
	explicit editing	96.36	95.85	95.63	95.80	96.48	96.41
	filled pause	97.47	97.63	97.47	97.63	97.62	97.61
	reparandum	65.20	68.26	69.71	69.34	68.75	68.84
Precision	interruption	62.07	66.51	76.05	45.06	43.49	64.06
	coordinating conjunction	86.84	86.30	88.47	88.73	87.34	87.73
	discourse marker	94.22	94.82	94.96	95.46	92.92	94.38
	disfluency	89.48	88.59	90.32	85.79	84.11	88.57
	explicit editing	97.13	97.41	96.78	96.65	98.37	98.05
	filled pause	98.17	98.15	97.93	97.91	97.84	98.42
	reparandum	76.93	74.12	77.07	71.45	69.53	74.19
Recall	interruption	25.61	24.44	16.74	50.04	54.48	28.79
	coordinating conjunction	94.59	94.30	93.62	93.80	94.77	94.32
	discourse marker	95.05	94.42	93.79	93.13	94.99	94.19
	disfluency	81.83	83.63	83.08	85.88	86.86	83.93
	explicit editing	95.60	94.35	94.51	94.98	94.66	94.82
	filled pause	96.79	97.11	97.01	97.36	97.41	96.81
	reparandum	56.57	63.25	63.63	67.35	67.98	64.21

Table A.4.: Disfluency detection on the switchboard data set with varying shift value, validation set performance.

	Shift	0	1	2	3	4	5
F-measure	comma	74.86	74.74	74.03	73.53	75.50	75.63
	incomplete slash-unit	32.09	38.05	37.47	38.35	36.22	34.63
	full stop	71.52	72.54	71.88	71.80	73.45	73.21
	punctuation	88.58	88.85	88.49	88.51	88.61	88.56
	question mark	63.79	62.63	62.16	59.40	62.16	58.09
	slash-unit	83.63	84.60	83.92	84.95	84.63	83.62
Precision	comma	67.53	66.57	65.36	64.57	68.47	70.30
	incomplete slash-unit	50.41	46.22	41.08	43.79	43.81	38.53
	full stop	75.70	76.20	74.70	76.58	71.64	69.95
	punctuation	87.77	87.47	86.26	86.59	86.66	87.67
	question mark	70.60	73.33	68.15	62.08	69.67	67.58
	slash-unit	80.71	81.75	80.78	82.48	80.85	77.77
Recall	comma	83.98	85.20	85.36	85.37	84.14	81.83
	incomplete slash-unit	23.54	32.34	34.44	34.12	30.87	31.44
	full stop	67.79	69.23	69.26	67.59	75.36	76.80
	punctuation	89.41	90.28	90.85	90.51	90.64	89.48
	question mark	58.18	54.66	57.14	56.94	56.11	50.93
	slash-unit	86.75	87.66	87.31	87.58	88.78	90.42

Table A.5.: Punctuation prediction on the switchboard data set with varying shift value, validation set performance.

