Christopher Walles, BSc

Segmental Conditional Random Fields for Phone Recognition

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Assoc.Prof. Dipl.-Ing. Dr.mont. Franz Pernkopf

Signal Processing and Speech Communications Laboratory

Dipl.-Phys. Martin Ratajczak

Graz, October 2015

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis dissertation.

Date

Signature

Abstract

Automatic speech recognition (ASR) is a broad field of research. Applications of ASR include voice user interfaces, like those nowadays found in smart phones, automatic speech to text transcription or dialogue systems for people with impairments. An important sub-task of ASR is phone recognition. A phone recognition system detects the phones in a given speech signal. Speech data is segmental in nature, *i.e* each phone is represented by a variable number of input data frames. Usually, there are more input data frames than output labels. This thesis deals with segmental conditional random fields (SCRFs) to tackle the task of phone recognition. SCRFs are the segmental generalization of conditional random fields. The latter are the discriminative counterparts to hidden Markov models (HMMs). This thesis gives an overview of the formulas and algorithms required to apply the segmental conditional random field in practice. In particular we present an efficient dynamic programming algorithm that allows for training a SCRF on unsegmented data. In order to make the SCRF more powerful we equip it with a neural network style hidden layer. In the experiments, we apply segmental conditional random fields to the task of phone recognition and present results on the TIMIT database. We show that the algorithm to train a SCRF on unsegmented data achieves in practice the same results as when the model is trained on the manual segmented data from TIMIT. The presented model configuration of the hidden layer SCRF that was trained with backpropagation outperforms other published SCRF approaches and achieves a phone recognition accuracy of 75.07% on the TIMIT core test set.

Keywords: Machine Learning, Speech recognition, Phone recognition, Markov random fields, Segmental conditional random fields, TIMIT

Zusammenfassung

Die automatische Spracherkennung ist ein großes Forschungsgebiet, denn die Anwendungsmöglichkeiten automatischer Spracherkennung sind vielfältig. Dazu gehören zum Beispiel die Sprachsteuerung von Geräten, die heute in Smartphones zu finden ist, die automatische Konvertierung von Sprache in Text oder Dialogsysteme für Menschen mit Behinderungen. Ein wichtiges Teilgebiet der automatischen Spracherkennung ist die Lauterkennung (*engl.* phone recognition). Die die Sprache beschreibenden Daten sind segmentell. Konkret bedeutet dies, dass jeder Laut durch eine variable Anzahl von Merkmalvektoren definiert ist. Darum behandelt diese Arbeit „segmental conditional random fields“ als Ansatz zur Lösung des Lauterkennungsproblems. Dabei handelt es sich, mathematisch betrachtet, um die segmentelle Generalisierung von bedingten Markov-Netzwerken (*engl.* conditional random fields). Diese wiederum stellen das diskriminative Gegenstück zu den hidden-Markov Modellen dar. Diese Arbeit gibt einen ausführlichen Überblick über die Formeln und Algorithmen, die benötigt werden, um „segmental conditional random fields“ in der Praxis zu implementieren. Im Besonderen wird ein auf dynamischem Programmieren basierender effizienter Algorithmus vorgestellt, der es erlaubt, dieses Modell auf unsegmentierten Daten zu trainieren. Darüber hinaus wird das Modell durch eine Schicht von künstlichen Neuronen, wie sie in neuronalen Netzen üblich ist, erweitert. Es werden Lauterkennungsexperimente mit dem verbreiteten TIMIT Datensatz durchgeführt. Dabei wird unter anderem gezeigt, dass „segmental conditional random fields“, die mit der vorgestellte Methode ohne gegebene Segmentierungen trainiert werden, dieselbe Genauigkeit erreichen wie jene, die mit den gegebenen Segmentierungen trainiert werden. Das um eine Schicht von künstlichen Neuronen erweiterte Modell übertrifft andere Ansätze mit „segmental conditional random fields“ in der Klassifikationsgenauigkeit. Auf dem TIMIT Kerntestdatensatz wird 75.07% Genauigkeit erreicht.

Schlüsselwörter: Maschinelles Lernen, Spracherkennung, Lauterkennung, Markov-Netzwerke, segmental conditional random fields, TIMIT

Contents

Chapter 1

Introduction

Automatic speech recognition (ASR) is concerned with the automatic recognition of information embedded in a speech signal and its transcription in terms of a set of characters [4]. ASR has gone through a long and still ongoing history of research. Some example applications of ASR systems are voice user interfaces like those nowadays found in smart phones, dictation, and dialog systems for people with impairments.

The basic structure of an ASR system is depicted in Figure 1.1. The input to an ASR system is a speech signal in time domain representation. Many systems perform a preprocessing step in which the given input is transformed into a different representation. This step is also known as feature extraction. Common preprocessing steps compute spectrograms [7] or cepstral coefficents. Two widely used cepstral representations are Mel-Frequency Cepstral Coefficients (MFCCs) [20], [9], [17] or Perceptual Linear Prediction (PLP) [13] coefficients. Both types of preprocessing are resulting in creating a certain number of preprocessed vectors per second of speech. The preprocessed data is then fed into the actual speech recogniser.

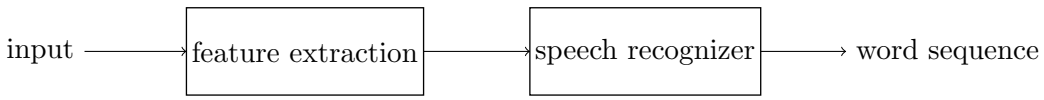


Figure 1.1: Basic structure of an automatic speech recognition (ASR) system. The input of the system is a speech signal in time domain representation. The first step of many systems is a preprocessing of the input signal. Subsequently the actual recognition is performed on the preprocessed data. The resulting output of an ASR system is a sequence of words.

A branch of automatic speech recognition research deals with phones. Those systems attempt to detect phones instead of words in the given input. Hence, in contrast to complete ASR systems, phone based systems do not rely on a dictionary. Furthermore if a language model is used, it is base on phones instead of words. There are three tasks involving phones: phone segmentation, phone classification and phone recognition. The first task, phone segmentation, is about detecting the phone boundaries for a given utterance and the phonetic transcription. This corresponds to detecting the black vertical lines in Figure 1.2. Phone classification is the task of determining the phonetic transcription of a segmented speech signal. Phone recognition is the hardest of the three tasks. It is the process of determining the phonetic transcription from a given unsegmented utterance. This corresponds to

determining both the transcription of the utterance and the segment boundaries.

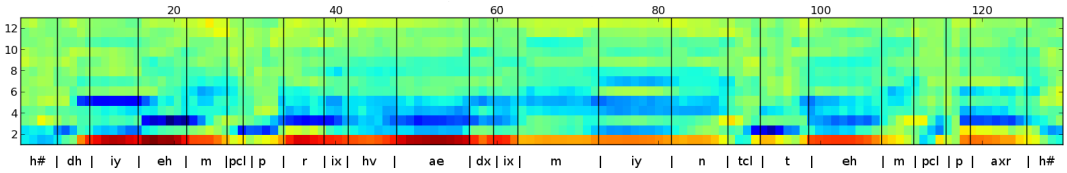


Figure 1.2: MFCC representation of a random utterance from the TIMIT speech corpus [5]. The phone boundaries are indicated by the vertical lines in black. The bottom line below the cepstrum is the aligned phonetic transcription of the utterance.

Phone classification is a task at frame-level. For each frame of acoustic features one phone is determined. In contrast, phone recognition is a segmental task as there is a variable number of feature frames per phone.

The TIMIT Acoustic-Phonetic Continuous Speech Corpus [5] is a popular dataset for training and comparing phone recognition systems. The dataset comprises 6300 American English sentences spoken by 630 different speakers. The dataset does not only contain the phonetic transcriptions, but also the positions of the boundaries. With regard to the TIMIT speech corpus, there are two types of phone recognition approaches. Some systems (have to) rely on the positions of the phone boundaries at training time. For example the phone recognition system in [10] was trained using the phone boundaries. Other systems do not require segmented training data at all. For instance the approach presented in [7]. However, all approaches known as phone recognition systems are capable of making predictions on the unsegmented data.

With the segmental conditional random fields this thesis pursues a segmental, discriminative, graphical model approach to tackle the problem of phone recognition. This model considers for a given utterance all possible segmentations in an efficient way. For each segment a potential is computed. The probability of a segmentation is proportional to the product of the segment potentials. The classification result of the model is the most probable segmentation. The phone recognition experiments presented in this thesis were conducted with the TIMIT dataset.

1.1 Contributions and Outline

In Chapter 2 we give an overview of conditional random fields (CRFs) which are discriminative, graphical, sequence models. We discuss the differences to hidden Markov models (HMMs) which are generative models. We further discuss why CRFs as frame level models are not well suited for phone recognition.

Therefore in Chapter 3 segmental conditional random fields (SCRFs) are introduced. Although the segmental conditional random field or Semi-Markov CRF has already been defined in [18] we present a comprehensive derivation and an intuitive explanation of the recursions and the involved algorithms. All necessary equations for training the model are derived. This is usually neglected in research papers. The presented sub-set of equations does not provide sufficient insight to immediately understand and implement the model. To the best of the author's knowledge there is not yet such a comprehensive summary of the needed formulas. Furthermore we present a method to train a SCRF without the need of given phone boundary

positions. Although [20], [1] have already trained SCRFs without given alignment, they did not present how to do this in a tractable way. We conclude Chapter 3 by comparing the segmental conditional random field approach to connectionist temporal classification (CTC). CTC is a method for using recurrent neural networks for segmental data.

In Chapter 4 we apply the segmental conditional random field to the task of phone recognition. We also present recognition results of a SCRF with a neural network style hidden layer, trained with backpropagation. With a TIMIT core test set accuracy of 75.07% this extension of the SCRF model outperforms two other SCRF approaches presented in [20] and in [10]. Furthermore we show that training the SCRF without given phone boundaries it is possible to achieve the same performance as with given boundaries. Finally we conclude the thesis in Chapter 5 by comparing the obtained results with other phone recognition approaches trained and evaluated on TIMIT and give directions of future work.

Chapter 2

Conditional random fields (CRFs)

In this section, we first introduce the notation of sequential data. Then we summarize and compare two popular models for sequential data, *i.e.* we discuss the similarities and differences between hidden Markov models (HMMs) and conditional random fields (CRFs). Finally, we conclude the section by motivating a segmental approach for phone recognition and point out related work of this thesis.

2.1 Sequential data

Simple classifiers, such as the logistic regression, assume that data samples are independent of each other and identically distributed (*i.i.d*). In contrast, sequential data exhibits correlation between individual data samples. This is an observation that is in particular true for speech data. Hence, for sequential data the *i.i.d* assumption will be a poor approximation. Examples for tasks involving sequential data are speech recognition, natural language modelling, or handwriting recognition. [3, page 605]

Let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$ be a sequence of T observed input vectors. For example consider the task of phone classification. Each $\mathbf{x}_t$ corresponds to some features representing a phone. In phone recognition each $\mathbf{x}_t$ corresponds to one time slice of preprocessed data (*e.g.* one MFCC frame).

In the task of phonetic classification one wants to determine for $\mathbf{x}_t$ the corresponding phone or label y_t. Each y_t is drawn from a finite set, called the state space or the label alphabet $\mathcal{Y}$. In phone classification or recognition $\mathcal{Y}$ is the set of all distinct phones. Hence, $\mathbf{Y} = (y_1, \ldots, y_T)$ denotes the state or labelling sequence.

2.2 The hidden Markov model (HMM)

A popular model for dealing with sequential data is the hidden Markov model (HMM). It models an observations sequence $\mathbf{X}$ under the assumption that there is some underlying, hidden state sequence $\mathbf{Y}$. The HMM models the joint distribution of inputs and states $p(\mathbf{X}, \mathbf{Y})$ [3, pages 610 ff.].

In order to make the calculations involved tractable, the HMM makes two independence assumptions. First, the current state does not depend on older states

except the previous one. This is known as the Markov assumption or Markov property. Second, each observation $\mathbf{x}_t$ does only depend on the current state y_t.

The probabilistic model of the HMM is specified by three distinct probability distributions: the initial state probabilities $p(y_1)$, the transitions probabilities $p(y_t|y_{t-1})$ and the emission probabilities of states $p(\mathbf{x}_t|y_t)$.

The joint probability of an observation and a state sequence in a HMM is given by:

$$p(\mathbf{X}, \mathbf{Y}) = p(y_1)p(\mathbf{x}_1|y_1) \prod_{t=2}^{T} p(y_t|y_{t-1})p(\mathbf{x}_t|y_t). \tag{2.1}$$

Models, such as the HMM, that capture the distribution of inputs and outputs are known as *generative* models. This enables to sample synthetic data from the model. In a generative model the class-conditional densities $p(\mathbf{x}|y)$ and the prior probabilities $p(y)$ of each class form the joint probability $p(\mathbf{x}, y)$. The posterior class probability $p(y|\mathbf{x})$ is then given by applying Bayes' theorem. In contrast, a *discriminative* approach directly models $p(y|\mathbf{x})$ [3, page 43].

If a generative and a discriminative model have a common underlying parametric family of probabilistic models, they are called a generative-discriminative pair [14]. An example of such a pair is naive Bayes and logistic regression. In [14] it is shown that given a sufficient amount of training data the discriminative logistic regression usually performs better in classification than the generate counterpart naive Bayes. However the generative model was found to converge faster and to perform better if the number of training examples is small. In any case a generative model tries to capture the joint distribution of $\mathbf{x}$ and y. However, this is not required in the task of classification, where we are only interested in obtaining the most probable estimate of y.

2.3 Conditional random fields

This section gives a brief overview of conditional random fields (CRFs), the discriminative counterparts of HMMs. A CRF models the conditional probability of a labelling (state) sequence $\mathbf{Y}$ given an observation sequence $\mathbf{X}$ [11];

$$p(\mathbf{Y}|\mathbf{X}) = \frac{1}{Z(\mathbf{X})} \prod_{t=1}^{T} \Phi(y_t, y_{t-1}, \mathbf{X}, t), \tag{2.2}$$

where $\Phi(y_t, y_{t-1}, \mathbf{X}, t)$ is the potential function and $Z(\mathbf{X})$ the partition function. The potential function is given in exponential form:

$$\Phi(y_t, y_{t-1}, \mathbf{X}, t) = \exp \psi(y_t, y_{t-1}, \mathbf{X}, t) \cdot \exp \phi(y_t, \mathbf{X}, t), \tag{2.3}$$

where $\psi(y_t, y_{t-1}, \mathbf{X}, t)$ is the logarithm of the transition factor or potential and $\phi(y_t, \mathbf{X}, t)$ the logarithm of the state factor.

The partition function of the conditional random field is defined as the sum over all possible labelling sequences and thus ensures that the distribution is correctly normalized, *i.e.*

$$Z(\mathbf{X}) = \sum_{\mathbf{Y}} \prod_{t=1}^{T} \exp \Phi(y_t, y_{t-1}, \mathbf{X}, t). \tag{2.4}$$

The logarithmic factors $\psi(y_t, y_{t-1}, \mathbf{X}, t)$ and $\phi(y_t, \mathbf{X}, t)$ are defined as

$$\phi(y_t, \mathbf{X}, t) = \sum_y \left(\mathbf{w}_y^T \mathbf{f}(\mathbf{X}, t) + \lambda_y\right) \delta_{yy_t} \tag{2.5}$$

$$\psi(y_t, y_{t-1}, \mathbf{X}, t) = \sum_{y,y'} (\mathbf{v}_{y,y'}^T \mathbf{g}(\mathbf{X}, t) + \mu_{y'y}) \delta_{yy_t} \delta_{y'y_{t-1}}, \tag{2.6}$$

where $\mathbf{w}_y$ and $\mathbf{v}_{y,y'}$ are the observation dependent parameters and λ_y and μ are observation independent "bias" weights. δ_{ij} denotes the Kronecker delta. $\mathbf{f}$ and $\mathbf{g}$ are the features or feature functions of the CRF. The feature functions are notated in bold to indicate that they return vectors. The definition of the feature functions is arbitrary and specific to the task at hand. For instance in a natural language processing task one dimension of a feature function might be defined as a boolean feature that returns one if the word $\mathbf{x}_t$ is capitalized. In phone classification a feature function might be the identity function returning the observation vector $\mathbf{x}_t$, which contains some acoustic data of the corresponding phone.

As already pointed out above, the HMM is generative model and thus needs to model the distribution of the inputs. In order to keep the computations of the model tractable, it is for many applications necessary to make independence assumptions among the inputs. The assumption made by HMMs is that each observation $\mathbf{x}_t$ has exactly one underlying hidden state it depends on. Because of the fact that the CRF is a discriminative model there is no need to make restricting assumptions on the inputs. Therefore the input context of the feature functions in a CRF is arbitrary, and even may range up to the whole sequence. This a huge advantage compared to HMMs.

However, the Markov assumption, which states that the current state or label does only depend on a limited number of previous states, is still held for CRFs. In practice higher order dependences among the states are made tractable by considering not all combinatorial possibilities [15], [19].

Although maximum entropy Markov models and discriminative Markov models are also discriminative models and thus also share the aforementioned benefits over HMMs, they have a weakness known as the label bias problem. Those models perform a per state normalization, which causes the transitions leaving a given state to compete only against each other, instead of competing against all other model transitions. This leads to the problem that the probability mass that arrives at a state has to be passed onto the next state completely. The observations do have an influence on where to pass the probability mass on, but not how much of it. This introduces a bias towards states with fewer outgoing transitions. CRFs are not affected by the label bias problem because they compute the normalization over the whole sequence. [11]

Since a CRF is a Markov random field, it can be expressed and depicted as a factor graph as shown in Figure 2.1. In the visualization the state potential function $\phi(y_t, \mathbf{X}, t)$ only depends on the current input and the transition potential function $\psi(y_t, y_{t-1}, \mathbf{X}, t)$ does not depend on any input.

The partition function of the CRF is efficiently computed by the forward recursion:

$$Z(\mathbf{X}) = \sum_y \alpha_T(y) \tag{2.7}$$

$$\alpha_t(y) = \sum_{y'} \Phi(y, y', \mathbf{X}, t) \alpha_{t-1}(y') \tag{2.8}$$

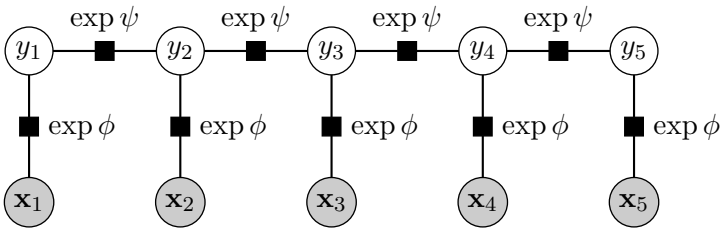


Figure 2.1: Factor graph representation of a conditional random field (CRF) as defined by Lafferty, McCallum and Pereira in [11] for an observation sequence of length 5. The state factor does only depend on the current observation and the transition factor is independent of the observation sequence.

The parameters are usually estimated by maximizing the conditional log likelihood.

