Universität Rostock

Traditio et Innovatio

Decoding the Output of Neural Networks

A Discriminative Approach

Dissertation

zur

Erlangung des akademischen Grades

doctor rerum naturalium (Dr. rer. nat.)

der Mathematisch-Naturwissenschaftlichen Fakultät

der Universität Rostock

vorgelegt von
Tobias Strauß, geb. am 10.02.1984 in Waren (Müritz)
aus Rostock
Rostock, 16.12.2016

urn:nbn:de:gbv:28-diss2017-0083-3

Identification of letters within a cursive line requires locating the beginning and the end points of individual letter-inscriptions. It is common to think of this as a task to be accomplished before the individual inscriptions can be recognized. But this is paradoxical, since the individual letters must be recognized as such before it can be determined where one inscription leaves off and another begins.

Sayre's paradox

Erstgutachter:	Prof. Dr. Roger Labahn Institut für Mathematik, Universität Rostock
Zweitgutachter:	Prof. Dr. Joan Andreu Sánchez Peiró PRHLT research center, Universitat Politècnica de València
Tag der Verteidigung:	24.05.2017

Contents

1 Introduction

Digital texts have many advantages over physical ones: They are easy to copy, easy to distribute and searchable. But many handwritten documents only exist in form of physical letters, papers or books. There is an incredible amount of such documents that public and private libraries sometimes measure their inventory in bookshelf kilometers. The majority is not yet digitalized. In the recent years, a need is arisen to digitize and to transcribe them, i.e., to convert images of texts into a sequence of machine-encoded characters, to make them accessible to the general public. Since 2004, Google tries to digitize and read at least printed documents which is commonly regarded as an easier task than handwritten text recognition (Vincent [2007]). For handwritten texts this task is called offline handwritten text recognition (HTR). This thesis investigates a subtask of HTR.

In recent years, artificial Neural Networks (NNs) have been evolved to the state-of-the-art solution for a broad class of real world problems under the label Deep Learning: Speech recognition, image and object classification, neural language models and translation systems are only the most prominent examples since they are already integrated in everyday software by high tech companies like Facebook, Google or Apple (LeCun et al. [2015]). The breakthrough of Neural Networks in the field of HTR was a bit less perceived although it was not less spectacular. At ICDAR 2009 competitions on Arabic and French HTR a Neural Network based recognition system outperformed the other submissions by halving the error rate of the next best system (Grosicki and Abed [2009]). This system was proposed in Graves and Schmidhuber [2009] and the approach is entirely discriminative, that means, the character sequences are modeled depending on the observations (which is an image of the writing in case of HTR). In contrast, most of the competing HTR systems were based on Hidden Markov Models (HMMs) – a generative sequence labeling model. That means, HMMs model the sequence of observations depending on the latent character sequence. Since then, many research groups adapted the ideas of Graves and Schmidhuber by combining Hidden Markov Models and such Neural Networks. Although the training, i.e., the parameter adaptation, is still discriminative, the decoding, i.e., the search for the most likely character sequence, is accomplished in a generative manner (Bideault et al. [2015a], Moysset et al. [2014], Bluche et al. [2014]).

It seems to be curious to train a model according to one objective function and decode according to another objective function. HMMs are a very popular decoding approach probably because there are many convenient software tools available and

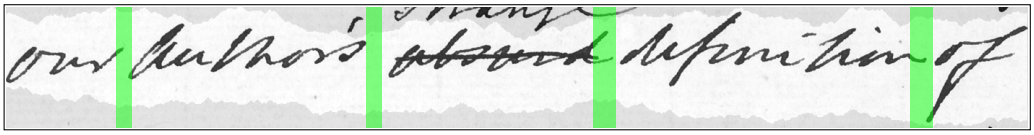
the theory is well understood. Compared to the sophisticated methods for Hidden Markov Models, discriminative decoding as it was proposed in Graves et al. [2006] is still at an early phase of its development. The decoding of HMMs is usually applies Automata to describe the set of feasible character sequences. The rules to describe these feasible sequences can be very complex, especially, if they are derived from natural language. Fortunately, these rules can be split up into different levels of abstraction each of them represented by a Finite State Transducer (FST) which is an extension of Automata. An FST models relations between two sequences over potentially different alphabets. For example, one Transducer relates sequences of HMM states to sequences of characters and another one relates sequences of characters to words. These two Finite State Transducers can be composed. The resulting Finite State Transducer relates sequences of states to words. The set of feasible state sequences defined by such FST contains only those elements which correspond to feasible words (Mohri et al. [2008]). Besides the theoretical advantages, this approach has been transferred to software implementations such that nearly all state-of-the-art HMM software is based on or at least supports Finite State Automata and Finite State Transducers (e.g. Rybach et al. [2009], Povey et al. [2011]).