	Shift	0	1	2	3	4	5
F-measure	interruption	26.04	35.66	36.86	39.18	25.28	27.94
	coordinating conjunction	90.58	91.36	91.10	92.03	91.52	91.49
	comma	79.25	78.64	78.65	78.69	77.95	78.96
	discourse marker	95.04	94.74	94.71	95.02	94.30	94.29
	disfluency	85.99	86.90	87.07	87.26	86.73	86.68
	explicit editing	96.05	96.61	95.84	96.08	95.87	95.08
	filled pause	97.71	97.86	97.73	97.78	97.71	97.52
	incomplete slash-unit	36.70	29.96	30.45	32.97	34.93	31.14
	full stop	68.20	65.18	67.19	68.01	70.13	69.57
	punctuation	88.25	88.37	88.18	88.64	88.49	88.36
	question mark	62.09	60.63	58.69	60.67	62.30	57.24
	reparandum	65.62	69.49	71.10	70.36	70.00	69.39
	slash-unit	84.07	84.98	84.78	85.58	84.77	85.19
Precision	interruption	76.40	72.00	72.71	71.61	73.11	76.70
	coordinating conjunction	89.36	91.86	92.26	91.24	91.39	90.49
	comma	78.70	76.58	76.74	77.01	75.40	80.18
	discourse marker	95.67	95.40	96.01	96.51	94.92	95.37
	disfluency	93.30	92.69	91.91	92.00	92.36	92.64
	explicit editing	98.24	98.38	97.47	98.04	97.86	98.25
	filled pause	98.58	98.77	98.67	98.77	98.69	98.44
	incomplete slash-unit	50.35	55.93	56.23	58.95	47.80	51.76
	full stop	81.29	84.59	83.64	83.57	78.74	82.45
	punctuation	90.47	90.75	90.85	90.83	89.06	90.97
	question mark	75.82	69.55	81.23	71.28	73.60	71.09
	reparandum	85.38	82.79	79.91	80.67	82.04	83.20
	slash-unit	85.59	87.58	88.20	86.76	84.43	85.78
Recall	interruption	15.69	23.70	24.69	26.97	15.28	17.08
	coordinating conjunction	91.84	90.87	89.96	92.84	91.65	92.51
	comma	79.81	80.80	80.66	80.45	80.68	77.79
	discourse marker	94.42	94.10	93.45	93.57	93.69	93.23
	disfluency	79.74	81.79	82.72	82.99	81.76	81.44
	explicit editing	93.95	94.89	94.25	94.19	93.97	92.10
	filled pause	96.85	96.96	96.81	96.81	96.75	96.63
	incomplete slash-unit	28.87	20.46	20.88	22.89	27.51	22.27
	full stop	58.74	53.02	56.14	57.33	63.22	60.17
	punctuation	86.14	86.11	85.65	86.55	87.93	85.89
	question mark	52.57	53.73	45.95	52.80	54.01	47.91
	reparandum	53.29	59.87	64.04	62.38	61.05	59.51
	slash-unit	82.60	82.52	81.60	84.42	85.11	84.61

Table A.6.: Punctuation prediction and disfluency detection on the switchboard data set with varying shift value, validation set performance.

B. Results for Shift Experiments, no Context Window

Appendix B contains the results for training the recurrent neural network model introduced in Section 6.2 with different shift values. Input vectors contain the word embeddings and POS tags for the current word without a context window. Results are reported on the validation set.

	Shift	0	1	2	3	4	5
F-measure	disfluency	43.85	0	0	0	0	0
	filler	78.12	0	0	0	0	0
	interruption	0	0	0	0	0	0
	non copy	0	0	0	0	0	0
	no disfluency	92.76	90.57	90.56	90.56	90.56	90.59
	(rough) copy	0	0	0	0	0	0
Precision	disfluency	89.45	0	0	0	0	0
	filled pause	85.86	0	0	0	0	0
	interruption	0	0	0	0	0	0
	non copy	0	0	0	0	0	0
	no disfluency	87.04	82.76	82.76	82.75	82.75	82.80
	(rough) copy	0	0	0	0	0	0
Recall	disfluency	29.04	0	0	0	0	0
	filled pause	71.65	0	0	0	0	0
	interruption	0	0	0	0	0	0
	non copy	0	0	0	0	0	0
	no disfluency	99.29	100	100	100	100	100
	(rough) copy	0	0	0	0	0	0

Table B.1.: Disfluency detection on the meeting data set with varying shift value, validation set performance. Input features contain word embedding and POS tags for one word.

	Shift	0	1	2	3	4	5
F-measure	comma	4.07	32.46	22.55	0	0	0
	no punctuation	93.55	95.02	94.98	93.63	93.63	93.62
	full stop	16.87	30.70	39.25	0	0	0
	punctuation	29.26	58.78	58.18	0	0	0
	question mark	0	0	0	0	0	0
Precision	comma	11.76	30.28	27.80	0	0	0
	no punctuation	89.99	93.77	93.66	88.02	88.02	88.01
	full stop	21.59	37.21	37.98	0	0	0
	punctuation	51.75	66.05	65.92	0	0	0
	question mark	0	0	0	0	0	0
Recall	comma	2.46	34.98	18.97	0	0	0
	no punctuation	97.41	96.30	96.34	100.00	100.00	100.00
	full stop	13.85	26.12	40.61	0	0	0
	punctuation	20.39	52.96	52.07	0	0	0
	question mark	0	0	0	0	0	0

Table B.2.: Punctuation prediction on the meeting data set with varying shift value, validation set performance. Input features contain word embedding and POS tags for one word.