2.4 Related Work

CRFs predict for each observation vector $\mathbf{x}_t$ one label y_t. That means that they are frame-level models. In phone recognition the number of labels is by far smaller than the number of observations. Thus CRFs are not off the shelve capable of handling segmental data. However, it is possible to use CRFs for segmental data with some restrictions. In [13], Morris and Fosler-Lussier applied CRFs to the task of phone recognition. During training they had to rely on the phone boundaries from the TIMIT dataset. When performing classification on a given observations sequence the CRF creates frame level output. This output was post-processed by merging all adjacent outputs of the same class. In this thesis we explore segmental conditional random fields (SCRFs), in which a state comprises a variable number of observations. Thus SCRFs relax the Markov assumption from frame-level to state-level. The transitions within segments are arbitrary. [21] Segmental conditional random fields are inherently capable of handling segmental data.

Since in phone recognition the number of states or labels is much smaller than the number of observations many phone recognition systems rely on HMMs to provide a forced alignment of the labelling sequence. In fact, those approaches are hybrid ones consisting of two distinct models. For example in [2] a convolutional neural network was combined with a HMM. Another example is the combination of deep belief networks with HMMs in [12]. Since those systems involve HMMs, they also exhibit the above discussed disadvantages of HMMs. The segmental conditional random field approach is a non-hybrid model for handling segmental data directly.

Segmental conditional random fields have already been applied to phone recognition. In [20], Zweig has used a SCRF in connection with vector quantization features. In [10], He and Fosler-Lussier used phone posteriors as features for a SCRF. The phone posteriors were obtained from a separately trained neural network. Thus this approach is again a hybrid one and not suited of being trained without boundary information. In this work we explore the direct utilization of MFCCs in a SCRF. Moreover we extend the SCRF, which is a linear model, by a non-linear hidden layer and train the parameters jointly with backpropagation.

Another recent approach to handle segmental data is called connectionist temporal classification (CTC), which is an extension for recurrent neural networks. After

introducing the segmental conditional random field we discuss the differences to this particular approach.

Chapter 3

Segmental conditional random fields (SCRFs)

In this chapter we introduce the segmental conditional random field and derive all formulas required to apply it in practice. That involves making predictions and learning the parameters using gradient ascent.

3.1 Segmental data

In this section we briefly introduce the mathematical notation required to deal with segmental data. As in the previous chapter let $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)$ be the observation sequence consisting of T observation vectors.

Let $\mathbf{S} = (s_1, \ldots, s_n)$ denote a sequence of n segments. Each segment of $\mathbf{S}$ is defined as a triple: $s_i = (y_i, t_i, u_i)$. y_i denotes the label or state of s_i. t_i and u_i are the indices of the first observation and respectively the last observation of the segment s_i. The length or duration of s_i is given by $l_i = u_i - t_i + 1$. Figure 3.1 blow gives a graphical illustration of an example segmentation.

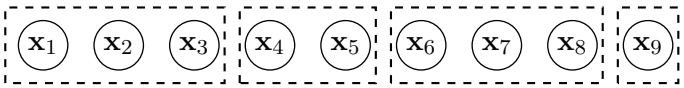


Figure 3.1: Illustrating example: an observation sequence $\mathbf{X}$ consisting of 9 observations is split into 4 segments. The segments are: $s_1 = (y_1, 1, 3), s_2 = (y_2, 4, 5), s_3 = (y_3, 6, 8), s_4 = (y_4, 9, 9)$.

Each segment comprises at least one observation. Here we will additionally constraint each segment to have a maximum length of $L_{\max} \in \mathbb{N}^*$. As shown later in Section 3.4, this maximum length constraint keeps the runtime of the model reasonable.

In a segment sequence $\mathbf{S}$ adjacent segments always touch. That means that the start index of segment s_i is given by the end position of the previous segment s_{i-1} plus one. In any segment sequence the first segment always starts at position $t_1 = 1$. For t_i we have:

$$t_i = \begin{cases} u_{i-1} + 1 & \text{for } 2 \leq i \leq n \\ 1 & \text{if } i = 1 \end{cases}. \tag{3.1}$$

A segment sequence $\mathbf{S}$ is considered to be consistent with an observation sequence

$\mathbf{X}$ if the last segment s_n ends exactly at $u_n = T$. For each $\mathbf{X}$ there are many different consistent segment sequences $\mathbf{S}$ (*c.f* Figure 3.2).

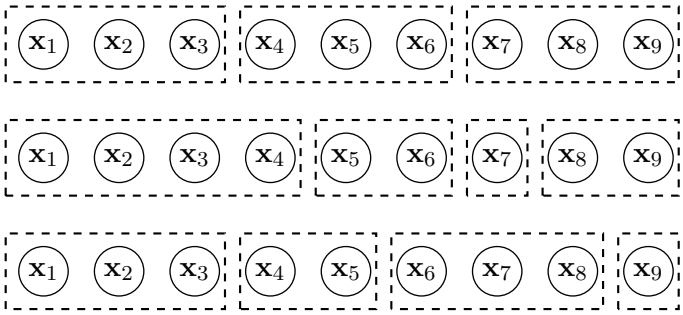


Figure 3.2: Some example segmentations of the observation sequence $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_9)$. Note that the segmentation in the top splits $\mathbf{X}$ into 3 segments, while the two examples below split $\mathbf{X}$ into 4 segments.

A segment s carries two distinct informations: the labelling and the position information. Therefore we split $s = (y, t, u)$ into the label y and the position part $a = (t, u)$. In the latter the a stands for alignment. The segmentation of $\mathbf{X}$, that is the start and end position of the segments, is uniquely defined by the sequence of alignments $\mathbf{A} = (a_1, \ldots, a_n)$. The label sequence $\mathbf{Y} = (y_1, \ldots, y_n)$ is always of the same length as the corresponding alignment sequence $\mathbf{A}$.

3.2 SCRF model

A segmental conditional random field (SCRF) [21, 10], also known as semi-Markov CRF [18], is a Markov random field [3, page 386] or undirected graphical model in which each state comprises a variable number of observations. Thus the SCRF is a segmental model. This is a generalization of an ordinary CRF where each state corresponds to exactly one observation. The SCRF models the conditional probability of a segment sequence $\mathbf{S}$ for a given observation sequence $\mathbf{X}$ (*c.f* Equation (3.2)). A first order Markov assumptions is made for the states, *i.e.* there is a dependency between two adjacent labels.

$$p(\mathbf{S}|\mathbf{X}) = p(\mathbf{Y}, \mathbf{A}|\mathbf{X}) = \frac{1}{Z(\mathbf{X})} \prod_{i=1}^{|\mathbf{S}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i) \tag{3.2}$$

The normalization term $Z(\mathbf{X})$ or also known as the partition function [3, page 386] ensures that the conditional distribution is correctly normalized. Therefore $Z(\mathbf{X})$ is the sum over all segment sequences $\mathbf{S}$ consistent with the given observation sequence $\mathbf{X}$, *i.e.*

$$Z(\mathbf{X}) = \sum_{\mathbf{S}} \prod_{i=1}^{|\mathbf{S}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i) = \sum_{\mathbf{A}} \sum_{\mathbf{Y}} \prod_{i=1}^{|\mathbf{A}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i). \tag{3.3}$$

Each Markov random field can be expressed as a product of factors over its variables [3, page 399]. A factor graph makes this decomposition explicit and therefore it allows a more intuitive representation. Figure 3.3 depicts a factor graph representation of a SCRF.

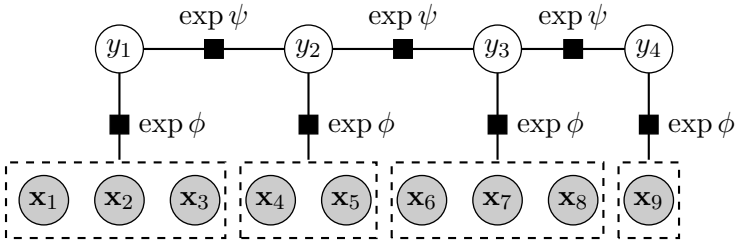


Figure 3.3: Factor graph representation of an example SCRF for $T = 9$ given observations partitioned into $n = 4$ segments. The state factor $\exp \phi$ depends on the current segment, while the transition factor $\exp \psi$ shown here is independent of any input.

The factor graph depicted in Figure 3.3 shows that there are two types of factors. There is one factor that depends on one state and the observations of the corresponding segment. In this work we call it the *state* factor. The other factor depends on two states and is thus called the *transition* factor. In this example the transition factor is independent of any input and thus only acts as a bias. However, it is possible that the latter also depends on observations. Nevertheless, the transition factor does not depend on the length of the corresponding segment. This important detail will later allow a more efficient way of dealing with the model (*c.f* Section 3.8).

The potential function is defined as product of the state and the transition factors, *i.e.*

$$\Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i) = \exp \psi(y_i, y_{i-1}, \mathbf{X}, t_i) \cdot \exp \phi(y_i, \mathbf{X}, t_i, u_i), \tag{3.4}$$

where $\psi(y_i, y_{i-1}, \mathbf{X}, t_i)$ represents the logarithm of the transition and $\phi(y_i, \mathbf{X}, t_i, u_i)$ the logarithm of the state factor.

Note that a transition factor does not exist for the first segment in any segmentation. For convenience we define the transition factor for the first segment to be one. Therefore we define the potential of the first segment as

$$\Phi(y_1, y_0, \mathbf{X}, t_1, u_1) = \exp \phi(y_1, \mathbf{X}, t_1, u_1).$$

Note that y_0 does not exist, it is introduced to simplify the notation.

The logarithm of the two factors is defined in the following ways:

$$\psi(y_i, y_{i-1}, \mathbf{X}, t_i) = \sum_{y,y'} \left(\mathbf{v}_{y'y}^T \mathbf{g}(\mathbf{X}, t_i) + \mu_{y'y} \right) \delta_{yy_i} \delta_{y'y_{i-1}}, \tag{3.5}$$

$$\phi(y_i, \mathbf{X}, t_i, u_i) = \sum_{y} \left(\mathbf{w}_y^T \mathbf{f}(\mathbf{X}, t_i, u_i) + \lambda_y \right) \delta_{yy_i}. \tag{3.6}$$

In the equations above δ_{ij} represents the Kronecker delta. λ and μ are the input independent bias parameters. The features functions $\mathbf{f}$ and $\mathbf{g}$ return vectors of data created from the given observations. The definition of the functions is again arbitrary. The state feature function $\mathbf{f}$ needs to create a fixed size vector independent of the current segment length. Clearly it is important that the vectors returned by the feature functions match with the dimensions of the parameter $\mathbf{w}$ and $\mathbf{v}$ respectively. In case a feature function returns the zero vector the corresponding factor will only depend on the bias parameter. There are $|\mathcal{Y}|$ different λ and $\mathbf{w}$ and $|\mathcal{Y}|^2$ different μ and $\mathbf{v}$ parameters in the model.

3.3 Enumeration of all segmentations

In this section we develop a recursive algorithm for enumerating all segmentations of a sequence of T observations. This algorithm will help to derive an efficient method for computing $Z(\mathbf{X})$ in the next section.

Let $\mathbf{L}_t = (l_1, \ldots, l_n)$ be a sequence of segment lengths, such that the lengths sum up to t and each length l satisfies the constraint $1 \leq l \leq L_{\max}$. Note that $\mathbf{L}_t$ can be converted into an alignment sequence $\mathbf{A}$ in a very simple way. The alignment $a_i = (t_i, u_i)$ of the i-th segment is given by $t_i = 1 + \sum_{j=1}^{i-1} l_j$ and $u_i = t_i + l_i - 1 = \sum_{j=1}^{i} l_j$.

Furthermore let $\mathcal{L}_t$ denote the set of all segmentations of t observations. In other words $\mathcal{L}_t$ contains all possible realizations of $\mathbf{L}_t$. The pseudo code of Algorithm 1 depicts how to compute $\mathcal{L}_t$.

Algorithm 1 Algorithm to enumerate all segmentations that comprise exactly t observations and each segment length is constrained to a maximum length of $L_{\max}$.

```
function ALL_SEGMENTATIONS(t, L_max)
    L_t ← {}
    if t ≤ L_max then
        L_t ← L_t ∪ {t}
    end if
    L ← MIN(t − 1, L_max)
    for l in {1, ..., L} do
        L_{t−l} ← ALL_SEGMENTATIONS(t − l, L_max)
        for all L_{t−l} in L_{t−l} do
            L_t ← L_t ∪ {(L_{t−l}, l)}
        end for
    end for
    return L_t
end function
```

If $t > L_{\max}$, $\mathcal{L}_t$ is recursively computed by extending all existing enumerations by one new segment. Before extending an existing enumeration it needs to be assured that the constraints will also hold after the extension. The maximum length constraint for a segment is satisfied by just inserting segments of length l, with $1 \leq l \leq L_{\max}$. Therefore all previous $L_{\max}$ sets $\mathcal{L}_{t-l}$ are extended by one new segment of length l to form $\mathcal{L}_t$.

That means that all segmentations in $\mathcal{L}_{t-1}$ are extended by a segment of length 1. The segmentations in $\mathcal{L}_{t-2}$ are extended by a segment of length 2 and so forth... If the sets $\mathcal{L}_{t-1}, \ldots, \mathcal{L}_{t-L_{\max}}$ contain all segmentations of $t-1, \ldots, t-L_{\max}$ observations, then by extending each of them by a segment of a specific length l, such that the segment lengths will sum to t, one obtains all segmentations of t observations $\mathcal{L}_t$.

It remains to show that the first $L_{\max}$ sets contain all possible segmentations. For an observations sequence of length 1 there is always exactly one possible segmentation: $\mathcal{L}_1 = \{(1)\}$. For an observation sequence of length 2 one possible segmentation is the extension of the previous segmentation by one segment of length 1, yielding $L_2 = (1, 1)$. If $L_{\max} \geq 2$ then $L_2 = (2)$ is also a possible segmentation of length 2. There are no other possible segmentations of length 2. Therefore: $\mathcal{L}_2 = \{(1, 1), (2)\}$, iff $L_{\max} \geq 2$.

For $t = 3$ it can be continued in the same way: Recall that $\mathcal{L}_2$ contains all possible segmentations of length 2 and that each segment comprises at least 1 observation.

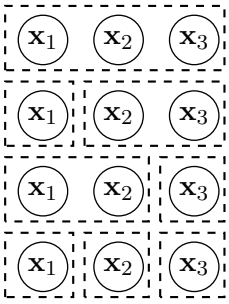


Figure 3.4: Illustration of the algorithm for enumerating all segmentations. All segmentations comprising 3 observations, for $L_{\max} \geq 3$. The segmentations are created from all segmentations of 2 observations by inserting a segment of length 1, all segmentations of 1 observation by inserting a segment of length 2 and a single segment of length 3.

To obtain segmentations of length 3 we can extend all segmentations in $\mathcal{L}_2$ by a segment of length 1. If $L_{\max} \geq 2$ all segmentations in $\mathcal{L}_1$ can be extended by a segment of length 2 to yield segmentations of length 3. If $L_{\max} \geq 3$ we additionally need to add $L_3 = (3)$. For $L_{\max} \geq 3$, $\mathcal{L}_3 = \{(1,1,1),(2,1),(1,2),(3)\}$. Figure 3.4 depicts a graphical example of this case.

It appears that for $t \leq L_{\max}$ the same procedure to generate all segmentations from existing ones can be applied. Additionally there is always a new segmentation consisting of only one single segment of length t. This one also needs to be added to the set in order to obtain all segmentations.

3.4 Computation of partition function - forward recursion

In this section we develop an efficient algorithm for computing the partition function $Z(\mathbf{X})$. The partition function of the SCRF is given in Equation (3.3).

The summation over all segment sequences $\mathbf{S}$, consistent with the given observation sequence, is also given by the sum over all alignment sequences $\mathbf{A}$, and for each $\mathbf{A}$ the sum over all labelling sequences $\mathbf{Y}$ that are consistent with $\mathbf{A}$ (*cf.* Equation (3.3)). Recall that each $\mathbf{A}$ can be expressed as a sequence of segment lengths $\mathbf{L}$ and vice versa. The set of all segment length sequences of T observations is given by the above introduced set $\mathcal{L}_T$.

A naive approach for computing $Z(\mathbf{X})$ would be to first compute $\mathcal{L}_T$ and then for each $L \in \mathcal{L}_T$ sum over all possible labelling sequences $\mathbf{Y}$. However this approach is problematic for two reasons. First, the number of elements in the set $\mathcal{L}_T$ grows exponentially in the number of observations.[1] Second, the summation over all possible label assignments does not consider the conditional independence property of the model. Thus the summation over all $\mathbf{Y}$, that have the same length as the current $\mathbf{A}$ or $\mathbf{L}$, also runs exponential in the length of $\mathbf{Y}$.

However there is no need to explicitly enumerate all segmentations of T observations. It is sufficient to get the sum over the scores of all segment sequences $\mathbf{S}$.

[1] A lower bound for the number of segmentations is obtained if $L_{\max} = 2$. Then the number of segmentations for T observations corresponds to the T-th Fibonacci number. The closed-form expression due to Moivre/Binet makes it clear that Fibonacci numbers increase exponentially. Therefore also the number of segmentations increases exponentially with T.

The score of a segment sequence $\mathbf{S}$ is given by the product over all potential functions. The algorithm to enumerate all segmentations, which we introduced in the previous section, shows that one can generate all segmentations of t observations by extending the previous enumerations by a new segment. But instead of keeping a set of enumerations we keep the sum over the scores of the different segmentations. Following the recursive principle of the algorithm we introduce a new quantity:

$\alpha_t(y)$ is the sum over all segment sequences of the first t observations of $\mathbf{X}$ where the last segment is labelled y. The constraint that a segment has the maximum length $L_{\max}$ has to be considered. Following that definition, $\alpha_T(y)$ represents the sum over all segment sequence scores that end with a segment labelled y. Hence the value of the partition function is given by summing out all possibilities of y for the last segment, *i.e.*

$$Z(\mathbf{X}) = \sum_{y} \alpha_T(y). \tag{3.7}$$

Next we show how $\alpha_t(y)$ can be computed recursively. Following the argumentation of the algorithm for enumerating all segmentations, we obtain the sum over the scores of segment sequences comprising the first t observations by multiplying the l previous sums by the potential of a new segment of length l starting at $t-l+1$ and ending at t. In order to meet the maximum segment length constraint, we only extend up the $L_{\max}$ previous summations. Moreover the dependency between two adjacent labels needs to be considered. That means that the potential of the new inserted segment depends on the label of the previous segment. Therefore we kept the label of the last segment explicit (that means that $\alpha_t(y)$ has the parameter y). In order to obtain the value of the sum of all segment sequences, we need to sum over all possibilities for the previous label whenever a new potential is added.

These considerations are giving the equation for determining $\alpha_t(y)$ recursively:

$$\alpha_t(y) = \sum_{l=1}^{L_{\max}} \sum_{y'} \alpha_{t-l}(y') \Phi(y, y', \mathbf{X}, t-l+1, t). \tag{3.8}$$

For convenience we define $\alpha_0(y) = 1$ and $\forall t < 0 : \ \alpha_t(y) = 0$. The forward recursion formula derived here is in agreement with the recursion presented in [18].