The considered Neural Networks of this thesis label the input as shown in Figure 1.1: The finite input sequence is an image of a writing (Figure 1.1(a)) and the Neural Network outputs a confidence of each label at each position (Figure 1.1(b)). The rows correspond to a specific character and the columns represent a specific area in the input image. Hence, the column index is called *position* or *time step* in the following. The brightness represents the confidence. The highest activity in this output matrix is in the center, at the very first and at the last row. The center contains the activations of the small letters over the time and the last row represents the space ␣. The first row represents an additional output label $\oslash$ and is active if none of the character labels is active. Loosely spoken it means: At this position there is no character. Thus, the labels provided by the Neural Network are $\mathbb{A}' = \mathbb{A} \cup \{\oslash\}$ where $\mathbb{A}$ is the alphabet of characters potentially displayed in the image.

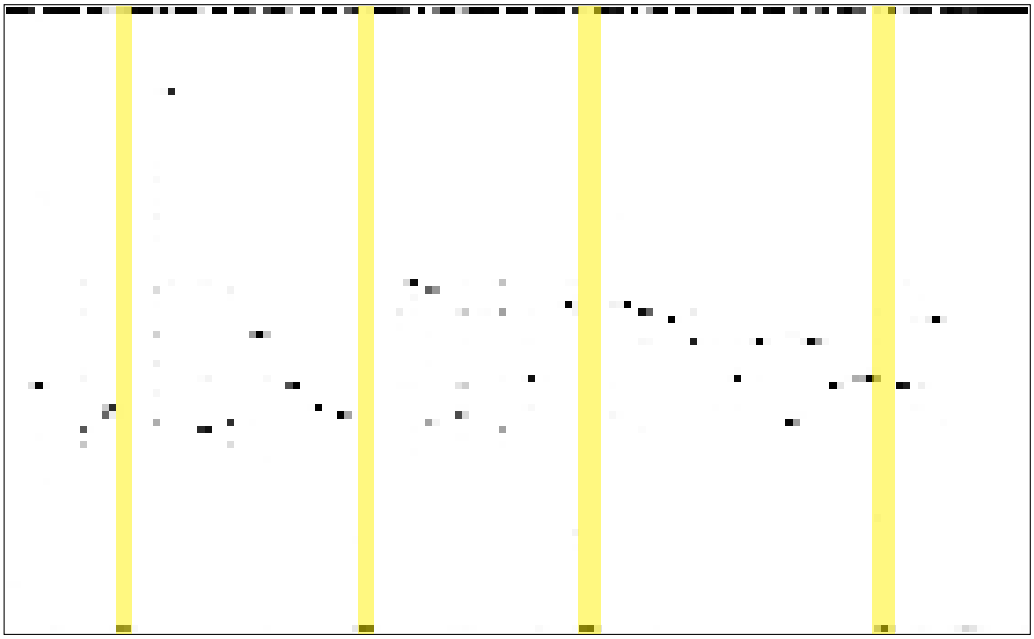
Definition 1.1 (ConfMat). Let $\boldsymbol{Y} \in [0,1]^{T \times |\mathbb{A}'|}$ a matrix with the elements $y_{i,j}$. We call $\boldsymbol{Y}$ Confidence Matrix (ConfMat) iff $\sum_{j=1}^{|\mathbb{A}'|} y_{i,j} = 1$ for each $i \in \{1, \dots, T\}$.

The number T of rows of $\boldsymbol{Y}$ is the number of ConfMat positions and $|\mathbb{A}'|$ is the number of labels (i.e., classes) the NN can distinguish. Figure 1.1(b) represents such a (transposed) ConfMat. In this thesis, we investigate the discriminative decoding of ConfMats for handwritten text recognition as it is suggested in e.g. Graves et al. [2006, 2008], Graves and Jaitly [2014]. These articles describe the feasible label sequences directly by the rules which label is allowed after another. We show that Automata and Finite State Transducers yield a convenient way to describe these label sequences. The decoding process itself can be formulated as an optimization problem on Weighted Automata.

Generally speaking, decoding is the process of converting information from one



(a) Line of handwritten text: “our Author’s absurd definition of”. Positions between two words highlighted in green.