	Shift	0	1	2	3	4	5
F-measure	comma	0	32.50	7.12	0	0	0
	disfluency	43.99	53.33	53.27	0	0	0
	filler	76.29	79.15	80.26	0	0	0
	interruption	0	1.53	4.21	0	0	0
	non copy	0	0	0	0	0	0
	no disfluency	92.59	92.96	92.99	90.58	90.55	90.50
	no punctuation	94.03	95.25	95.01	93.58	93.58	93.62
	full stop	9.00	34.67	39.35	0	0	0
	punctuation	15.61	62.17	48.79	0	0	0
	question mark	0	0	0	0	0	0
	(rough) copy	0	30.68	23.60	0	0	0
Precision	comma	0	29.89	29.87	0	0	0
	disfluency	82.55	77.23	78.08	0	0	0
	filler	79.14	83.56	82.64	0	0	0
	interruption	0	13.64	29.63	0	0	0
	non copy	0	0	0	0	0	0
	no disfluency	87.19	88.81	88.77	82.78	82.73	82.65
	no punctuation	88.90	94.41	91.90	87.94	87.94	88.00
	full stop	25.81	38.75	41.87	0	0	0
	punctuation	84.52	66.88	74.78	0	0	0
	question mark	0	0	0	0	0	0
	(rough) copy	0	45.89	40.28	0	0	0
Recall	comma	0	35.62	4.04	0	0	0
	disfluency	29.98	40.72	40.42	0	0	0
	filler	73.65	75.18	78.02	0	0	0
	interruption	0	0.81	2.27	0	0	0
	non copy	0	0	0	0	0	0
	no disfluency	98.69	97.51	97.65	100	100	100
	no punctuation	99.79	96.10	98.34	100	100	100
	full stop	5.45	31.37	37.12	0	0	0
	punctuation	8.60	58.08	36.20	0	0	0
	question mark	0	0	0	0	0	0
	(rough) copy	0	23.04	16.69	0	0	0

Table B.3.: Punctuation prediction and disfluency detection on the meeting data set with varying shift value, validation set performance. Input features contain word embedding and POS tags for one word.

	Shift	0	1	2	3	4*	5*
F-measure	interruption	1.16	25.05	27.97	31.96	27.40	23.59
	coordinating conjunction	82.85	89.01	90.59	90.75	90.39	90.06
	discourse marker	85.13	94.12	94.58	94.36	94.01	93.56
	disfluency	71.49	83.14	85.91	85.74	84.61	83.93
	explicit editing	63.77	96.17	96.20	96.10	95.72	94.85
	filled pause	97.50	97.60	97.62	97.70	97.59	97.52
	reparandum	1.75	55.49	66.32	67.45	64.57	61.95
Precision	interruption	100.00	61.44	75.65	65.97	69.39	67.06
	coordinating conjunction	75.46	84.02	85.73	87.57	89.31	87.67
	discourse marker	82.10	93.45	94.19	94.64	95.49	94.49
	disfluency	85.72	89.29	90.30	88.44	88.34	88.48
	explicit editing	98.05	97.73	96.97	97.57	98.67	95.83
	filled pause	97.69	98.10	97.96	98.32	98.51	98.44
	reparandum	42.97	74.91	78.83	73.08	70.62	71.69
Recall	interruption	0.59	15.73	17.15	21.09	17.07	14.31
	coordinating conjunction	91.84	94.63	96.03	94.16	91.49	92.60
	discourse marker	88.38	94.79	94.97	94.09	92.58	92.66
	disfluency	61.31	77.79	81.93	83.20	81.18	79.83
	explicit editing	47.25	94.66	95.45	94.66	92.94	93.88
	filled pause	97.31	97.11	97.28	97.08	96.69	96.61
	reparandum	0.89	44.06	57.24	62.62	59.47	54.54

Table B.4.: Disfluency detection on the switchboard data set with varying shift value, validation set performance. Input features contain word embedding and POS tags for one word. *Training was run for 150 epochs with the goal to reach convergence. However, even with this prolonged training time the model was still improving.