Let M be the number of feature function evaluations required to compute the potential function. The recursion is computed for each observation. In each recursion step, the potential functions of up to $L_{\max}$ new segments need to be computed. Each potential function depends on two labels. Thus in one recursion step $\mathcal{O}(L_{\max} \cdot |\mathcal{Y}|^2)$ potential function evaluations are required. Therefore the time complexity of computing the partition function is $\mathcal{O}(T \cdot L_{\max} \cdot |\mathcal{Y}|^2 \cdot M)$. When saving all intermediate values of the forward recursion the memory complexity is $\Theta(T \cdot |\mathcal{Y}|)$.

At this point it becomes clear why we have introduced a maximum segment length constraint. If a segment could have arbitrary length, the recursion would be required to run back to the first observation each step. That would mean that the runtime of the recursion is quadratic in the number of observations. This would lead to an extensive runtime requirement.

A property that becomes apparent when comparing the forward recursion of the SCRF in Equation (3.8) with the forward recursion of the CRF in Equation (2.8) is that if $L_{\max} = 1$ the SCRF is the same as a CRF. Thus the segmental conditional random field is a generalization of the conditional random field.

3.5 The backward recursion

Using the same principle as above, but going from the last to the first observation, the backward recursion can be derived. $\beta_t(y)$ is the sum of the scores of all segment sequences of the observations t (inclusive) to the end of the observation sequence $\mathbf{X}$ where the preceding segment is labelled as y. Note that the potential of the segment that has label y is not yet added to the sum!

Per definition the first segment does not have a preceding one and therefore $\beta_t(y)$ is only defined for $2 \leq t \leq T$. The backward recursion is required to compute the partial derivative of the log likelihood function with respect to the model parameters (*cf.* Section 3.7).
The backward recursion of the SCRF is given in Equation (3.9).

$$\beta_t(y) = \sum_{l=1}^{L_{max}} \sum_{y'} \beta_{t+l}(y') \Phi(y', y, \mathbf{X}, t, t+l-1), \quad 2 \leq t \leq T \tag{3.9}$$

For convenience we define $\beta_{T+1}(y) = 1$ and $\forall t > T+1 : \ \beta_t(y) = 0$. This is again in agreement with the equation given in other work [10], however the index t there is off by one.

Obviously the partition function can also be computed using the backward recursion:

$$Z(\mathbf{X}) = \sum_{y} \sum_{l=1}^{L_{\max}} \Phi(y, \mathbf{X}, 1, l) \beta_{1+l}(y) = \sum_{y} \alpha_T(y). \tag{3.10}$$

The potentials of all initial segments starting at position 1 need to be taken into account here, since those have not been considered in the recursion yet.
The same time and memory complexities as for the forward recursion apply.

3.6 Most probable segment sequence

This section describes how to compute the most probable segment sequence $\mathbf{S}^*$ for the given observation sequence under the current parameter configuration, *i.e.*

$$\mathbf{S}^* = \arg\max_{\mathbf{S}} p(\mathbf{S}|\mathbf{X}). \tag{3.11}$$

A formula for efficiently computing $\mathbf{S}^*$ can be derived in a similar way as the forward recursion for efficiently computing the partition function $Z(\mathbf{X})$ was derived. The argumentation works the same way, but instead of summation maximization is applied. This can be seen as a extension of the Viterbi algorithm to segmental data [3, pages 415, 629]. The forward step of the Viterbi algorithm for segmental condition random fields is given by:

$$\hat{\alpha}_t(y) = \begin{cases} \max_{l'} \max_{y'} \hat{\alpha}_{t-l'}(y') \Phi(y, y', \mathbf{X}, t-l'+1, t) & \text{if } t \geq 1 \\ 0 & \text{if } t = 0 \\ -\infty & \text{if } t < 0 \end{cases} \tag{3.12}$$

In order to obtain the most likely label sequence $\mathbf{Y}^*$ and the corresponding alignment $\mathbf{A}^*$ one needs to perform a backtracking step. In practice one wants to compute the quantities for the backtracking step at the same time as the forward recursion is computed.

Let $\mathbf{S}_t^*$ be the most likely sequence of segments comprising the first t observations and the last segment is labelled y. Then the quantity $\hat{l}_t(y)$ stores the length of the last segment of $\mathbf{S}_t^*$:

$$\hat{l}_t(y) = \arg\max_{l'} \max_{y'} \hat{\alpha}_{t-l'}(y')\Phi(y, y', \mathbf{X}, t-l'+1, t), \quad 1 \leq t \leq T. \tag{3.13}$$

Let $\mathbf{S}_t^*$ be the most likely sequence of segments comprising the first t observations and the last segment is labelled y and has length l. Then the quantity $\hat{y}_t(y, l)$ stores the value of the next to last segment of $\mathbf{S}_t^*$:

$$\hat{y}_t(y, l) = \arg\max_{y'} \hat{\alpha}_{t-l}(y')\Phi(y, y', \mathbf{X}, t-l+1, t), \quad 1 \leq t \leq T. \tag{3.14}$$

Using those three quantities the most probable sequence of segments $\mathbf{S}^*$ of $\mathbf{X}$ can be determined in the backtracking step. $\hat{\alpha}_T(y)$ denotes the score of the most likely sequence of segments, where the last segment is labelled y. Therefore the last label y_n^* of $\mathbf{S}^*$ is given by the argument maximum of $\hat{\alpha}_T(y)$. Next, the length of the last segment l_n^* needs to be determined. That can be done by inserting the obtained value of y_n^* into the quantity $\hat{l}_T(y_n^*)$. Subsequently, the previous label y_{n-1}^* needs to be determined. y_{n-1}^* can be obtained from the quantity $\hat{y}_T(y_n^*, l_n^*)$ inserting the previously obtained most probable label and length of the last segment. The length of the second to last segment can then be determined by $\hat{l}_{T-l_n^*}(y_{n-1}^*)$. This procedure is repeated until the beginning of the observation sequence is reached.

Below in Algorithm 2 is a pseudo code of the backtracking procedure to obtain $\mathbf{Y}^*$ and $\mathbf{L}^*$. Recall that $\mathbf{L}^*$ can be converted into $\mathbf{A}^*$ and vice versa.

Algorithm 2 Backtracking procedure to determine the most probable labelling $\mathbf{Y}^*$ and the corresponding sequence of segment lengths $\mathbf{L}^*$ given in terms of the segment lengths.

```
Y* ← list()                                   ▷ this is an empty list
L* ← list()                                   ▷ this is an empty list
Y* ← add_front(Y*, arg max_y α̂_T(y))          ▷ add item to front of Y*
L* ← add_front(L*, l̂_T(y*_n))
l_rem ← T − l*_n                              ▷ remaining number of observations
t = T                                         ▷ current position in X
while l_rem > 0 do
    Y* ← add_front(Y*, ŷ_t(L*[0], Y*[0]))     ▷ L*[0] is the first (front) item of L*
    t ← t − L*[0]
    l ← l̂_t(Y*[0])
    l_rem ← l_rem − l
    L* ← add_front(L*, l)
end while
```

3.7 Parameter learning

Parameters are determined by maximizing the conditional log-likelihood over a given training set. [18] For the moment we assume that each training example consists of an observation sequence $\mathbf{X}$, the labels $\mathbf{Y}$, and the corresponding alignment sequence

A of the labels. Recall that the alignment sequence assigns each label to a subsequence of observations. In a section below we introduce an efficient model training method that does not need a given alignment sequence.

The log-likelihood of the model is given by

$$\log p(\mathbf{S}|\mathbf{X}) = \sum_{i=1}^{|\mathbf{S}|} \log \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i) - \log Z(\mathbf{X}). \tag{3.15}$$

We wish to maximize $\log p(\mathbf{S}|\mathbf{X})$ on a given training dataset. For that purpose we apply stochastic gradient ascent to adapt each parameter of the model. The partial derivative of $\log p(\mathbf{S}|\mathbf{X})$ with respect to the model parameter w_{yd} is:

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{yd}} = \tag{3.16}$$

$$\sum_{i=1}^{|\mathbf{S}|} \mathbf{f}_d(\mathbf{X}, t_i, u_i)\delta_{yy_i} - \frac{1}{Z(\mathbf{X})} \sum_{\mathbf{S}'} \sum_{i=1}^{|\mathbf{S}'|} \mathbf{f}_d(\mathbf{X}, t_i, u_i)\delta_{yy_i} \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j), \tag{3.17}$$

where w_{yd} is the d-th element of the $|\mathcal{Y}|$ dimensional parameter vector $\mathbf{w}_y$. A step by step derivation of this result is given in Appendix A.1. The left term of Equation (3.17) is easy to compute if the alignment of $\mathbf{Y}$ is part of the training data. However, the right term of the gradient is problematic because in this form it is required to explicitly enumerate all segment sequences comprising the T observations of the current training example.

First we analyse the term: The sum runs over all segment sequences and adds up the product of the d-th component of the input vector for each segment that is labelled y, multiplied with the score of the complete segment sequence.

To circumvent the need of explicitly enumerating all segment sequences we consider every possible segment labelled y. Recall that a segment is defined by it's position and label. Thus the segment can start at each position and have length $1 \leq l \leq L_{\max}$, such that it does not overrun the observation sequence $\mathbf{X}$. The d-th component of the input vector of this segment, has to be multiplied with the score of all segment sequences it is part of. In other words we need to consider all segment sequences that have a segment of length l, starting at position t, ending at position $t + l - 1$, and that has label y.

A sequence of segments $\mathbf{S}$, containing a specific segment $s' = (y, t', u')$, can be decomposed into a prefix and a suffix sequence: $\mathbf{S} = \mathbf{S}_{\text{pre}} \cup (s') \cup \mathbf{S}_{\text{suf}}$. The prefix sequence $\mathbf{S}_{\text{pre}}$ is a segmentation of the first $t - 1$ observations. Parameters other than that are arbitrary. That is, the number of segments, their positions, and the labels do not matter at all. The first segment of the suffix sequence $\mathbf{S}_{\text{suf}}$ starts at position u' and is a segmentation of the observations from u' (exclusive) to the end of the observation sequence $\mathbf{X}$. Parameters other than that are again arbitrary.

The scores of all possible prefix and the suffix segment sequences are given by the forward and backward recursions. Thus $\alpha_{t'-1}(y)$ is the score of all possible prefix segment sequences that have a last segment that is labelled y. Analogue, $\beta_{u'+1}(y)$ is the score of all possible suffix segment sequences that have a first segment that is labelled y.

Thus by making use of the forward and backward recursions we can compute

the gradient in an efficient way, *i.e.*

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{yd}} = \sum_{i=1}^{|\mathbf{S}|} \mathbf{f}_d(\mathbf{X}, t_i, u_i)\delta_{yy_i} \tag{3.18}$$

$$- \frac{1}{Z(\mathbf{X})} \sum_{t=1}^{T} \sum_{l=1}^{L} \mathbf{f}_d(\mathbf{X}, t, u)\beta_{t+l}(y) \sum_{y'} \alpha_{t-1}(y')\Phi(y, y', \mathbf{X}, t, u), \tag{3.19}$$

where $L = \min\{L_{\max}, T-t+1\}$ in order to avoid going beyond T in the observation sequence and $u = t + l - 1$ (*i.e.* the index of the last observation of the segment).

As already elaborated above, the score of the prefix sequence is captured by α_{t-1} and the score of the suffix sequence by β_{t+l}. However, the potential function of the current segment is not yet considered by the recursions. Therefore this value needs to be multiplied with the recursion quantities. Since the potential function depends on the label of the previous segment and we want the score of all segment sequences that have a segment of label y, we need to sum over all possibilities for the previous label. The previous label of the considered segment is the same as the label of the last segment in the prefix sequence $\mathbf{S}_{\text{pre}}$. The transition from the current segment at position t to the subsequent segment, which is the first in the suffix sequence $\mathbf{S}_{\text{suf}}$, does not have to be handled in a special way, because $\beta_{t+l}(y)$ was defined to already contain the transition factor from the segment labelled y to the next one.

Assuming that the α and β recursions are already fully computed, computing the partial derivative of the log-likelihood with respect to the w_{yd} parameter takes at maximum $\mathcal{O}(T \cdot L_{\max} \cdot |\mathcal{Y}|)$ time.

In a graphical model, summing out all variables but one, and dividing by the partition function yields the marginal probability of the remaining variable [3, page 396]. Hence, pulling the division by the partition function in Equation (3.19) into the double sum one obtains the **marginal** probability of a segment $s = (y, t, u)$:

$$p(s|\mathbf{X}) = p((y, t, u)|\mathbf{X}) = \frac{\beta_{t+l}(y) \sum_{y'} \alpha_{t-1}(y')\Phi(y, y', \mathbf{X}, t, u)}{Z(\mathbf{X})}. \tag{3.20}$$

Thus we can rewrite Equation (3.19) using the marginal probability to

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{yd}} = \sum_{i=1}^{|\mathbf{S}|} \mathbf{f}_d(\mathbf{X}, t_i, u_i)\delta_{yy_i} - \sum_{t=1}^{T} \sum_{l=1}^{L} \mathbf{f}_d(\mathbf{X}, t, u)p((y, t, u)|\mathbf{X}), \tag{3.21}$$

where again $u = t + l - 1$.

The formula for efficiently computing the partial derivative with respect to the bias parameter λ_y is the same, but without multiplying the output of the feature function. For details on the differences in the derivation process we refer to Appendix A.2.

For the two label dependent parameter $v_{yy'd}$ the same principle applies. However, there is one additional constraint: We need to consider all segment sequences that contain a segment of length l, starting at position t, labelled y, and the preceding segment is labelled y'. However, we do not make any further assumptions about the previous segment. This additional constraint simplifies Equation (3.19) in the sense that it eliminates the need to sum over all possible preceding segment labels y'. The details of the derivation are given in Appendix A.3. Here we present the

partial derivative that can be computed in an efficient way:

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial v_{yy'd}} = \sum_{i=1}^{|\mathbf{S}|} \mathbf{g}_d(\mathbf{X}, t_i,)\delta_{yy_i}\delta_{y'y_{i-1}} \tag{3.22}$$

$$- \sum_{t=2}^{T}\sum_{l=1}^{L} \mathbf{g}_d(\mathbf{X}, t)\frac{\alpha_{t-1}(y')\Phi(y, y', \mathbf{X}, t, t+l-1)\beta_{t+l}(y)}{Z(\mathbf{X})}. \tag{3.23}$$

Important to note is that the summation now starts at $t = 2$ because the segments starting at position 1 do not have a predecessor. This result can not be interpreted in terms of a marginal probability of a single segment any more. Moreover it is also not the marginal probability of two segments, because the position information of the previous segment is missing. Therefore the previous segment is not uniquely defined.

The partial derivative with respect to the two label dependent bias parameter $\mu_{yy'}$ is again the same as in Equation (3.23), but without multiplying the output value of the feature function.

3.8 Efficient SCRF

The potential function defined in Equation ((3.4)) is the exponential of the sum over two terms. We insert the definition of the potential function into the forward recursion of the SCRF:

$$\begin{aligned}\alpha_t(y) &= \sum_{l=1}^{L_{\max}}\sum_{y'} \alpha_{t-l}(y')\Phi(y, y', \mathbf{X}, t-l+1, t) \\ &= \sum_{l=1}^{L_{\max}}\sum_{y'} \alpha_{t-l}(y')\exp\left[\psi(y, y', \mathbf{X}, t-l+1) + \phi(y, \mathbf{X}, t-l+1, t)\right].\end{aligned}$$

The state factor ϕ depends on the current label, the starting position, and the length of the segment. The transition factor ψ requires the current and previous labels and the starting position of the current segment. It does not depend on the length of the current segment. If we could get rid of the dependency on the segment length l in the transition factor, the recursion could be split into two parts. The part summing over all possible segment lengths would then become independent of the part summing over all possible labels of the previous segment. In other words the transition potential becomes independent of the state potential, resulting in a more efficient way of handling the SCRF.

As it turns out this dependency can be dropped if the forward recursion is divided into two recursions as in [10]:

$$\alpha_t^{\text{state}}(y) = \sum_{l=1}^{L_{\max}} \alpha_{t-l}^{\text{bound}}(y)\exp\left[\phi(y, \mathbf{X}, t-l+1, t)\right], \qquad 1 \le t \le T, \tag{3.24}$$

$$\alpha_t^{\text{bound}}(y) = \sum_{y'} \alpha_t^{\text{state}}(y')\exp\left[\psi(y, y', \mathbf{X}, t)\right], \qquad 1 \le t < T, \tag{3.25}$$

where $\alpha_t^{\text{state}}(y)$ represents the same as $\alpha_t(y)$. $\alpha_t^{\text{bound}}(y)$ represents the accumulated potentials of the first t observations of $\mathbf{X}$ including the transition factor to the

subsequent segment where the subsequent segment will be labelled y. In order to compute the partition function using the modified forward recursion one obviously needs to take the $\alpha_T^{\text{state}}(y)$ quantity, *i.e.*

$$Z(\mathbf{X}) = \sum_y \alpha_T^{\text{state}}(y). \tag{3.26}$$

This redefinition changes the asymptotic time complexity of the recursion to $\mathcal{O}(T \cdot (L_{\max}|\mathcal{Y}| + |\mathcal{Y}|^2) \cdot M)$. If $L_{\max} \leq |\mathcal{Y}|$ then asymptotically the dependency on $L_{\max}$ can be neglected and the forward recursion has the same complexity as for ordinary, non-segmental CRFs. In practice however there is always overhead and the efficient SCRF will have a longer runtime than the ordinary CRF.

The same trick can also be applied to the backward recursion:

$$\beta_t^{\text{state}}(y) = \sum_{l=1}^{L_{\max}} \beta_{t+l}^{\text{bound}}(y) \exp\left[\phi(y, \mathbf{X}, t, t+l-1)\right], \qquad 1 \leq t \leq T, \tag{3.27}$$

$$\beta_t^{\text{bound}}(y) = \sum_{y'} \beta_t^{\text{state}}(y') \exp\left[\psi(y, y', \mathbf{X}, t)\right], \qquad 1 < t \leq T, \tag{3.28}$$

where $\beta_t^{\text{bound}}(y)$ corresponds to $\beta_t(y)$. $\beta_t^{\text{state}}(y)$ represents the sum over the scores of all segment sequences comprising the observations starting at t to the end of $\mathbf{X}$, and the first segment is labelled y.

The partition function using the new backward recursion can be computed in the following way:

$$Z(\mathbf{X}) = \sum_y \sum_{l=1}^{L_{\max}} \Phi(y, \mathbf{X}, 1, l) \beta_{l+1}^{\text{bound}}(y) = \sum_y \beta_1^{\text{state}}(y). \tag{3.29}$$

The redefined recursions simplify the computation of the marginal probability of a segment to:

$$p(s|\mathbf{X}) = p((y,t,u)|\mathbf{X}) = \frac{\alpha_{t-1}^{\text{bound}}(y) \exp\left[\phi(y, \mathbf{X}, t, u)\right] \beta_u^{\text{bound}}(y)}{Z(\mathbf{X})}, \tag{3.30}$$

where the summation over all possibilities for the previous segment label is already covered by $\alpha_{t-1}^{\text{bound}}$ and thus not necessary when computing the marginal.