(b) The corresponding output of a Neural Network. Black means high confidence, white means very unlikely. Rows correspond to labels, columns correspond to positions in the image. Active positions of the space (last row) which are highlighted in yellow coincide with the green areas above.

Figure 1.1: Textline and the corresopnding output of the Neural Network. Textline from HTRtS data set (Sánchez et al. [2015]).

representation to another. In this thesis, the information is encoded in the network's output and it is converted to a character sequence over the alphabet $\mathbb{A}$. The feasible subset $\mathcal{X}$ of the set of all possible character sequences $\mathbb{A}^*$ may contain e.g. words of a specific language, numbers from a certain range, syntactically correct sentences, etc.

Definition 1.2 (Decoding). For any $T \in \mathbb{N}$ and any alphabet $\mathbb{A}$, let $\boldsymbol{Y} \in \mathbb{R}^{T \times |\mathbb{A}'|}$ be a ConfMat. Let $\mathcal{X} \subset \mathbb{A}^*$ be a set of feasible character sequences over some alphabet $\mathbb{A}$. Given a confidence function $\mathrm{O} : [0,1]^{T \times |\mathbb{A}'|} \times \mathbb{A}^* \to \mathbb{R}$, the optimization problem

$$\begin{aligned} &\mathrm{O}(\boldsymbol{Y}, \boldsymbol{z}) \to \max \\ \text{s.t. } &\boldsymbol{z} \in \mathcal{X} \end{aligned} \tag{OP}$$

of finding the most confident $\boldsymbol{z} \in \mathcal{X}$ with respect to $\boldsymbol{Y}$ and O is called *decoding*.

The function O will typically be any kind of likelihood of $\boldsymbol{z}$ given $\boldsymbol{Y}$ but we also investigate other metrics.

We investigate elementary decoding strategies with different Automata, for different situations and show how to prune branches of the search space to stay efficient. As a major result of this thesis, we propose a method which decodes the ConfMat of a Neural Network constrained to Regular Languages. This method is convenient for information retrieval tasks such as keyword spotting since a broad class of patterns which form the feasible character sequences $\mathcal{X}$ can easily be described by Regular Expressions. Its ability to align parts of the Regular Expression to parts of the ConfMat $\boldsymbol{Y}$ facilitates the analysis of the result. We show that the proposed methods are capable of solving real world tasks as keyword spotting and full text recognition. Since HTR inherently involves a natural language, the tools from Natural Language Processing potentially improve the recognition accuracy. One such tool is a Language Model. We give a method to integrate Language Models into the decoding which is a generalization of the method proposed in Graves et al. [2008] and is also linked to methods used for decoding of NN-HMM combinations.

In the remainder of this chapter, we introduce a basic notation which will be important for the thesis. The next two chapters introduce (Finite State) Automata, their connection to Regular Expressions, Finite State Transducers, Weighted and Probabilistic Automata and efficient methods for optimization problems on Weighted Automata. Chapter 4 gives a brief introduction to the two most common sequence labeling approaches: Hidden Markov Models and Neural Networks. Both algorithms are strongly related to Language Models which are introduced very briefly. In the same chapter, we also introduce a popular training procedure called Connectionist Temporal Classification (CTC) which is assumed throughout the thesis as training procedure for Neural Networks. Afterwards, we start investigating discriminative approaches for decoding of single words. We investigate slight modifications of the CTC decoding, review a keyword spotting approach and show effective speed-up mechanisms. In Chapter 6, the feasible character sequences are restricted to Regu-

lar Languages. We derive an efficient decoding heuristic which is proven to be exact under mild conditions. We confirm these results experimentally. In Chapter 7, we apply the derived methods to recognition tasks such as keyword spotting and full text recognition. To improve the results, we derive a method to integrate Language Models into the decoding.

Notation

In the following, we introduce some notation to simplify the description.

A finite sequence will be denoted by its elements $o_1, \dots, o_n \in \mathbb{O}$ or equivalently by the sequence vector $\boldsymbol{o}_{1:n} \in \mathbb{O}^n$. To specify the subsequence from s to e, we abbreviate $\boldsymbol{o}_{s:e} = o_s, o_{s+1}, \dots, o_e$. Vectors $\boldsymbol{v} \in \mathbb{R}^n$ are usually written in bold font. A matrix $\boldsymbol{M} \in \mathbb{R}^{m \times n}$ is written bold and in upper-case. The elements are scalars and as such they are written in lower case and plain font, i.e., the element at row i and column j is $m_{i,j}$.