	Shift	0	1	2	3	4	5
F-measure	comma	50.38	74.93	75.74	73.75	75.41	65.25
	incomplete slash-unit	7.97	34.53	37.42	35.94	31.64	0
	full stop	55.66	72.17	73.36	74.32	74.13	65.44
	punctuation	63.15	86.63	88.57	88.60	88.58	81.68
	question mark	30.03	59.12	62.23	59.90	55.95	0
	slash-unit	58.22	81.04	83.94	85.15	84.84	76.18
Precision	comma	64.83	67.62	68.27	63.88	68.42	57.26
	incomplete slash-unit	43.40	44.69	45.14	40.56	47.40	0
	full stop	62.30	71.68	71.97	74.39	73.78	68.80
	punctuation	64.08	84.15	86.51	85.55	87.16	79.71
	question mark	32.45	65.25	69.85	74.46	68.67	0
	slash-unit	47.19	75.95	78.95	82.11	81.21	71.70
Recall	comma	41.20	84.01	85.06	87.22	83.99	75.84
	incomplete slash-unit	4.39	28.13	31.95	32.27	23.74	0
	full stop	50.29	72.67	74.81	74.25	74.49	62.40
	punctuation	62.24	89.26	90.72	91.87	90.04	83.74
	question mark	27.95	54.04	56.11	50.10	47.20	0
	slash-unit	75.98	86.87	89.60	88.42	88.81	81.25

Table B.5.: Punctuation prediction on the switchboard data set with varying shift value, validation set performance. Input features contain word embedding and POS tags for one word.

	Shift	0	1	2	3	4	5
F-measure	interruption	0	10.06	17.52	39.93	14.04	0
	coordinating conjunction	81.91	90.17	90.80	90.99	91.08	71.17
	comma	47.94	77.38	78.90	77.72	78.07	67.76
	discourse marker	81.46	94.30	94.59	94.26	94.43	83.34
	disfluency	69.92	83.49	85.95	86.57	86.05	67.46
	explicit editing	63.70	96.38	96.35	96.23	96.10	3.80
	filled pause	97.49	97.83	97.72	97.69	97.71	96.73
	incomplete slash-unit	2.36	27.68	28.82	20.48	22.78	0
	full stop	50.75	67.20	68.91	64.05	67.53	53.75
	punctuation	55.25	85.89	88.41	88.00	87.58	79.35
	question mark	7.45	60.61	58.52	49.66	47.28	0
	reparandum	0.70	54.57	65.94	69.04	68.11	6.00
	slash-unit	54.71	80.60	85.00	84.66	84.56	74.51
Precision	interruption	0	85.19	87.41	64.84	81.51	0
	coordinating conjunction	75.45	88.18	88.59	90.13	90.09	79.29
	comma	83.82	76.96	77.87	75.17	78.02	74.98
	discourse marker	84.56	95.38	96.16	97.40	96.05	88.04
	disfluency	86.77	93.77	93.44	91.95	91.87	91.47
	explicit editing	97.67	98.52	97.92	98.04	98.01	92.86
	filled pause	97.70	98.84	98.55	98.33	98.55	96.92
	incomplete slash-unit	40.38	52.81	56.60	58.81	54.69	0
	full stop	74.98	80.03	82.15	86.30	83.14	79.47
	punctuation	87.73	88.29	90.45	90.88	90.98	89.30
	question mark	53.85	79.08	78.32	81.14	77.33	0
	reparandum	38.10	86.02	85.78	81.98	79.89	61.96
	slash-unit	78.89	81.48	84.66	88.28	87.66	83.30
Recall	interruption	0	5.34	9.74	28.85	7.68	0
	coordinating conjunction	89.58	92.25	93.12	91.86	92.10	64.55
	comma	33.57	77.81	79.97	80.45	78.12	61.80
	discourse marker	78.58	93.24	93.08	91.32	92.88	79.11
	disfluency	58.56	75.23	79.58	81.80	80.92	53.44
	explicit editing	47.26	94.33	94.83	94.48	94.26	1.94
	filled pause	97.28	96.84	96.90	97.06	96.89	96.53
	incomplete slash-unit	1.22	18.76	19.34	12.40	14.39	0
	full stop	38.36	57.91	59.34	50.92	56.85	40.61
	punctuation	40.32	83.62	86.47	85.30	84.42	71.39
	question mark	4.00	49.14	46.72	35.78	34.05	0
	reparandum	0.36	39.96	53.55	59.62	59.37	3.15
	slash-unit	41.87	79.73	85.35	81.32	81.68	67.39

Table B.6.: Punctuation prediction and disfluency detection on the switchboard data set with varying shift value, validation set performance. Input features contain word embedding and POS tags for one word.