Using the redefined recursions the partial derivative with respect to two label dependent parameter changes as follows:

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial v_{yy'd}} = \sum_{i=1}^{|\mathbf{S}|} \mathbf{g}_d(\mathbf{X}, t_i,) \delta_{yy_i} \delta_{y'y_{i-1}} \tag{3.31}$$

$$- \sum_{t=2}^{T} \sum_{l=1}^{L} \mathbf{g}_d(\mathbf{X}, t) \frac{\alpha_{t-1}^{\text{state}}(y') \Phi(y, y', \mathbf{X}, t, t+l-1) \beta_{t+l}^{\text{bound}}(y)}{Z(\mathbf{X})}. \tag{3.32}$$

3.9 L^2-SCRF

The efficient segmental conditional random field can be derived because of the fact that we have defined that the transition factor does not depend on the length of the current segment.

In this section we introduce the L^2-SCRF, which is a model in which the transition factor depends on the observations of two segments. That means that the starting and ending positions, as well as the labels of two adjacent segments are available to the transition factor. The introduction of this model can be motivated from two perspectives:

First, in [10] it was shown that efficient SCRFs with state features comprising the observations of the current segment and transition features utilizing a fixed window of context at the segment boundary, outperform the default SCRF utilizing the inputs of the current segment as transition features. It is unknown whether the context of two adjacent segments leads to a further improvement of phone recognition performance.

Second, in [17] conditional random fields were used for phone classification. In that work the first order transition factor depends on the data of the current and the previous phone. This model achieved a significantly higher phone classification accuracy then the model using only transition bias. Following this principle and adopting it to phone recognition we now introduce the L^2-SCRF.

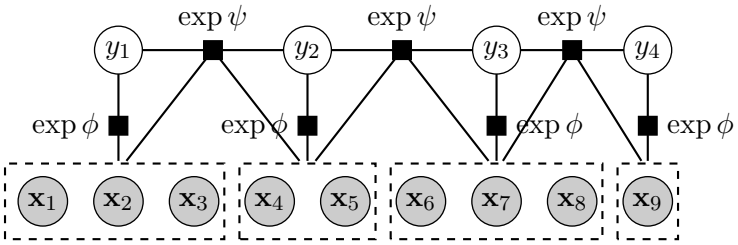


Figure 3.5: Factor graph representation of an example L^2-SCRF for $T = 9$ given observations partitioned into $n = 4$ segments. The transition factor depends on the observations of two segments.

This requires a redefinition of the logarithmic transition factor and the corresponding feature function. In the L^2-SCRF they both depend on the starting position of the previous segment, the starting and ending positions of the current segment:

$$\psi(y_i, y_{i-1}, \mathbf{X}, t_{i-1}, t_i, u_i) = \sum_{y,y'} \left(\mathbf{v}_{y'y}^T \mathbf{g}(\mathbf{X}, t_{i-1}, t_i, u_i) + \mu_{y'y}\right) \delta_{yy_i} \delta_{y'y_{i-1}}. \tag{3.33}$$

Thus also the potential function is extended by one additional parameter, *i.e.*

$$\Phi(y_i, y_{i-1}, \mathbf{X}, t_{i-1}, t_i, u_i) \equiv \exp\left[\psi(y_i, y_{i-1}, \mathbf{X}, t_{i-1}, t_i, u_i) + \phi(y_i, \mathbf{X}, t_i, u_i)\right].$$

The dependency on two segment lengths makes it necessary to redefine the recursions. In all of them the length of the previous segment needs to be explicit. Hence the forward recursion changes to

$$\alpha_t(y, l) = \sum_{l'=1}^{L_{\max}} \sum_{y'} \alpha_{t-l}(y', l') \Phi(y, y', \mathbf{X}, t - l - l' + 1, t - l + 1, t), \tag{3.34}$$

where t is the ending position of the current segment. Hence the segment starting position is given by $t - l + 1$. The starting position of the previous segment is then l' observations to the left. In order to assert that the recursion is also correct at

the beginning of the observation sequence: $\alpha_0(y, l) = 1$ and for $t < 0 : \alpha_t(y, l) = 0$. $\alpha_t(y, l)$ is the sum over all segment sequences comprising the first t observations, where the last segment is labelled y and has length l.

The asymptotic runtime of the forward recursion of this model is now quadratic in the maximum segment length: $\mathcal{O}(T \cdot L_{\max}^2 \cdot |\mathcal{Y}|^2 \cdot M)$, where M again denotes the number of feature function evaluations required to compute the potential function. The runtime of this model is excessive for larger values of T, $L_{\max}$ or $\mathcal{Y}$.

The backward recursion is defined similarly:

$$\beta_t(y, l) = \sum_{l'=1}^{L_{\max}} \sum_{y'} \beta_{t+l}(y', l') \Phi(y, y', \mathbf{X}, t, t + l, t + l + l' - 1). \tag{3.35}$$

In order to assert that the recursion is also correct at the ending of the observation sequence: $\beta_{T+1}(y, l) = 1$ and for $t > T + 1 : \beta_t(y, l) = 0$. $\beta_t(y, l)$ is the sum over all segment sequences comprising the observations t to the end, where the first segment is labelled y and has length l.

As for the SCRF, the partition function can be computed from the last value of the forward recursion or the first value of the backward recursion. However, here is the need to additionally sum out all possible lengths of the last or respectively first segment.

$$Z(\mathbf{X}) = \sum_{y} \sum_{l=1}^{L_{\max}} \alpha_T(y, l) = \sum_{y} \sum_{l=1}^{L_{\max}} \beta_1(y, l) \tag{3.36}$$

The most probable segment sequence can be determined efficiently by replacing summations by maximizations in Equation (3.34).

Finally we consider the marginal probabilities of this model. Those are required to compute the partial derivative of the log-likelihood function with respect to the model parameters. Recall that for the "default" SCRF model it was not possible to compute the marginal probability of two adjacent segments. This is due to the fact that the SCRF is agnostic about the length of the previous segment. The L^2-SCRF model however has the information required to uniquely define two adjacent segments. Thus it is possible to compute the marginal of them.

Here $s' = (y', t', u')$ denotes the preceding segment of $s = (y, t, u)$. Hence $t = u' + 1$. Let l' denote the length of s' and let l denote the length of s. They are given by: $l' = u' - t' + 1$ and $l = u - u'$.

The marginal probability of two segments is given by

$$p(s', s|\mathbf{X}) = \frac{\alpha_t(y', l') \exp\left[\psi(y_i, y_{i-1}, \mathbf{X}, t_{i-1}, t_i, u_i)\right] \beta_t(y, l)}{Z(\mathbf{X})}. \tag{3.37}$$

As depicted in the Equation 3.37 above, the marginal of two adjacent segments can be computed from the quantities of the forward and backward recursions. However, there is one speciality here: $\alpha_t(y', l')$ covers the sum over all segment sequences of the first t observations and $\beta_t(y, l)$ covers the segment sequences of the last $T - t$ observations, starting at t. What is not yet considered is the transition factor from the segment s' to the segment s. Thus the value of transition factor is multiplied to the values of alpha and beta. Again the division by the partition function ensures that one obtains a probability [3, page 396].

The marginal of a single segment is given in terms of the marginal of two adjacent segments and marginalizing out all possible preceding segments. Again let l denote

the length of s, then the marginal of a segment is given by

$$p(s|\mathbf{X}) = p((y,t,u)|\mathbf{X}) = \sum_{l'} \sum_{y'} p((y', t-l', t-1), s). \tag{3.38}$$

Furthermore the L^2-SCRF model is the foundation if one wants to introduce a SCRF with a second order transition factor. A second order transition factor captures the labelling of three adjacent segments. Therefore a second order transition factor would not be present for the first two segments in any segment sequence $\mathbf{S}$. Hence, the recursions (forward, backward, Viterbi) require the information weather the sum or maximum over all segment sequences constructed from t observations consist of more than two segments. This information is given by keeping the length of the previous segment explicit. In other words, if a segment ends at t and the previous two segment lengths added up are smaller than t, then there are at least three segments involved and a second order transition factor is in place.

3.10 Hidden layer SCRF

The potential functions of the SCRF are computed as dot products between the inputs and the learned weights. Thus the SCRF is a linear model. We extend the model by introducing a hidden layer of non-linear activations. This hidden layer corresponds to the hidden layer used in feed forward neural networks.

We refer to a SCRF that has been equipped with a hidden layer as *hidden layer segmental conditional random field*. In this model, the observations that are transformed into a fixed size input vector by the feature function serve as inputs to the hidden layer. The output of the hidden layer is then used as input to compute the logarithm of the SCRF factor. Both the state and the transition factors can be extended by a hidden layer.

The logarithm of the state factor in a SCRF with a hidden layer inserted underneath the state factor is given by:

$$\phi(y_i, \mathbf{X}, t_i, u_i) = \sum_{y} \left(\mathbf{w}_y^T \mathbf{z}(\mathbf{X}, t_i, u_i) + \lambda_y\right) \delta_{y y_i}, \tag{3.39}$$

where the components of the n dimensional vector $\mathbf{z}(\mathbf{X}, t_i, u_i)$ are given by:

$$z_j(\mathbf{X}, t_i, l_i) = h(\mathbf{w}_j^T \mathbf{f}(\mathbf{X}, t_i, u_i) + b_j). \tag{3.40}$$

By n we denote the number of neurons or units in the hidden layer. z_j is the output value of the j-th neuron. Just like in a feed forward neural network the output of each neuron is given by the non-linear transformation of it's activation. $h(x)$ is some differentiable non-linear activation function. The activation is computed from the dot product between the output of the feature function $\mathbf{f}(\mathbf{X}, t_i, u_i)$ and the weights of the neuron [3, pages 227ff].

Adding a hidden layer underneath the transition factor can be done in the same way.

Apart from using the output of the hidden layer, instead of the output of the feature function directly, all the recursions and algorithms of the SCRF remain the same.

3.11 Backpropagation

In order to learn the parameters of the hidden layer we apply the backpropagation algorithm [3, pages 241ff].

The partial derivatives of the potential function parameters remain in principle the same. An obvious but crucial difference is that instead of multiplying the output of the feature function $\mathbf{f}$ the output of the hidden layer is used. For illustration we provide the partial derivative of the log-likelihood with respect to w_{yd} for a SCRF that has a hidden layer underneath the state factor. The formula is closely related to Equation 3.21, *i.e.*

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{yj}} = \sum_{i=1}^{|\mathbf{S}|} z_j(\mathbf{X}, t_i, u_i)\delta_{yy_i} - \sum_{t=1}^{T}\sum_{l=1}^{L} z_j(\mathbf{X}, t, t+l-1)p((y, t, t+l-1)|\mathbf{X}).$$

Now we consider the derivative of the log-likelihood with respect to the hidden layer parameter w_{jd} of neuron j. A detailed derivation of the partial derivative is given in Appendix A.5. The result of the derivation is:

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{jd}} = \sum_{i=1}^{|\mathbf{S}|} w_{y_i j} h'(a_j)\mathbf{f}_d(\mathbf{X}, t_i, u_i) \tag{3.41}$$

$$- \frac{1}{Z(\mathbf{X})}\sum_{\mathbf{S}'}\sum_{i=1}^{|\mathbf{S}'|} w_{y_i j} h'(a_j)\mathbf{f}_d(\mathbf{X}, t_i, u_i)\prod_{j=1}^{|\mathbf{S}'|}\Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j). \tag{3.42}$$

The right side of derivative contains a sum over all segment sequences of T observations. As already elaborated in the Sections 3.4 and 3.7 this is practically intractable as there are exponentially many segmentations. By using the same argumentation as in Section 3.7 this explicit summation can be avoided and the partial derivative computed efficiently using the quantities of the forward-backward algorithm. Let $u = t + l - 1$

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{jd}} = \sum_{i=1}^{|\mathbf{S}|} w_{y_i j}\mathbf{f}_d(\mathbf{X}, t_i, u_i)h'(a_j) \tag{3.43}$$

$$- \frac{1}{Z(\mathbf{X})}\sum_{t=1}^{T}\sum_{l=1}^{L}\mathbf{f}_d(\mathbf{X}, t, u)h'(a_j)\sum_{y} w_{yj}p((y, t, u)|\mathbf{X}). \tag{3.44}$$

There is however an important difference, *i.e.* the output of the hidden layer is independent of the current segment label. Therefore we need to sum over all possible labels for the current segment.

3.12 Marginalization of alignments

The partial derivatives of the SCRF parameters derived in Section 3.7 require a given alignment $\mathbf{A}$ of the labelling sequence $\mathbf{Y}$. There are multiple ways of obtaining such an alignment. One possibility is to manually align each labelling sequence in the training set to the corresponding observation sequence. Another approach is to rely on a first pass system that creates an alignment of $\mathbf{Y}$

In this section we choose a different path. We develop and present an efficient method for marginalizing out the alignment $\mathbf{A}$ from the conditional probability of

the model. In the next section we present the gradients required to train the model without a given alignment sequence.

Recall the definition of the conditional probability of the SCRF in terms of the labelling and the alignment sequences:

$$p(\mathbf{S}|\mathbf{X}) = p(\mathbf{Y}, \mathbf{A}|\mathbf{X}) = \frac{1}{Z(\mathbf{X})} \prod_{i=1}^{|\mathbf{S}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i).$$

If the alignment $\mathbf{A}$ is not available (in the training data) it can be marginalized out:

$$\begin{aligned} p(\mathbf{Y}|\mathbf{X}) &= \sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} p(\mathbf{Y}, \mathbf{A}|\mathbf{X}) \\ &= \sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} \frac{1}{Z(\mathbf{X})} \prod_{i=1}^{|\mathbf{Y}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i) \\ &= \frac{\sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} \prod_{i=1}^{|\mathbf{Y}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{Z(\mathbf{X})}. \end{aligned}$$

It is important than when marginalizing out the alignments one must only consider those that are consistent with the given labelling $\mathbf{Y}$ and the observation sequence $\mathbf{X}$. In Section 3.1 we defined that a valid alignment $\mathbf{A}$ of $\mathbf{Y}$ and $\mathbf{X}$ has as many elements as there are labels, and the ending position of the last alignment item $a_n = (t_n, u_n)$ is equal to the number of observations in $\mathbf{X}$. Additionally the length of each segment is constrained to comprise at maximum $L_{\max}$ observations.

At this point we encounter the same problem as above when an efficient formula for computing the partition function was developed. There are exponentially many valid alignments. The last segment in all valid alignments ends at T and can have the length 1 up to $L_{\max}$. Thus for the last segment there are up to $L_{\max}$ possible starting positions. The starting position of a segment is given in terms of the end position of the preceding segment plus one. So for each of the up to $L_{\max}$ starting positions of the last segment there are as many possible starting positions of the penultimate segment. For each of them there are again up to $L_{\max}$ possible predecessors and so forth. Therefore an upper bound for the number of alignments is given by $(L_{\max})^n$. It is intractable to explicitly enumerate them and perform summation over their potential scores. Once again we have to make use of dynamic programming in order to compute the required sum in an efficient way.

We split the problem of finding an efficient method for computing this sum into sub problems. First we consider the task of deciding whether the i-th segment $a_i = (t_i, u_i)$, starting at t_i and ending at u_i in the observation sequence, is part of at least one valid sequence $\mathbf{A}$.

In order to obtain an efficient algorithm, we need to find a method of locally determining whether the constrains are met. That is, using only the start and end positions of a segment we have to decide whether the complete alignments it is part of will comprise exactly T observations and n segments.

The maximum length constraint is very simple to verify. From the known starting and ending positions (t_i, u_i) the length of the segment can be computed and hence this constraint can be checked, *i.e.*

$$u_i - t_i + 1 \leq L_{\max}.$$

Next we develop the criteria to check whether the starting position of the segment is valid. That means that t_i needs to be far enough away from the beginning of the observation sequence as to leave enough space for $i-1$ preceding segments of minimum length. However, it also needs to be asserted that the starting position t_i is close enough to the beginning of the observation sequence. It has to be possible that t_i can be reached with $i-1$ segments. Since each segment has a maximum length, the right most possible position of t_i is given by $(i-1)L_{\max}+1$. The first segment of all alignments always starts at $t_1 = 1$. We can formulate those properties as:

$$i \leq t_i \leq (i-1)L_{\max} + 1. \tag{3.45}$$

Next, we examine the ending position u_i with respect to the number of given segments and observations. The ending position, similar to the starting position, needs to be close enough to the end of the observation sequence such that the last observation can be reached with $(n-i)$ remaining segment of maximum length. Moreover it needs to leave enough space to insert $(n-i)$ remaining segments. The last segment of all alignments always ends at T, *i.e.*

$$T - (n-i)L_{\max} \leq u_i \leq T - (n-i). \tag{3.46}$$

Note that the possible starting positions do not yet take the number of observations into account. However, the constraint that the alignment will comprise exactly T observations is met by the possible ending positions. Since $t_i \leq u_i$ always holds for any segment, this constraint is also met.

Algorithm 3 gives a pseudo code for checking whether a given pair of start and ending positions define the position of the i-th segment that is part of any valid alignment sequence with respect to given observation sequence $\mathbf{X}$ and labelling sequence $\mathbf{Y}$.

Algorithm 3 Pseudo code for checking whether a given pair of start and ending positions define the position of the i-th segment that is part of at least one valid alignment sequence. The code expects that the presented segment is valid in the sense that $t \leq u$ and the maximum length constraint is met.

```
function isValidSegment(i, t, u, |X|, |Y|)
    if u < 1 ∨ u > T then
        return False
    end if
    if t < i ∨ t > (i − 1)L_max + 1 then
        return False
    end if
    if T − (n − i)L_max < u ∨ u > T − (n − i) then
        return False
    end if
    return TRUE
end function
```

Now that we have a simple method of verifying whether a segment is part of any valid alignment sequence, we turn into the problem of computing the sum of all alignment scores. The score of a single alignment is given by the product of it's potentials. A recursive approach for computing the sum over all alignments is motivated by the observation that alignments are consisting of touching segments.

The starting position of segment i is given by the ending position of segment $i-1$ plus one. We introduce $\gamma_i(u;\mathbf{X},\mathbf{Y})$ as the sum over the scores of all alignments consisting of exactly i segments, where the last segment ends at positions u. For the last segment i ending at u there are up to $L_{\max}$ possible starting positions. For each of those possibilities it has to be checked that they are valid. This can be done with Algorithm 3. The potential of each valid segment is then multiplied by the scores of all alignments consisting of $i-1$ segments ending one position ahead of the starting position of that segment. Hence, $\gamma_i(u;\mathbf{X},\mathbf{Y})$ can be determined recursively:

$$\gamma_i(u;\mathbf{X},\mathbf{Y}) = \sum_{l=1}^{L\max} \Phi(y_i, y_{i-1}, \mathbf{X}, u-l+1, u)\gamma_{i-1}(u-l;\mathbf{X},\mathbf{Y})\mathbf{1}_{\{\text{COND}\}}, \tag{3.47}$$

for $1 \leq i \leq n$ and where COND stands for the condition that $a_i = (u-l+1, u)$ defines a valid segment (*i.e.* isValidSegment$(i, t-l+1, t)$). For convenience $\gamma_0(0;\mathbf{X},\mathbf{Y}) = 1$ and $\forall u \in \mathbb{Z}, u \neq 0\ \gamma_0(u;\mathbf{X},\mathbf{Y}) = 0$.

Following the definition of γ the probability $p(\mathbf{Y}|\mathbf{X})$ is given by:

$$p(\mathbf{Y}|\mathbf{X}) = \sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} p(\mathbf{Y},\mathbf{A}|\mathbf{X}) = \frac{\gamma_n(T;\mathbf{X},\mathbf{Y})}{Z(\mathbf{X})}. \tag{3.48}$$

We now consider the asymptotic runtime requirement of computing $p(\mathbf{Y}|\mathbf{X})$. The recursion runs over each of the n segments. For each segment there are up to T possible ending positions. For each ending position up to $L_{\max}$ segments can be part of the alignment sequence. Again let M be the number of feature function evaluations required to compute the potential function. Hence the asymptotic upper bound for computing $\gamma_n(T;\mathbf{X},\mathbf{Y})$ is $\mathcal{O}(n \cdot T \cdot L_{\max} \cdot M)$.

3.13 Training without alignments

When marginalizing out all alignments the numerator of the likelihood function and therefore also the derivative with respect to the model parameters is changing. The derivative of the partition function remains the same. The gradient of the part that changes, namely the numerator, is derived here with respect to the input dependent state factor parameter w_{yd}:

$$\begin{aligned}
&\frac{\partial \log \sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} \prod_{i=1}^{|\mathbf{Y}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} \\
&= \frac{\sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} \sum_{i=1}^{|\mathbf{Y}|} \mathbf{f}_d(\mathbf{X}, t_i, u_i)\delta_{yy_i} \prod_{j=1}^{|\mathbf{Y}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} \prod_{i=1}^{|\mathbf{Y}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)} \\
&= \frac{\sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} \sum_{i=1}^{|\mathbf{Y}|} \mathbf{f}_d(\mathbf{X}, t_i, u_i)\delta_{yy_i} \prod_{j=1}^{|\mathbf{Y}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\gamma_{|\mathbf{Y}|}(|\mathbf{X}|;\mathbf{X},\mathbf{Y})}.
\end{aligned}$$

In this derivation the chain rule was applied because of the logarithm. Then the general product rule were applied to yield the above state result. This result contains some terms that are intractable to compute. The denominator of the derivative is the sum over all alignment scores. This sum can be computed efficiently, as derived in the previous section.

In order to compute the numerator efficiently the same argumentation as for the partial derivative of the partition function applies: The sum runs over all alignments and for each segment that is labelled y adds up the product of the d-th component of the input vector ($\mathbf{f}_d(\mathbf{X}, t_i, u_i)$) multiplied with the score of the alignment sequence (*i.e.* the product of the potentials: $\prod_{j=1}^{|\mathbf{S}|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j)$).

Instead of explicitly enumerating all alignments we consider for every label y_i in $\mathbf{Y}$ all possible segments. This can be done by considering all pairs of starting and ending positions. For each hypothetical segment it has to be checked that it is part of at least one valid alignment sequence. The method for doing this was outlined in the previous section. Additionally we only consider those segments that are labelled y. This dependency on y stems from the fact that we are considering the partial derivative with respect to w_{yd}. Each segment that satisfies those properties is going to be multiplied by the score of all alignments it is part of. Thus the alignments the segment is part of can be split into the prefix alignments and suffix alignments. The score of the prefix alignments consisting of the first $i-1$ labels is given by the γ recursion. In order to obtain the scores of the suffix alignments, which are consisting of the $n-i$ last labels, we need to introduce a recursion that runs backwards.
Hence, let $\zeta_i(t; \mathbf{X}, \mathbf{Y})$ be the over the scores of all alignments consisting of exactly $n-i$ segments, where the first segment starts at position t. The recursion is given by

$$\zeta_i(t; \mathbf{X}, \mathbf{Y}) = \sum_{l=1}^{L_{\max}} \Phi(y_i, y_{i-1}, \mathbf{X}, t, t+l-1)\zeta_{i+1}(t+l)\mathbf{1}_{\text{COND}}, \tag{3.49}$$

for $1 \leq i \leq |\mathbf{Y}|$ and where COND stands for the condition that $a_i = (t, t+l-1)$ defines a valid segment (*i.e.* isValidSegment$(i, t, t+l-1)$). For convenience we define $\forall t : \zeta_{|\mathbf{Y}|+1}(t; \mathbf{X}, \mathbf{Y}) = 1$. The same time complexity as for the γ recursion applies.

Using those two recursions the score of all alignments with a specific segment $s_i = (y_i, t_i, u_i)$ can be computed. Hence the efficient way of computing the partial derivative with respect to w_{yd} is

$$\frac{\partial \log \sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} \prod_{i=1}^{|\mathbf{Y}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} \tag{3.50}$$

$$= \sum_{i=1}^{|\mathbf{Y}|} \sum_{t=1}^{T} \sum_{u=t}^{t+L-1} \mathbf{f}_d(\mathbf{X}, t, u)\delta_{yy_i} \cdot \tag{3.51}$$

$$\frac{\gamma_{i-1}(t-1; \mathbf{X}, \mathbf{Y})\Phi(y_i, y_{i-1}, \mathbf{X}, t, u)\zeta_{i+1}(u+1; \mathbf{X}, \mathbf{Y})}{\gamma_{|\mathbf{Y}|}(|\mathbf{X}|)}, \tag{3.52}$$

where $L = \min\{L_{\max}, T-t+1\}$ to avoid going beyond T in the observation sequence $\mathbf{X}$. This result looks very similar to the gradient of the logarithm of the partition function. However, a key difference here is that the labelling is given.

The partial derivative with respect to the bias parameter depending on one label λ_y is similar except that the dependency on the d-th component of the feature function is missing.

Finally we conclude this section by examining the derivative of the numerator of the marginalized log likelihood function with respect to the two label dependent parameter $v_{yy'd}$. Since the observations and the labels are given there is no need to sum over hypothetical label sequences. That results in the fact that the partial

derivative with respect to the two label dependent factor parameters is very similar to Equation 3.52, apart from the dependency on two labels.

$$\frac{\partial \log \sum_{\mathbf{A} \text{ s.t. } |\mathbf{A}|=|\mathbf{Y}|} \prod_{i=1}^{|\mathbf{Y}|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial v_{yy'd}} \tag{3.53}$$

$$= \sum_{i=1}^{|\mathbf{Y}|} \sum_{t=1}^{T} \sum_{l=1}^{L} \mathbf{g}_d(\mathbf{X}, t) \delta_{yy_i} \delta_{y'y_{i-1}} \cdot \tag{3.54}$$

$$\frac{\gamma_{i-1}(t-1; \mathbf{X}, \mathbf{Y}) \Phi(y_i, y_{i-1}, \mathbf{X}, t, t+l-1) \zeta_{i+1}(t+l; \mathbf{X}, \mathbf{Y})}{\gamma_{|\mathbf{Y}|}(|\mathbf{X}|)}, \tag{3.55}$$

where $L = \min\{L_{\max}, T - t + 1\}$ to respect the length of the observations sequence $\mathbf{X}$.

Thus the asymptotic computational complexity of computing the partial derivatives with respect to the numerator of $p(\mathbf{Y}|\mathbf{X})$ is $\mathcal{O}(n \cdot T \cdot L_{\max} \cdot M)$. In contrast if $\mathbf{A}$ is given the complexity of the partial derivative of the numerator of $p(\mathbf{Y}, \mathbf{A}|\mathbf{X})$ is only $\mathcal{O}(n \cdot M)$.

3.14 Relation to CTC

Recurrent neural networks (RNNs) are powerful models for sequence classification. However, they make frame level predictions. That means a RNN produces one output for each observation. So the output sequence is of the same length as the observation sequence. Therefore RNNs require that the training data is segmented and the outputs must be post-processed. A hybrid approach to solve those problems involves HMMs. However this approach does not exploit the full potential of RNNs [6]. To overcome the limitations Graves et al introduced connectionist temporal classification (CTC). In this section we briefly introduce CTC and show the differences to segmental conditional random fields.

RNNs equipped with CTC have a softmax output layer and a blank symbol is added as an additional label. Hence the outputs of the network can be interpreted as probabilities. The probabilities of the network outputs are conditionally independent given the internal state of the network. The labelling, not to be confused with the output sequence, of the network is computed by a mapping of the output sequence. This mapping is defined by merging all adjacent equal output classes in the output sequence and then removing all blank symbols. The objective of a CTC network is the conditional probability of the labelling sequence given the observation sequence. This conditional probability is defined in terms of the sum over all output sequences that can be converted into the given labelling. This sum is computed in an efficient way with a recursion of a dynamic programming approach.

When making predictions a CTC network computes for each observation frame the output probabilities of each label. The predicted labelling is then given by either best path or prefix search decoding. The first method simply takes the most probable output sequence and maps it to the labelling sequence. The latter computes the most probable labelling by expanding the most probable output sequence prefixes to obtain the most probable labelling.

In contrast, the SCRF models the distribution of segment sequences for given observation sequences. Hence it produces segmental predictions in one step without the need of an intermediate step. For that purpose the model groups the observations

into segments and computes for each hypothetical segment a potential. Because of that, in a SCRF one always needs to address the problem of transforming a variable number of observations into a fixed sized feature vector in order to compute that potential. This problem is not present in CTC networks, because RNNs make one prediction per observation frame. This output of the RNN is then made "segmental".

Furthermore the SCRF models a dependency between two neighbour labels. This dependency on the outputs is not captured by CTC networks. However it is in the nature of RNNs to be able to model long term input context. Efficient SCRFs do have the context of the current hypothetical segment and the corresponding boundary.

The training of a CTC network is in principle very similar to the training of a segmental conditional random field when the alignment is not given. CTC distributes the n given labels onto the T observations. In contrast, the SCRF splits the T observations into n segments. The dynamic programming recursion of CTC has an upper bound of $\mathcal{O}(n \cdot T)$. Recall that the SCRF algorithm for summing over all alignments has an upper bound of $\mathcal{O}(n \cdot T \cdot L_{\max})$. The dependency on the maximum length parameter comes from the fact that the SCRF requires the segment information explicitly. A segment is defined by it's starting and ending position. A recurrent neural network does only need to have a label at each output t. It is agnostic about the actual starting and ending position of the current label. The fact that is does not require this information results in a less complex recursion.

Chapter 4

Experiments: Phone Recognition

In this chapter, we present the results of phone recognition experiments with the introduced segmental conditional random field model. We trained and evaluated the model on the TIMIT corpus [5], which contains 6300 sentences of American English spoken by 630 speakers from different dialect regions within the U.S.

This chapter is structured as follows: First we introduce the extraction of the features and the experimental setup. Then we deal with the problem of finding a feature function that produces a fixed sized input vector from segments of variable length. In those experiments the transition factor did not depend on any inputs and therefore only acted as a bias. Next, we explore the use of input dependent transition factors. Subsequently, we present recognition results of the L^2-SCRF which models a dependency on the current and the previous segment. Then experimental results of the SCRF extended by a hidden layer trained with backprop are shown. All those experiments were performed using the alignment information from the TIMIT corpus for training. Finally, we show experimentally that the algorithm for marginalizing out the alignment sequences during training also works in practice. In those experiments we did not use the alignments. We directly compare the two training methods and the achieved results. The chapter is concluded with a summary of the obtained experimental results and a comparison with the results of related work.

4.1 Implementation

All introduced models were implemented in python 2.7[1] using the packages `numpy` and `scipy`. When implementing the recursions presented in the previous chapter, one certainly faces the problem of numerical instabilities. In `numpy` the exponential overflows for arguments of ≈ 700 and greater. So the arguments of the exponential need not exceed this limit. A common trick of avoiding those overflows when

[1] https://www.python.org/download/releases/2.7/

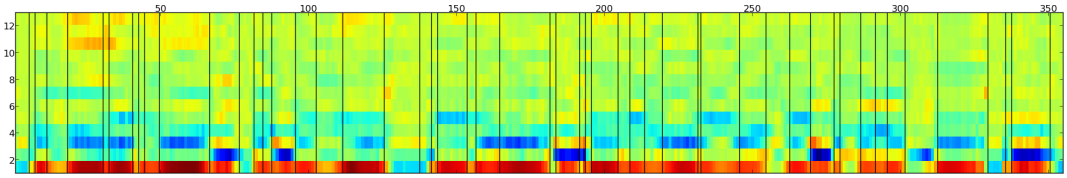
computing the logarithm of a sum over exponential terms is the following:

$$\begin{aligned} x &= c + \log \sum_i \exp v_i - c \\ &= c + \log \sum_i \frac{\exp v_i}{\exp c} = c + \log \frac{\sum_i \exp v_i}{\exp c} \\ &= c + \log \sum_i \exp v_i - \log \exp c \\ &= \log \sum_i \exp v_i, \end{aligned}$$

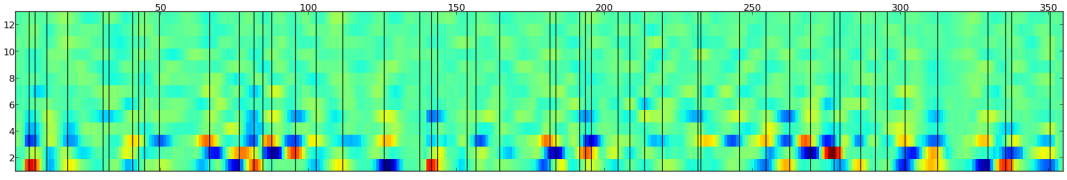
where $\mathbf{v}$ denotes an arbitrary vector of real numbers and c is an arbitrary real valued scalar. Hence, in the implementation the logarithmic quantities of the recursion are computed. Before summing over all terms the correction constant c is determined. This avoids the overflow of the exponential.

All implemented models were tested using a comprehensive set of test cases. The value of the partition function computed from the forward and backward recursions is checked by setting up a problem in which the explicit summation over all segmentations and segment sequences was still tractable. The result of the explicit summation is compared with the result from the forward and backward recursions. The implementation of the most probable segment sequence decoding is checked in the same way, *i.e.* the prediction is compared to the most probable sequence determined from enumerating all possible sequences. The correctness of the partial derivatives are checked by approximating them using finite differences.

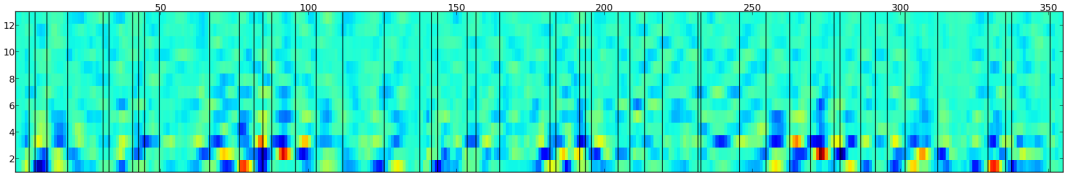
4.2 Feature extraction



(a) 12-th order MFCCs plus energy



(b) 12-th order MFCC deltas



(c) 12-th order MFCC double deltas

Figure 4.1: Mel frequency cepstral coefficient representation of a random utterance from TIMIT training set. The vertical black lines indicate the phone boundaries.

The waveform audio data in the TIMIT dataset was transformed into 12-th order Mel Frequency Cepstral Coefficients (MFCCs) and the energy-coefficient using a 25 ms window. Additionally we included deltas and double-deltas. Figure 4.1 depicts a plot of the MFCC representation of a random utterance from the TIMIT training dataset. The frame rate is 100 frames/s, therefore there was one 39 dimensional input vector per 10 ms of speech. The silent parts in the beginning and in the end of each utterance (h#) were cut to a maximum length of 5 frames. Those sections are absolutely silent and do not contain any information. Moreover we ignored glottal stops. This is common practice when using TIMIT [9].

4.3 Experimental setup

All models were trained on the 462 speaker training set with the SA utterances removed. So there were 3696 training sequences. To counteract over-fitting all experiments were conducted with early stopping. For this a 50 speaker development set consisting of 400 utterances following [8] was used. Final comparison of results is performed on the 24 speaker core-test set consisting of 192 utterances. The phone recognition performance is measured by of the phone accuracy. The phone accuracy of a training example is given by the Levenshtein distance between the predicted and the target phone sequence [4]. The Levenshtein distance is the number of insertion, deletion, and modification operations required to transform one sequence into another. The accuracy numbers are given as percentages, *i.e.* between 0 and 100.

According to [9] we collapsed the 61 phone classes into 39. Details are presented in Table 4.1. Hence the size of the label alphabet $\mathcal{Y}$ of the SCRF is $|\mathcal{Y}| = 39$.

0	iy	V	1	ih,ix	VS	2	eh	V
3	ae	V	4	ah, ax-h, ax	VS	5	uw, ux	V
6	uh	V	7	aa, ao	VS	8	ey	V
9	ay	V	10	oy	VS	11	aw	V
12	ow	V	13	er, axr	V	14	l, el	SVG
15	r	SVG	16	w	SVG	17	y	SVG
18	m, em	NF	19	n, en, nx	NF	20	ng, eng	NF
21	dx	S	22	jh	AF	23	ch	AF
24	z	F	25	s	F	26	sh, zh	F
27	hh, hv	SVG	28	v	F	29	f	F
30	dh	F	31	th	F	32	b	S
33	p	S	34	d	S	35	t	S
36	g	S	37	k	S			

Table 4.1: The 61 phones of TIMIT are distributed to 39 distinct classes as in [9]. The last class 38 contains the closure phones of the stops and the pauses: *bcl, pcl, dcl, tcl, gcl, kcl, epi, pau, h#*. Abbreviations of the phone types: V vowel, SVG semivowels and glides, NF nasals, AF affricate, F fricative, S stop. For details on the phones and types we refer to [5].

Training was performed by optimizing the conditional log likelihood of the model using stochastic gradient ascent. The model parameters were updated using the gradient computed from 4 utterances (mini-batch gradient ascent). The parameter

gradients of each mini-batch were computed in parallel on separate CPUs.

Instead of using the same learning rate for each epoch an empirically determined optimized learning rate schedule was used. Initially the model is untrained, so the first learning rates were selected high. Then the learning rate was gradually decreased over the epochs. After 16 iterations the minimum learning rate of $\eta = 0.0001$ was applied for the remaining epochs. In any tested case further lowering the learning rate did not result in a significant increase of the conditional log likelihood. Figure 4.2 depicts the learning rate schedule used in all experiments in this work.

Experiments were stopped if either the conditional log likelihood did not increase by more than $\epsilon = 0.02$ or if the accuracy on the development set had decreased for 15 iterations.

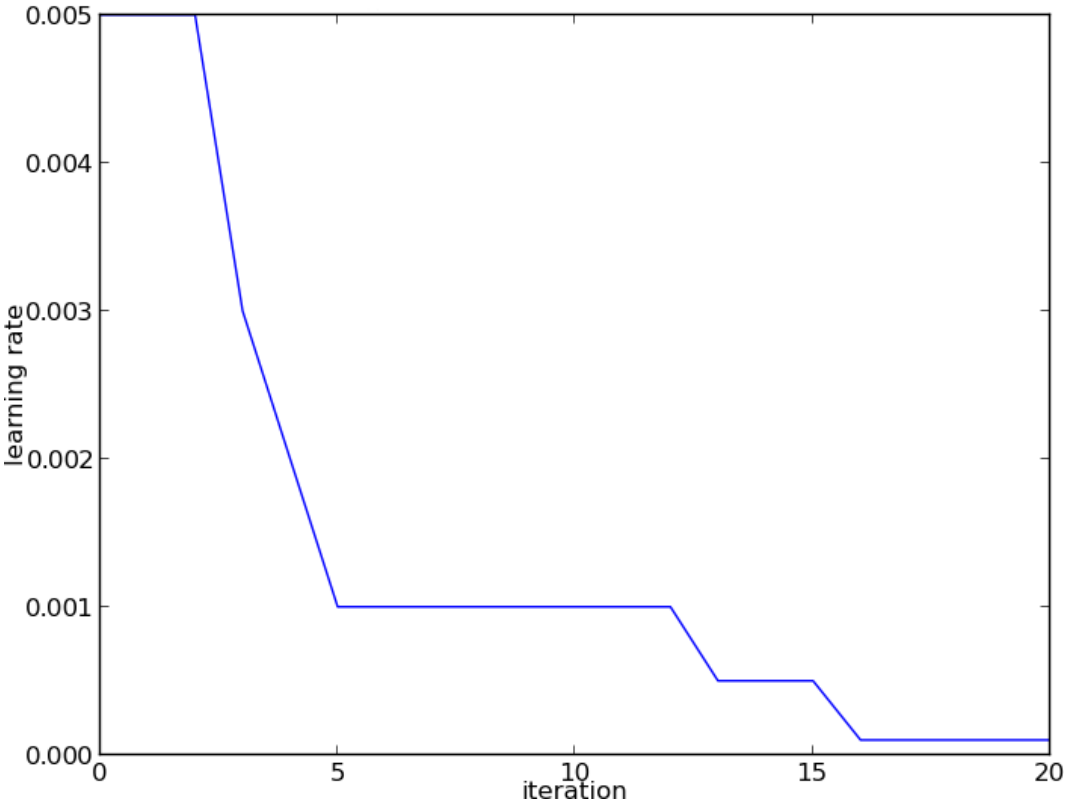


Figure 4.2: Learning rate schedule for model training. After 16 iterations of training the learning rate was not further decreased.

Training the model with given phone boundaries, the objective function of the SCRF model is convex [10]. Thus all experiments in this configuration were performed once and all parameters were initially set to zero. Those experiments converged within 15 to 35 epochs of training.

Training the SCRF without given alignment or adding a hidden layer results in a non-convex objective function. Those experiments were performed with initial weights sampled from a uniform distribution. For the experiments performed multiple times only the results from the best run with respect to the development set are reported. Training the SCRF without phone boundaries took significantly longer. The details are discussed in the corresponding section below.

For the experiments carried out with given alignment information, the inputs

were normalized to have zero mean and unit variance as they were presented to the model. That means that the outputs of the feature functions were normalized. This is a contrast to normalizing the extracted features directly. For the latter method of input normalization the resulting phone recognition accuracies were significantly worse compared to using the feature level normalization. When training the model without given alignments this form of normalization is not possible as it requires the alignments to be present. For those experiments the best results were achieved when the inputs were not normalized at all.

Another important parameter of the introduced segmental conditional random field is the maximum segment length $L_{\max}$. Figure 4.3 is a histogram plot of the distribution of segment lengths (*i.e.* the duration of phones) in the training dataset. In order to make predictions of long segments possible the parameter shall be set to the highest occurring value. On the other hand the runtime of the training and decoding algorithms depend linearly on the maximum length of a segment. Therefore, for faster runtime one wants to make $L_{\max}$ as small as possible. For all performed experiments the maximum length parameter was chosen to be $L_{\max} = 31$. With this parameter 99.95% of all segments in the training data are captured. A detailed discussion on the influence of the parameter on the recognition result is given in Section 4.7.

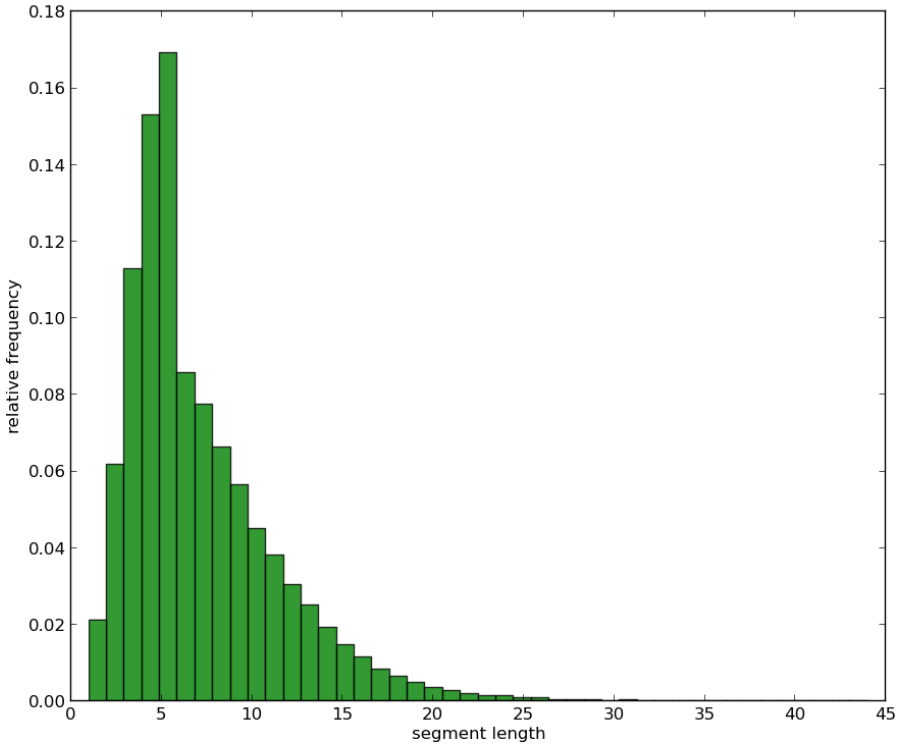


Figure 4.3: Distribution of segment lengths in the TIMIT training dataset.

4.4 State features

The SCRF model requires a fixed sized input, independent of the current segment length. This task is accomplished by the state feature function $\mathbf{f}(\mathbf{X}, t_i, u_i)$. This section documents the experiments carried out to empirically determine feature function

suited for the task of phone recognition. We evaluated various ways of processing the input vectors of a segment to form the input values of the model. Table 4.2 gives an overview of all tested state feature functions $\mathbf{f}(\mathbf{X}, t_i, u_i)$ and their phone recognition accuracies on the development set. In all experiments presented in this section the transition factor did not depend on any input and thus only acted as a bias!

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	#dimensions	#parameters	phone accuracy
$\frac{1}{l_i}\sum_{i=0}^{l_i-1}\mathbf{X}_{t+i}$	39	3081	30.00
$\mathbf{X}_{t_i}\|\|\mathbf{X}_{t_i+l_i-1}$	78	4602	54.54
$\mathbf{f}_1(\mathbf{X}, t_i, u_i)$	91	5109	57.24
$\mathbf{f}_2(\mathbf{X}, t_i, u_i)$	104	5616	57.09
$\mathbf{f}_3(\mathbf{X}, t_i, u_i)$	117	6123	58.17
$\mathbf{f}_4^{\text{m-sel}}(\mathbf{X}, t_i, u_i)$	130	6630	56.89
$\mathbf{f}_6^{\text{sel}}(\mathbf{X}, t_i, u_i)$	312	13728	59.04
$\mathbf{f}_{1-6}^{\text{joint}}$	325	14235	**59.85**
$\mathbf{f}_{3-6}^{\text{joint}}$	351	15249	59.81

Table 4.2: Comparison of phone recognition performance of SCRF with different state feature functions. The accuracies are given for the development set. The transition factor does not depend on any input and thus only acts a bias. The "*dimensions*" column denotes the dimensionality of the vector returned by the corresponding feature function. Details about the listed feature functions are given in the text.

Simply concatenating all input vectors of a segment to form the feature vector is not a good idea for two reasons: First this approach is prone to over-fitting. As depicted in the histogram in Figure 4.3 most of the segments are of shorter length than the maximum segment length. For a segment shorter than $L_{\max}$ the remaining positions have to be padded with zeros. Then the feature vector of a segment would depend on its length. The dataset contains many different realisations of the same phone. They differ in their MFCC coefficients as well as in their length. This makes it hard to obtain model parameters that are generalizing well. The second reason is that the resulting feature vector would be very large ($31 \cdot 39 = 1209$ elements).

For those two reasons we did not evaluate this feature function. A first very naive way to build a fixed size feature vector is to simply average the input vectors of each segment. This results in a very poor phone recognition accuracy of 30.00%. The delta and double delta values in the data were averaged as well, which does not make any sense from a signal processing point of view. The deltas convey information about changes in the cepstrum. By averaging the delta vectors of a segment this information gets lost.

Therefore it is reasonable to consider the deltas at the boundaries of a phone. Thus our second approach only averages the MFCCs of a segment and includes the deltas of the first and the last frame of the segment. This feature function is denoted as $\mathbf{f}_1$ in Table 4.2. Applying this feature function the phone recognition accuracy increased to 57.24%. This shows that the deltas at the segment boundaries are very important for phone recognition.

This raises the question of how important the averaged MFCCs are. The feature function just utilizing the first and the last input vector of a segment achieves a

phone recognition accuracy of 54.54%. This shows that the frames from within a segment are important as well.

Halberstadt and Glass [9] used three MFCC averages computed approximately over segment thirds. Using three averaged MFCCs and the first and last frame completely further increased the recognition accuracy to 58.17%. This feature function is denoted as $\mathbf{f}_3$.

In general we use the notation $\mathbf{f}_i(\mathbf{X}, t_i, u_i)$ to denote a feature function that takes the first and last input vectors of a segment and depending on the subscript i the MFCCs of the segment are averaged in i groups. When averaging the MFCCs of a segment in 2 groups the obtained result was 57.09%. This is a little worse than for the $\mathbf{f}_1(\mathbf{X}, t_i, u_i)$ feature function.

A side effect of using more input groups is that the training time increases. This is due to the introduced overhead in constructing the feature vector. Moreover the feature vector is increased in size and as a consequence the number of model parameters is also increased.

n	#dimensions	phone accuracy
0	78	54.54
2	104	56.15
3	117	56.74
4	130	56.89
5	143	56.85

Table 4.3: Comparison of development set phone recognition accuracies for the $\mathbf{f}_n^{\text{m-sel}}$ state feature functions. Those feature functions take the first and the last input frames of the segment and are additionally including n MFCC frames from within the segment. The positions of selected frames are determined from a linear range over the segment length.

We evaluated another approach of constructing a feature vector: Instead of averaging the MFCCs of a segment, we included some selected ones into the feature vector. The MFCCs were selected from a linear range over the segment duration. The first and last frames again were kept as a whole (*i.e.* MFCCs and deltas). The recognition results for this feature function for a varying number of selected MFCCs are given in Table 4.3. This type of features is not as effective as the features using the averaged MFCCs.

This leads to yet another approach of constructing a fixed size feature vector from a variable length segment. Instead of only using selected MFCCs we evaluated the use of selected complete input frames. That means not only the MFCCs, but also the deltas are part of the feature vector. This feature function is denoted as $\mathbf{f}_n^{\text{sel}}(\mathbf{X}, t_i, u_i)$ in the overview Table 4.2. The subscript variable n denotes the number of selected complete frames from within the segment. Recall that this feature function additionally includes the first and the last frames of a segment. Table 4.4 gives an overview of the recognition results of this type of feature function.

Finally, we evaluated the recognition performance of a feature function that unifies the averaged MFCC $\mathbf{f}_i$ feature function and the selected frames feature function $\mathbf{f}_n^{\text{sel}}$. Thus those feature functions concatenate the first and the last observation vectors of a segment with n selected observation vectors from within the segment and the MFCCs averaged in i parts. In Table 4.2 this type of feature function is denoted as $\mathbf{f}_{i-n}^{\text{joint}}$, where i denotes the number of averaged MFCC groups and n denotes the

number of selected complete frames.

n	#dimensions	phone accuracy
0	78	54.54
3	195	57.80
4	234	58.34
5	273	58.64
6	312	**59.04**
7	351	58.93

Table 4.4: Comparison of development set phone recognition accuracies for the $\mathbf{f}_n^{\text{sel}}$ state feature functions. Those feature functions take the first and the last input frames of the segment and are additionally including n frames from within the segment. The positions of selected frames are determined from a linear range over the segment length.

4.5 Segment length bias

In the previous section we identified a state feature function suitable for phone recognition. When making a prediction the model hypothesizes over all possible segment lengths and has to make a decision about it. We pursue the idea of supporting that process by including a segment length bias.

Inspecting the example utterance in Figure 4.1, one can see that the lengths of the individual segments are very different. We already inspected the distribution of segment lengths of the entire training dataset. Now we take a closer look and inspect the distribution of segment lengths for each individual phone class.

Figure 4.4 shows that the distributions of segment lengths are specific to each phone class. Therefore a segment length bias for each individual phone class appears to be beneficial as this information should support the process of determining the proper segment length.

This is also underlined by empirically collected data. The averaged predicted segment length in the validation dataset with the $\mathbf{f}_1(\mathbf{X}, t_i, u_i)$ state features was 8.776. In contrast the average segment length in the training data is 6.819 with a standard deviation of 4.251. Therefore we conclude that the model predicts too long segments and thus does not recognise some phones.

We explored three ways of including a segment length bias to the model. The first evaluated method comes from Halberstadt and Glass [9]. They included the logarithm of the segment length to their feature vector. By $\mathbf{f}_i^{\text{loglen}}(\mathbf{X}, t_i, u_i)$ we denote a feature function that does the same as the $\mathbf{f}_i(\mathbf{X}, t_i, u_i)$ feature function and additionally includes the logarithm of the segment duration. Thus the SCRF is extended by $|\mathcal{Y}|$ parameters. We evaluated the performance of this function for $i = 1$. Using this type of segment length bias increases the recognition accuracy by more than 2%. Inspecting the learned parameters, one finds that the weight for the logarithmic length is negative for each phone class. Multiplied with a negative weight the logarithm of the segment length is decreasing for larger arguments and thus making longer segment less favourable. Figure 4.5 plots the learned weights over the segment length histograms of four selected phone classes.

A different approach involves extending the state factor by another indicator

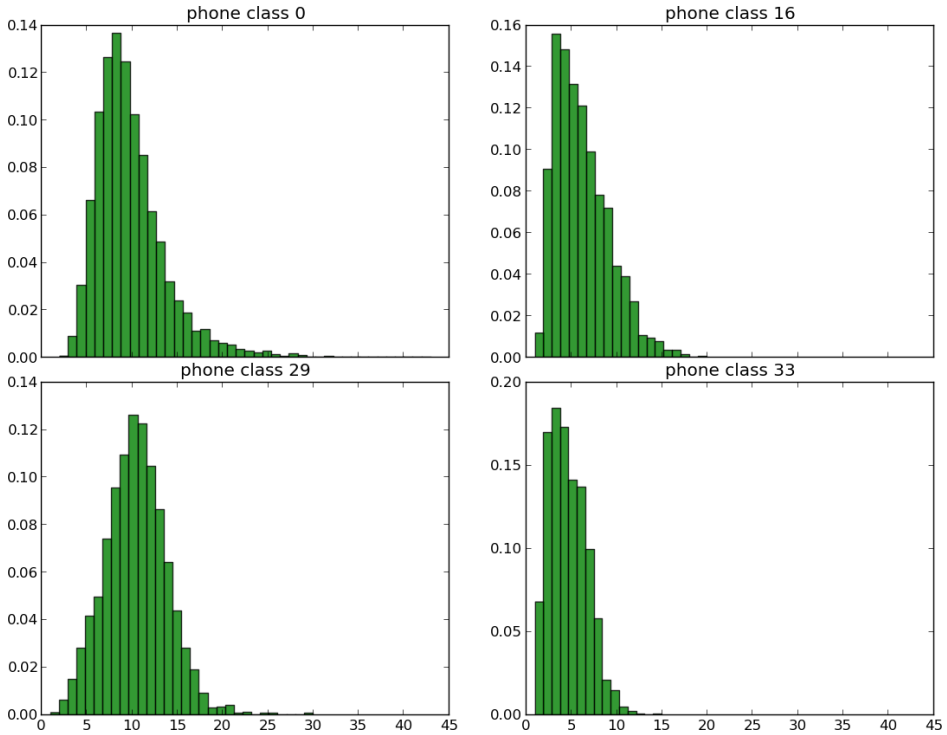


Figure 4.4: Distribution of segment lengths for selected phone classes in the TIMIT training dataset.

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	length bias	#parameters	average-length	accuracy
$\mathbf{f}_1(\mathbf{X}, t_i, u_i)$	no length bias	5109	8.776	57.24
$\mathbf{f}_1(\mathbf{X}, t_i, u_i)$	distribution	5148	8.763	57.27
$\mathbf{f}_1(\mathbf{X}, t_i, u_i)$	indicator	6318	8.388	59.52
$\mathbf{f}_1(\mathbf{X}, t_i, u_i)$	log length	5148	8.195	59.63

Table 4.5: Overview of phone recognition performance of SCRF for different types of segment length biases. The accuracies are given for the development set. Again there is only transition bias. The table shows a explicit correlation between the average predicted segment length and the recognition accuracy.

feature:

$$\phi(y_i, \mathbf{X}, t_i, u_i) = \sum_y \left(\mathbf{w}_y^T \mathbf{f}(\mathbf{X}, t_i, u_i) + \lambda_y + \nu_{yl_i}\right) \delta_{yy_i}, \tag{4.1}$$

where ν_{yl_i} is the weight of the length indicator feature. Note that there are $|\mathcal{Y}| \times L_{\max}$ such parameters. In order to be able to make comparisons to the previous methods we evaluated the segment length indicator feature together with the $\mathbf{f}_1(\mathbf{X}, t_i, u_i)$ state features. The duration indicator feature improves the recognition result by about 2% and lowers the average predicted segment length (*cf.* Table 4.5).

Another very different approach is to estimate the distribution of segment lengths for each phone from the training dataset. This is done by determining the histogram of the segment lengths for each phone class. See Figure 4.4 for a plot of histograms for some selected phone classes. For each hypothetical segment of length l and label y the corresponding histogram value is multiplied by the per phone class trainable weight and added to the local potential $\phi(y_i, \mathbf{X}, t_i, u_i)$. This introduces $|\mathcal{Y}|$ new

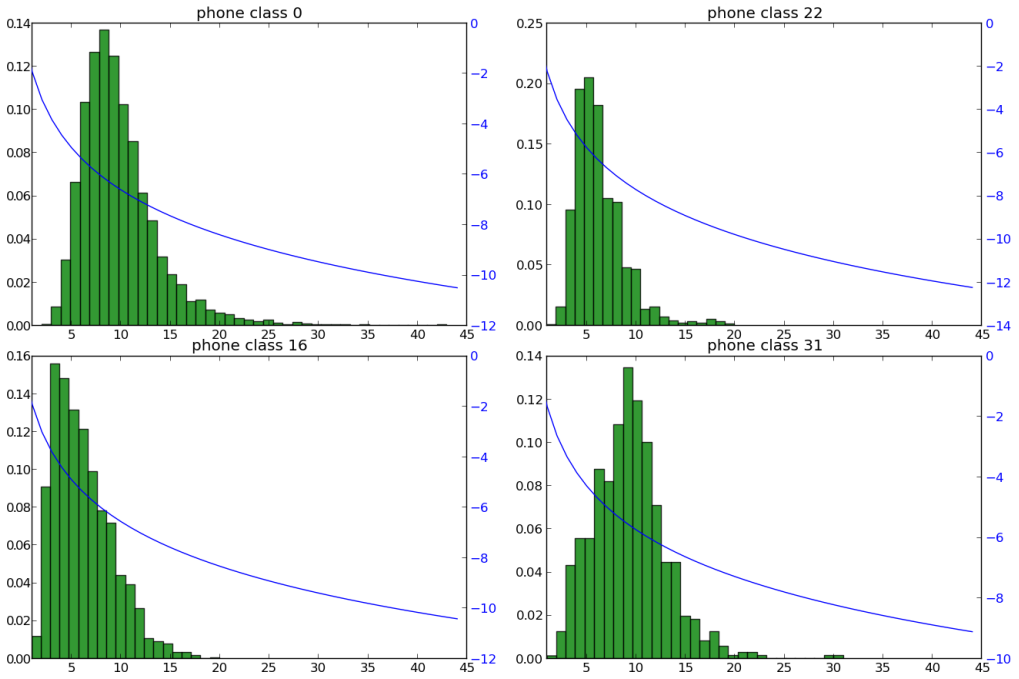


Figure 4.5: Histograms of segment lengths for selected phone classes in the TIMIT training dataset. The curves are the logarithms of the segment durations multiplied by the learned parameters.

parameters. As it turned out this form of segment length bias did not increase the recognition performance at all.

Table 4.5 gives an overview of all tested length bias methods. The results confirm our first conjecture that the recognition accuracy of the model is linked to the averaged predicted segment length. The recognition performance decreases with increasing deviation of the average predicted segment length from the true average segment length. The duration indicator works almost as well as the including the logarithm of the segment duration. Since the latter is much simpler to implement and performs even slightly better we selected that approach.

Finally we also evaluated the performance of the logarithm of the segment duration for best performing feature functions from the previous section. Table 4.6 summarizes those results.

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	length bias	#parameters	phone accuracy
$\mathbf{f}_3(\mathbf{X}, t_i, u_i)$	log length	6162	60.30
$\mathbf{f}_{1-6}^{\text{joint}}(\mathbf{X}, t_i, u_i)$	log length	14274	61.29
$\mathbf{f}_{3-6}^{\text{joint}}(\mathbf{X}, t_i, u_i)$	log length	15288	**61.43**

Table 4.6: Overview of phone recognition performance of SCRF for best performing state features including the logarithm of the segment length as segment length bias. The accuracies are given for the development set. Again there is only transition bias.

4.6 Transition features

So far we have only considered input dependent state features and input independent transition bias. In [20] Zweig has found out that contextual information around segment boundaries is particularly important for speech recognition. Therefore we explored the use of input dependent transition features.

In order to make efficient use of boundary features in the SCRF we have defined the transition factor to be independent of the current segment length. Therefore the process of determining a feature function is strongly simplified. The transition feature function $\mathbf{g}_n(\mathbf{X}, t_i)$ utilizes n observation vectors from the segment boundary. Half of the frames are from the segment left of the boundary and the other half of the frames are from segment to the right.

We have determined the optimal number of boundary frames by keeping the state feature function $\mathbf{f}_3^{\text{loglen}}$ fixed in those experiments. Table 4.7 gives an overview of the recognition results. A side effect of adding input dependent transition features is that the number of model parameters gets drastically increased. The best result is obtained for 10 frames of boundary context. This input dependent transition factor leads to an improvement of almost 10% over just using transition bias. More than 10 frames did not lead to a further improvement of the recognition performance.

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	#frames	#parameters	phone accuracy
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	4	243438	68.91
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	6	362076	70.08
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	8	480714	70.40
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	10	599352	**70.63**
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	12	717990	70.60

Table 4.7: Overview of phone recognition performance for different number of frames at the boundary and always the same state features. The accuracies are given for the validation set. Half of the frames are from the segment left of the boundary and the other half of the frames are from segment to the right.

This poses the question of how much influence the state features have on the recognition performance. Therefore we examined the recognition performance with the transition feature function using 10 frames at segment boundary and without any state feature function. The recognition performance in this configuration was 66.93% and thus about 4% worse than with the $\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$ state features. This shows that even in the presence of input dependent transition features the state features are still important!

Next we examine the optimal state and transition feature function combination. In the previous section we have identified that the $\mathbf{f}_{3-6}^{\text{joint}}(\mathbf{X}, t_i, u_i)$ state feature function delivered the best performance. Therefore we also determined the recognition performance of those state features in conjunction with the input dependent transition features. As it turned out those features provided no significant improvement of recognition performance over the $\mathbf{f}_3^{\text{loglen}}$ state feature function. As it seems those features do not provide any additional information that could support the recognition process of the model. Table 4.8 compares the recognition performance for different state feature functions and different number of boundary context frames as input dependent transition features.

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	#frames	#parameters	phone accuracy
$\mathbf{f}_1^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	4	242424	68.80
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	4	243438	68.91
$\mathbf{f}_1^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	6	361062	69.63
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	6	362076	70.08
0	10	594750	66.93
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	10	599352	70.63
$\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	10	608478	**70.83**

Table 4.8: Comparison of the influence of different state features with constant number of boundary context frames on the phone recognition performance. The accuracies are given for the validation set.

4.7 Maximum segment length

This section examines the influence of the $L_{\max}$ parameter on the phone recognition result. A smaller maximum segment length parameter has the benefit of making training and decoding faster. On the other hand this reduces the ability of the model to predict longer segments. Thus we have to trade off and seek an acceptable solution.

$L_{\max}$	coverage	time/epoch	phone accuracy
18	98.05%	26.8 min	68.80
21	99.15%	28.6 min	69.47
24	99.62%	29.9 min	69.83
31	99.95%	34.1 min	**70.08**
44	100%	41.2 min	70.03

Table 4.9: Overview of phone recognition performance for different selections of the $L_{\max}$ parameter. The accuracies are given for the validation set. In all experiments $\mathbf{f}_3^{\text{loglen}}$ was used as state feature function and 6 frames from the segment boundary as input dependent transition features. The percentages in the coverage column give how many percent of segments in the training set are covered by the corresponding choice of $L_{\max}$.

Table 4.9 summarizes the phone recognition results for different choices of $L_{\max}$. All these experiments were conducted with the $\mathbf{f}_3^{\text{loglen}}$ state feature functions. The transition feature function utilized 6 observation vectors form the segment boundary as described in the previous section. The table also states for each choice the percentage of covered segments in the training set. Larger choices of $L_{\max}$ increase the coverage and also the recognition performance up to certain point. At approximately $L_{\max} = 31$ the optimal choice is found. With this maximum segment length parameter more than 99.9% of segments in the training set are covered.

4.8 L^2-SCRF

In this section we present the phone recognition result of the L^2-SCRF model introduced in Section 3.9. Recall that this model has a runtime that is quadratic in

the maximum segment length $L_{\max}$. Depending on the used feature functions the default SCRF trained with the recursions initially presented in the Sections 3.4, 3.5, 3.7 has a training time of one to several hours hours per epoch. The asymptotic runtime of the L^2-SCRF is larger by a factor of $L_{\max}$. Recall that for phone recognition on TIMIT we have determined that the optimal parameter of $L_{\max} = 31$. Thus one can expect that the runtime of a single training epoch for the L^2-SCRF is one to several days! In order to still be able to evaluate whether the context of two segments has a positive influence on the phone recognition performance and has an advantage over boundary context we set $L_{\max} = 10$. That keeps the runtime of the model reasonable. This however requires a special way of processing the input data and the model output. Thus we have split all phone segments in the training data longer than $L_{\max} = 10$ into multiple segments. When predicting, adjacent segments of the same label were merged together.

We compared the phone recognition performance of the L^2-SCRF model with the efficient SCRF were both used the same state feature function. As transition feature function the former uses the context of two segment processed by the state feature function and the latter uses 10 frames from the segment boundary. In order to make an accurate comparison between the two models we have trained the efficient SCRF using the $L_{\max} = 10$ parameter and the same input preprocessing and output post-processing.

model	$\mathbf{f}(\mathbf{X}, t_i, u_i)$	phone accuracy
efficient SCRF	$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	69.91
L^2-SCRF	$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	68.27

Table 4.10: Comparison of phone recognition accuracy of the L^2-SCRF vs. the efficient SCRF. Both models use the same state feature functions. The L^2-SCRF utilizes the context of two segments, while the efficient SCRF uses 10 frames from the segment boundary as input dependent transition features. The accuracies are given for the development set.

From Table 4.10 one can see that the explicit context of two phones does not provide an improvement over segment boundary context. Hence the expensive L^2-SCRF model does not pay-off and the efficient SCRF is to be preferred. It is interesting to note that using the small $L_{\max}$ parameter and merging model outputs results in a decrease of phone recognition accuracy for the efficient SCRF by about 0.7% compared to using $L_{\max} = 31$ and applying no special handling of the outputs.

4.9 Hidden layer SCRF

In this section we present results from phone recognition experiments with the hidden layer SCRF. With the number of neurons a hidden layer introduces a new hyper parameter to the model. Therefore, we need to find an optimal choice for that parameter.

In the experiments conduced with the hidden layer SCRF we selected the hyperbolic tangent tanh as non-linear activation function. According to common practice in neural networks we initialized the hidden layer weights by sampling from a uniform distribution from a range that depends on the number of connections in the layer.

First, we examine the effect of hidden layer state factor and determine the corresponding approximate optimal number of hidden neurons. For those experiments the input dependent transition features comprised 10 frames of boundary context and are used in plain. Both feature functions were kept fixed in order to determine the influence of the number of hidden neurons on the result. Above, we have found out that in conjunction with the boundary input dependent transition factor two particular state feature functions achieved the best results. The first and simpler feature function denoted as $\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$ averages the MFCCs within a segment in three parts and adds the first and the last frames of the segment. Furthermore the logarithm of the segment length is added. The second feature function $\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$ additionally includes 6 complete frames from within the segment. We now discuss the influence on the phone recognition accuracy when using those state feature functions in connection with a hidden layer. For both functions we determined the approximate optimal number of hidden neurons.

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	#neurons	#parameters	phone accuracy
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	150	618300	72.92
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	300	641850	73.89
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	400	657550	73.95
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	500	673250	73.95
$\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	200	672950	73.63
$\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	400	751150	74.44
$\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	600	829350	75.12
$\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	800	907550	**75.50**
$\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	1000	985750	75.18

Table 4.11: Comparison of the development set phone recognition accuracies of SCRFs with hidden layer state features, with different numbers of neurons. The transition feature function used in all models in this table were the 10 complete frames from the segment boundary. The transition features are used in plain which means that they were not passed through a hidden layer.

Table 4.11 gives an overview of phone recognition performances for those two state feature functions and different numbers of neurons. For the $\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$ feature function the approximate optimal number of hidden neurons is about 400 to 500. Using those features as inputs to the hidden layer the phone recognition accuracy improves by more than 3% over the linear model without a hidden layer. As it turns out the $\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$ feature function serving as input to the hidden layer leads to an even larger improvement in phone recognition accuracy. With 800 neurons a performance increase of $\approx 5\%$ over the linear model is achieved.

Next, we discuss the optimal number of neurons for a hidden layer underneath the input dependent transition factor. As feature function we used the one identified as optimal in Section 4.6. The 10 observation frames from the segment boundary were used as inputs to the hidden layer. The state feature function was used without a hidden layer and was kept fixed in thos experiments. This way we determined the best number of neurons for the hidden layer transition factor. As presented in Table 4.12 the best result was obtained for 1000 hidden neurons. The non-linear transition features improved the recognition result by a bit more than 3% compared to the linear version of the SCRF.

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	#frames	#neurons	#parameters	phone accuracy
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	10	500	673250	73.85
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	10	1000	751750	**74.14**
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	10	1500	830250	73.91

Table 4.12: Comparison of the development set phone recognition accuracies of SCRFs with hidden layer transition features, for different number of neurons. The transition feature function used in all models in this table were the 10 complete frames from the segment boundary. The state feature function $\mathbf{f}_3^{\text{loglen}}$ was used without a hidden layer.

Now that we have a good parameter setting for the state and the transition feature functions, we are able to combine those two and check the final recognition performance. Table 4.13 gives the result of the two models with the different state feature functions on the development set. As we can see the use of both non-linear state and transition factors led to an overall improvement compared to the use of only one non-linear factor. The final model configuration of non-linear state and transition factor, using the $\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$ state feature function led to an improvement of more than 5% compared to the linear SCRF.

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	#neurons ϕ	#neurons ψ	#parameters	phone accuracy
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	400	1000	1975360	74.99
$\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	800	1000	2225360	**76.38**

Table 4.13: Comparison of the development set phone recognition accuracies of hidden layer SCRFs. The first column gives the used state feature function. The transition feature function uses 10 frames from the segment boundary. The second and third columns give the number of hidden neurons used for the state and transition features respectively.

4.10 Training without alignments

Finally we discuss the training of the SCRF without given alignments. We used selected model configurations determined above and trained them again by optimizing $p(\mathbf{Y}|\mathbf{X})$ instead of directly optimizing $p(\mathbf{Y}, \mathbf{A}|\mathbf{X})$. As already pointed out in Section 3.13, computing the gradient of the numerator of the objective function takes asymptotically $T \cdot L_{\max}$ more time when marginalizing out all possible alignments than if the alignment is given. Hence also the time per training epoch is significantly increased. In the experiments conducted in this work the training time per epoch increased by a factor of about two to five. As pointed out in [10] another theoretical side effect of this method is that the objective is no more convex. Hence we used a different learning rate schedule to train those models. The learning rate was not annealed to a minimum. The learning rate for the first three epochs was set to $\eta = 0.005$. This allowed the model parameters to change to something meaningful compared to the random initial state. Then all subsequent epochs were trained with a constant learning rate of $\eta = 0.001$.

For the experiments conducted in this section we do not rely on the alignments in the training data at all. This means that the strategy of input value normalization

used in the previous experiments is not available any more. This strategy relied on the label alignments in the training data. We tried to normalize all input vectors in the dataset to have zero mean and unit standard deviation. But this did not work better than not normalizing the inputs at all. So all experiments carried out to optimize $p(\mathbf{Y}|\mathbf{X})$ have not used any normalization of inputs.

$\mathbf{f}(\mathbf{X}, t_i, u_i)$	$\mathbf{g}(\mathbf{X}, t_i)$	$p(\mathbf{Y}, \mathbf{A}\|\mathbf{X})$	#epochs	$p(\mathbf{Y}\|\mathbf{X})$	#epochs
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	*n.a.*	60.30	26	60.44	91
$\mathbf{f}_{3-6}^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	*n.a.*	61.43	15	61.31	65
$\mathbf{f}_3^{\text{loglen}}(\mathbf{X}, t_i, u_i)$	10	70.63	17	70.53	110

Table 4.14: Comparison of phone recognition performances of various SCRF model configurations. The $p(\mathbf{Y}, \mathbf{A}|\mathbf{X})$ column gives the accuracies when the model was trained with the given phone boundaries. The column to the left shows after how many epochs of training have been performed to obtain the result. The $p(\mathbf{Y}|\mathbf{X})$ column gives the results of the same model trained without the boundary information. The column to the left indicates how many epochs of training it took to achieve the stated result. All accuracies are given for the development set.

Table 4.14 summarizes and compares the results of the models trained without alignments to the models trained with them. There are two important observations here. First, training the SCRF without alignments in the way introduced in Section 3.13 it is possible to achieve comparable results. This is a proof of concept, that the algorithm developed to marginalize out all alignments is actually working in practice. Second, it takes much more epochs of training to achieve this result. Thus if a (speech) dataset has the alignment of the transcriptions available it is of advantage to use them. However the model does not have to rely on them in order to achieve the same results.

4.11 Comparison of results

We conclude this chapter by summarizing the achieved results and comparing them to related and other work on the TIMIT database. Table 4.15 gives for each model type an overview of the best phone recognition accuracies on the development and the core-test set achieved in this work.

model	dev acc	core test acc
linear SCRF - transition bias (Section 4.5)	61.43	60.39
linear SCRF - boundary context (Section: 4.6)	70.83	69.77
hidden layer SCRF (Section: 4.9)	76.38	75.07

Table 4.15: Best performing SCRF model configurations. The first row shows the result of the SCRF with the $\mathbf{f}_{3-6}^{\text{loglen}}$ state features and only transition bias. The second row gives the result from the SCRF with the same state features and additional the context of 10 frames from the segment boundary as input dependent transition factor. The last row gives the best performing hidden layer SCRF with the same state features and 800 neurons and 1000 neurons for the same input dependent transition feature function.

We start our comparison of results by examining the recognition accuracy achieved

by related work. In [20], Zweig has used vector quantization features in conjunction with a segmental conditional random field and achieved 66.9% accuracy on the core-test set. The linear SCRF of this work which directly utilized MFCCs and empirical determined feature functions achieved a result almost 3% better. He and Fosler-Lussier [10] achieved with a SCRF that used phone posterior features 73.5% accuracy on the core test set. Since the phone posteriors are the outputs of a trained neural network it does not make sense to compare with our linear model. Hence we need to compare with our best hidden layer SCRF model. The hidden layer SCRF trained with backprop achieved a result more than 1% better than the phone posterior SCRF of He and Fosler-Lussier.

model	features	acc dev	acc core test
SCRF[20]	vector quantization	67.6%	66.9%
frame CRFs[10]	phone posteriors	72.6%	70.8%
efficient SCRF[10]	phone posteriors	76.0%	73.5%
Deep belief networks[16]	MFCC	78.00%	77.00%
Deep segmental NN[1]	spectrogram	*n.a.*	77.1%
hybrid HMM - CNN[2]	spectrogram	*n.a.*	79.93%
CTC deep-RNN[7]	spectrogram	*n.a.*	81.6%
Deep-RNN[7]	spectrogram	*n.a.*	82.3%

Table 4.16: Comparison of phone recognition performances of various different approaches sorted by their TIMIT core test set accuracies. Models: Convolutional neural network (CNN); Connectionist temporal classification (CTC); Hidden Markov model (HMM); Neural network (NN); Recurrent neural network (RNN)

Recent and more successful approaches of phone recognition involve deep neural network architectures. Table 4.16 gives an overview of different selected phone recognition approaches and their accuracies on TIMIT. To the best of the author's knowledge deep recurrent neural networks (RNNs) mark with 82.3% accuracy on the core test set the current state of the art. In this approach two RNN concepts were unified: Long Short-Term Memory (LSTM) and a deep architecture. The best CTC based deep LSTM RNN in [7] achieved 81.6% accuracy. For details about the different methods, the interested reader is referred to the references.

However, we belief that the combination of segmental conditional random fields with more expressive factors will lead to increased performance and hence close the gap to the results summarized in Table 4.16.

Chapter 5

Conclusion

5.1 Summary

We have presented segmental conditional random fields as discriminative segmental generalization of conditional random fields. We presented the formulas and algorithms required to understand and implement the model. Moreover we extended the linear SCRF by introducing a hidden layer. With the L^2-SCRF we introduced a runtime demanding model that has the context of two adjacent segments available. Additionally we developed a method for training SCRFs without requiring that the labelling sequences in the training set are aligned to the observations. We applied SCRFs to the task of phone recognition and trained the model on TIMIT using the given phone boundaries and without. We have empirically determined feature functions that are well suited for phone recognition. We have seen that the choice of feature functions is critical for segmental conditional random fields. The best linear model configuration achieved a phone accuracy of 69.77% on the TIMIT core test set. The state feature function of this model concatenates the first and last observation vectors of a segment, the MFCCs of the segment averaged in three parts and 6 observation vectors from within the segment. Additionally as a segment length bias the logarithm of the segment length was added. As transition feature function we used 10 observation vectors from the segment boundary. Moreover we have conduced experiments with the hidden layer segmental conditional random field and achieved a core test set accuracy of 75.07%. For that model the used state and transition feature functions have been the same. The backpropagation trained hidden layer SCRF outperformed other published approaches involving segmental conditional random fields in connection with vector quantization or phone posterior features. Those experiments have been performed using the alignments of the target labelling sequences. Furthermore we have shown that it is possible to achieve the same recognition results when training the model without utilizing those alignments. This shows that the algorithm we have introduced for marginalizing out all alignments consistent with an observation and a labelling sequence also works in practice.

5.2 Future work

As we have seen, the introduction of a hidden layer has shown benefits in recognition performance. We believe that the gap to state of the art results can be closed by including more expressive models than the simple fully-connected hidden layer. For

example in [2] with a hybrid convolutional neural network - HMM approach a phone recognition accuracy on the TIMIT core test set of $\approx 80\%$ was achieved. We are confident that the combination of convolutional layers and the segmental conditional random field will be beneficial for the performance. However this makes it necessary to switch the preprocessing to spectrogram features for example. MFCCs are not suited for convolutional neural networks as they are computed from a discrete cosine transform which decorrelates the frequency bands.

Segmental conditional random fields with higher order factors are problematic because their runtime not only scales exponentially in the size of the label space, but also in the maximum segment length parameter. So the strategy of sparse transitions proposed and used for conditional random fields does not help in the segmental context.

Until now for the SCRF model it is necessary to manually find a heuristic to transform a variable number of inputs into a fixed sized feature vector. Therefore another interesting direction of future work would be to develop a training algorithm that allows to learn the feature functions from data. For example recurrent neural networks learn time dependent correlations between data automatically. A method for learning a feature function could probably work in a similar way.

Appendix A

SCRF parameter gradients

A.1 One label and input dependent parameter

Recall that $\mathbf{w}_y$ is a $D-$dimensional vector. Therefore we need to determine the gradient with respect to each component of $\mathbf{w}_y$.

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{yd}} = \frac{\sum_{i=1}^{|\mathbf{S}|} \log \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} - \frac{\log Z(\mathbf{X})}{\partial w_{yd}}$$

First we compute the derivative of the left part of the above expression. Inserting the definition of the potential:

$$\log \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i) = \psi(y_i, y_{i-1}, \mathbf{X}, t_i) + \phi(y_i, \mathbf{X}, t_i, u_i)$$

Only $\phi(y_i, \mathbf{X}, t_i, u_i)$ the right term of the potential function depends on w_{yd} and therefore the derivative of $\psi(y_i, y_{i-1}, \mathbf{X}, t_i)$ is zero. Inspecting the definition of $\phi(y_i, \mathbf{X}, t_i, u_i)$ one can see that only if the training label $y_i = y$ the term is non-zero with respect to w_{yd}:

$$\frac{\partial \phi(y_i, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} = \frac{\partial \left(\mathbf{w}_y^T \mathbf{f}(\mathbf{X}, t_i, u_i) + \lambda_y\right) \delta_{yy_i}}{\partial w_{yd}} = \mathbf{f}_d(\mathbf{X}, t_i, u_i)\delta_{yy_i}$$

Therefore the derivative of the left part of the above expression is simply:

$$\frac{\partial \sum_{i=1}^{|\mathbf{S}|} \log \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} = \sum_{i=1}^{|\mathbf{S}|} \mathbf{f}_d(\mathbf{X}, t_i, u_i)\delta_{yy_i}$$

The gradient of the partition function is a little bit more complicated:

$$\frac{\partial \log Z(\mathbf{X})}{\partial w_{yd}} = \frac{\partial \log \sum_{\mathbf{S}'} \prod_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}}$$

To compute this derivative we need to apply the chain rule, *i.e.* $F(x) = f(g(x))$, $F'(x) = f'(g(x))g'(x)$. In this particular case the outer function is the logarithm $f(g) = \log g$. The derivative of the outer function is $f'(g) = \frac{1}{g} = \frac{1}{Z(\mathbf{X})}$.
The inner function $g(x) = \sum_{\mathbf{S}'} \prod_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)$ is derived as:

$$\frac{\partial \sum_{\mathbf{S}'} \prod_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} = \sum_{\mathbf{S}'} \frac{\partial \prod_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} = \dots \quad \text{(A.1)}$$

Next, we need to apply the general product rule. For a function of products $f = \prod_{i=1}^{n} f_i$ the derivative is $f' = \sum_{i=1}^{n} f_i' \prod_{\substack{j=1 \\ j\neq i}}^{n} f_j$. In order to compute the derivative of the potential we need again the chain rule because $\Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)$ is given in exponential form. For readability we have substituted $x := \psi(y_i, y_{i-1}, \mathbf{X}, t_i) + \phi(y_i, \mathbf{X}, t_i, u_i)$.

$$\begin{aligned}\frac{\partial \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} &= \frac{\partial \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial x} \frac{\partial x}{\partial w_{yd}} \\ &= \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i) \mathbf{f}_d(\mathbf{X}, t_i, u_i) \delta_{y y_i}\end{aligned}$$

Having obtained this derivative we can now use the product rule and continue with Equation A.1:

$$\begin{aligned}\cdots &= \sum_{\mathbf{S}'} \frac{\partial \prod_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{yd}} \\ &= \sum_{\mathbf{S}'} \sum_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, l_i) \mathbf{f}_d(\mathbf{X}, t_i, u_i) \delta_{y y_i} \prod_{\substack{j=1 \\ j\neq i}}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j) \\ &= \sum_{\mathbf{S}'} \sum_{i=1}^{|\mathbf{S}'|} \mathbf{f}_d(\mathbf{X}, t_i, u_i) \delta_{y y_i} \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j).\end{aligned}$$

Therefore the gradient of the logarithm of the partition function with respect to (w.r.t.) w_{yd} is:

$$\frac{\partial \log Z(\mathbf{X})}{\partial w_{yd}} = \frac{1}{Z(\mathbf{X})} \sum_{\mathbf{S}'} \sum_{i=1}^{|\mathbf{S}'|} \mathbf{f}_d(\mathbf{X}, t_i, u_i) \delta_{y y_i} \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j).$$

Having obtained all partial results we can write down the gradient of the log-likelihood with respect to w_{yd}:

$$\begin{aligned}&\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{yd}} = \\ &\sum_{i=1}^{|\mathbf{S}|} \mathbf{f}_d(\mathbf{X}, t_i, u_i) \delta_{y y_i} - \frac{1}{Z(\mathbf{X})} \sum_{\mathbf{S}'} \sum_{i=1}^{|\mathbf{S}'|} \mathbf{f}_d(\mathbf{X}, t_i, u_i) \delta_{y y_i} \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j).\end{aligned}$$

A.2 One label dependent bias parameter

For the bias parameter λ_y the gradient is even simpler because the derivative of the potential is simply:

$$\frac{\partial \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, l_i)}{\partial \lambda_y} = \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i) \delta_{y y_i}$$

Therefore, the partial derivative of $\log p(\mathbf{S}|\mathbf{X})$ with respect to λ_y is

$$\frac{\partial \log Z(\mathbf{X})}{\partial \lambda_y} = \sum_{i=1}^{|\mathbf{S}|} \delta_{y y_i} - \frac{1}{Z(\mathbf{X})} \sum_{\mathbf{S}} \sum_{i=1}^{|\mathbf{S}|} \delta_{y y_i} \prod_{j=1}^{|\mathbf{S}|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j).$$

A.3 Two label and input dependent parameter

Here we derive the partial derivative of $\log p(\mathbf{S}|\mathbf{X})$ with respect to every component of the weight vector $v_{yy'd}$, *i.e.*

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial v_{yy'd}} = \frac{\sum_{i=1}^{|\mathbf{S}|} \log \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial v_{yy'd}} - \frac{\log Z(\mathbf{X})}{\partial v_{yy'd}}. \tag{A.2}$$

The method and the rules of derivation are the same as for the partial derivation w.r.t. w_{yd} so here are only those steps that are different. For the left hand side of Equation A.2 the derivative of the logarithm of the potential function is different:

$$\frac{\partial \psi(y_i, y_{i-1}, \mathbf{X}, t_i)}{\partial v_{yy'd}} = \mathbf{g}_d(\mathbf{X}, t_i)\delta_{yy_i}\delta_{y'y_{i-1}}.$$

The initial steps to derive the right hand side of Equation A.2 w.r.t to $v_{yy'd}$ are the same as w.r.t w_{yd} (*cf.* Section A.1). First the chain rule is applied and then the general product rule. The derivative of the potential function is different, because we are now considering the derivative w.r.t $v_{yy'd}$. For readability we define again $x := \psi(y_i, y_{i-1}, \mathbf{X}, t_i) + \phi(y_i, \mathbf{X}, t_i, u_i)$.

$$\begin{aligned}\frac{\partial \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial v_{yy'd}} &= \frac{\Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial x}\frac{\partial x}{\partial v_{yy'd}} \\ &= \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)\mathbf{g}_d(\mathbf{X}, t_i)\delta_{yy_i}\delta_{y'y_{i-1}}\end{aligned}$$

Having obtained this derivative we can now use the product rule and continue the derivation:

$$\begin{aligned}&\sum_{\mathbf{S}'} \frac{\partial \prod_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial v_{yy'd}} \\ =&\sum_{\mathbf{S}'}\sum_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)\mathbf{g}_d(\mathbf{X}, t_i)\delta_{yy_i}\delta_{y'y_{i-1}} \prod_{\substack{j=1 \\ j\neq i}}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j) \\ =&\sum_{\mathbf{S}'}\sum_{i=1}^{|\mathbf{S}'|} \mathbf{g}_d(\mathbf{X}, t_i)\delta_{yy_i}\delta_{y'y_{i-1}} \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j)\end{aligned}$$

Putting all the pieces together the gradient of the logarithm of the partition function is given by

$$\frac{\partial \log Z(\mathbf{X})}{\partial v_{yy'd}} = \frac{1}{Z(\mathbf{X})}\sum_{\mathbf{S}'}\sum_{i=1}^{|\mathbf{S}'|} \mathbf{g}_d(\mathbf{X}, t_i)\delta_{yy_i}\delta_{y'y_{i-1}} \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j).$$

Having obtained all partial results we have the gradient of the log-likelihood with respect to $v_{yy'd}$:

$$\begin{aligned}\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial v_{yy'd}} =& \sum_{i=1}^{|\mathbf{S}|} \mathbf{g}_d(\mathbf{X}, t_i)\delta_{yy_i}\delta_{y'y_{i-1}} \\ &- \frac{1}{Z(\mathbf{X})}\sum_{\mathbf{S}'}\sum_{i=1}^{|\mathbf{S}'|} \mathbf{g}_d(\mathbf{X}, t_i)\delta_{yy_i}\delta_{y'y_{i-1}} \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j).\end{aligned}$$

A.4 Two label dependent bias parameter

For the bias parameter $\mu_{yy'}$ the partial derivative is very similar to the partial derivative of $v_{yy'd}$. The difference is that the dependency on the input vector $\mathbf{g}(\mathbf{X}, t_i)$ is omitted.

$$\frac{\partial \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial \mu_{yy'}} = \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)\delta_{yy_i}\delta_{y'y_{i-1}}$$

Therefore the partial derivative of $\log p(\mathbf{S}|\mathbf{X})$ with respect to $\mu_{yy'}$ is:

$$\begin{aligned}\frac{\partial \log Z(\mathbf{X})}{\partial \mu_{yy'}} &= \sum_{i=1}^{|\mathbf{S}|} \delta_{yy_i}\delta_{y'y_{i-1}} \\ &- \frac{1}{Z(\mathbf{X})} \sum_{\mathbf{S}} \sum_{i=1}^{|\mathbf{S}|} \delta_{yy_i}\delta_{y'y_{i-1}} \prod_{j=1}^{|\mathbf{S}|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j)\end{aligned}$$

A.5 Hidden layer partial derivative

Considering a hidden layer SCRF, the partial derivative of the hidden layer parameters can be determined by applying the backpropagation method [3].

Here we assume a SCRF with a state factor that is equipped with a hidden layer. Let w_{jd} be the hidden layer weight connecting the d-th input with the j-th hidden neuron. Do not confuse this with the input dependent weight w_{yj} from the state factor receiving its input from neuron j.

$$\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{jd}} = \frac{\sum_{i=1}^{|\mathbf{S}|} \log \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{jd}} - \frac{\log Z(\mathbf{X})}{\partial w_{jd}}$$

The derivation follows a similar path as the partial derivative with respect to w_{yd} in Section A.1. Here we present only the important differences in the intermediate steps and the final result.

First we consider the derivative of the state factor with respect to the neuron weight w_{jd}, *i.e.*

$$\frac{\partial \phi(y_i, \mathbf{X}, t_i, u_i)}{\partial w_{jd}} = \frac{\partial \left(\mathbf{w}_y^T \mathbf{z}(\mathbf{X}, t_i, u_i) + \lambda_y\right) \delta_{yy_i}}{\partial w_{jd}} = w_{y_i j}\mathbf{f}_d(\mathbf{X}, t_i, u_i)h'(o_j),$$

where $w_{y_i j}$ is the weight from the state factor, depending on the current label y_i and o_j is the output activation of neuron j. The output activation of neuron j is given as

$$o_j = \mathbf{w}_j^T \mathbf{f}(\mathbf{X}, t_i, u_i) + b_j,$$

where b_j is the bias weight of the neuron. Thus the partial derivative of the left hand side of $\log p(\mathbf{S}|\mathbf{X})$ is:

$$\frac{\sum_{i=1}^{|\mathbf{S}|} \log \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{jd}} = \sum_{i=1}^{|\mathbf{S}|} w_{y_i j}\mathbf{f}_d(\mathbf{X}, t_i, u_i)h'(o_j).$$

Next, we consider the derivative of the potential function w.r.t to the hidden layer parameter:

$$\frac{\partial \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{jd}} = w_{y_i j}\mathbf{f}_d(\mathbf{X}, t_i, u_i)h'(o_j)\Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i).$$

Having all intermediate results we can now derive the partial derivative of the logarithm of the partition function with respect to the hidden layer weight w_{jd} of neuron j:

$$\begin{aligned}\frac{\partial \log Z(\mathbf{X})}{\partial w_{jd}} &= \frac{\partial \log \sum_{\mathbf{S}'} \prod_{i=1}^{|\mathbf{S}'|} \Phi(y_i, y_{i-1}, \mathbf{X}, t_i, u_i)}{\partial w_{jd}} \\ &= \frac{1}{Z(\mathbf{X})} \sum_{\mathbf{S}'} \sum_{i=1}^{|\mathbf{S}'|} w_{y_i j} \mathbf{f}_d(\mathbf{X}, t_i, u_i) h'(o_j) \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j).\end{aligned}$$

The final result of the derivative of the log-likelihood with respect to the weight of a neuron in a hidden layer underneath the state factor is given by

$$\begin{aligned}\frac{\partial \log p(\mathbf{S}|\mathbf{X})}{\partial w_{jd}} &= \sum_{i=1}^{|\mathbf{S}|} w_{y_i j} h'(o_j) \mathbf{f}_d(\mathbf{X}, t_i, u_i) \\ &- \frac{1}{Z(\mathbf{X})} \sum_{\mathbf{S}'} \sum_{i=1}^{|\mathbf{S}'|} w_{y_i j} h'(o_j) \mathbf{f}_d(\mathbf{X}, t_i, u_i) \prod_{j=1}^{|\mathbf{S}'|} \Phi(y_j, y_{j-1}, \mathbf{X}, t_j, u_j).\end{aligned}$$

List of Tables

List of Algorithms

List of Figures

Bibliography

[1] Ossama Abdel-Hamid, Li Deng, Dong Yu, and Hui Jiang. Deep segmental neural networks for speech recognition. In *Proc. INTERSPEECH*, August 2013.

[2] Ossama Abdel-hamid, Abdel rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[4] Lopes Carla and Perdigão Fernando. Phone recognition on the TIMIT database. In *Speech Technologies*. www.intechopen.com, 2011.

[5] DARPA-ISTO. The DARPA TIMIT acoustic-phonetic continuous speech corpus (TIMIT). NIST Speech Disc CD1-1.1, October 1990.

[6] Alex Graves, Santiago Fernández, and Faustino Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *International Conference on Machine Learning (ICML) 2006*, pages 369–376, 2006.

[7] Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP)*, 2013.

[8] A. K. Halberstadt. *Heterogenous Acoustic Measurements and Multiple Classifiers for Speech Recognition*. PhD thesis, Mass. Inst. of Technol., Cambridge, MA, 1998.

[9] Andrew K. Halberstadt and James R. Glass. Heterogeneous acoustic measurements for phonetic classification. In *booktitle*, pages 401–404, In EUROSPEECH 1997.

[10] Yanzhang He and Eric Fosler-Lussier. Efficient segmental conditional random fields for one-pass phone recognition. In *INTERSPEECH*, pages 1898–1901, 2012.

[11] J. Lafferty, McCallum A., and Pereira F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, pages 282–289, 2001.

[12] A. Mohamed, G.E. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):14–22, 2012.

[13] J. Morris and E. Fosler-Lussier. Conditional random fields for integrating local discriminative classifiers. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(3):617–628, 2008.

[14] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Neural Information Processing Systems (NIPS)*, 2001.

[15] Xian Qian, Xiaoqian Jiang, Qi Zhang, Xuanjing Huang, and Lide Wu. Sparse higher order conditional random fields for improved sequence labeling. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 849–856, New York, NY, USA, 2009. ACM.

[16] Abdel rahman Mohamed, George Dahl, and Geoffrey Hinton. Deep belief networks for phone recognition. In *NIPS 22 workshop on deep learning for speech recognition*, 2009.

[17] Martin Ratajczak, Sebastian Tschiatschek, and Franz Pernkopf. Structured regularizer for neural higher-order sequence models. In *European Conference on Machine Learning, ECML*, pages 168–183, 2015.

[18] Sunita Sarawagi and William W. Cohen. Semi-Markov conditional random fields for information extraction. In *Advances in Neural Information Processing Systems 17*, pages 1185–1192, 2004.

[19] Nan Ye, Wee S. Lee, Hai L. Chieu, and Dan Wu. Conditional random fields with high-order features for sequence labeling. In *Advances in Neural Information Processing Systems 22*, pages 2196–2204. Curran Associates, Inc., 2009.

[20] Geoff Zweig. Classification and recognition with direct segment models. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2012.

[21] Geoffrey Zweig and Patrick Nguyen. A segmental CRF approach to large vocabulary continuous speech recognition. In *Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2009.