Let $\mathcal{S}$ be a set. We denote $\mathcal{S}^* = \bigcup_{i=0}^{\infty} \mathcal{S}^i$ the set of finite sequences of $\mathcal{S}$. $\mathcal{S}^*$ is called the *Kleene star* or *Kleene closure* on $\mathcal{S}$.

Discrete probability distributions will be important in this thesis. The discrete probability measure P of a discrete probability space $(\Omega, \mathcal{A}, \mathrm{P})$ will be denoted in upper case. We also denote the probability mass function by P. In contrast, we denote probability density functions by p for any continuous probability space.

Let $f : \mathbb{R} \to \mathbb{R}$ be any function. For any $\boldsymbol{x} \in \mathbb{R}^n$, $f(\boldsymbol{x})$ denotes the function which maps $(x_1, \dots, x_n) \mapsto (f(x_1), \dots, f(x_n))$ component-wise.

If $f(x) = c\, g(x)$ for some constant c and two functions f, g, we typically write $f(x) \propto g(x)$.

2 Automata Theory

Finite State Automata are widely used for example in theoretical computer science and text editing software. They can be used to describe a set of character sequences following a specific structure called *Regular Language*. This Regular Language may be described compactly by so called *Regular Expressions*. We will introduce Finite State Automata and consider minimal Automata in the first two sections. Weighted Automata will be introduced in Section 2.3. A simple method to combine Automata with an additional output tape is given in Section 2.4.

2.1 Finite Automata

Regular expressions define specific Regular Languages, i.e., specific subsets of the set of character sequences $\mathbb{A}^*$. There is a correspondence between these languages and Finite State Automata – a model of computation of that language. We use both – the Regular Expression to describe the set of expected sequences and the Automaton to exploit the transition graph. This section gives a brief introduction into the field of Regular Expressions and Finite State Automata which mainly follows Sipser [2006] and Hopcroft et al. [2001].

Definition 2.1 (Regular Expression). For any alphabet $\mathbb{A}$, the empty word (ε) is the character sequence over $\mathbb{A}^*$ of length 0. Then $a \in \mathbb{A}$, ε and $\emptyset$ are *Regular Expressions* which represent the *Regular Languages* $\mathcal{L}(a) = \{a\}$, $\mathcal{L}(\varepsilon) = \{\varepsilon\}$ and $\mathcal{L}(\emptyset) = \emptyset$, respectively. Furthermore, if $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are Regular Expressions, also

- $\boldsymbol{r}_1\boldsymbol{r}_2$ (concatenation),
- $\boldsymbol{r}_1|\boldsymbol{r}_2$ (alternation) and
- $\boldsymbol{r}_1^*$ (Kleene closure)

are Regular Expressions representing the Regular Languages

- $\mathcal{L}(\boldsymbol{r}_1\boldsymbol{r}_2) := \{\boldsymbol{w}\boldsymbol{v} \in \mathbb{A}^* \mid \boldsymbol{w} \in \mathcal{L}(\boldsymbol{r}_1), \boldsymbol{v} \in \mathcal{L}(\boldsymbol{r}_2)\}$,
- $\mathcal{L}(\boldsymbol{r}_1|\boldsymbol{r}_2) := \mathcal{L}(\boldsymbol{r}_1) \cup \mathcal{L}(\boldsymbol{r}_2)$ and
- $\mathcal{L}(\boldsymbol{r}_1^*) := (\mathcal{L}(\boldsymbol{r}_1))^*$.

Thus, Regular Expressions define Regular Languages containing specific sequences of literals from $\mathbb{A}$. Those expressions can be represented by a model of computation:

Definition 2.2 (Finite State Automaton). The *Nondeterministic Finite Automaton (NFA)* N is a 5-tuple $(Q, \mathbb{A}, \delta, q_0, F)$, where Q is the finite set of states, $\mathbb{A}$ is the alphabet, $\delta : Q \times \mathbb{A} \cup \{\varepsilon\} \to \mathcal{P}(Q)$ is the state transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states.

We call N a *Deterministic Finite Automaton (DFA)* iff $\forall q \in Q : \delta(q, \varepsilon) = \{q\}$ and $\forall q \in Q, a \in \mathbb{A} : |\delta(q, a)| \leq 1$.

Thus, the transition function defines not only the successor states but also the feasible next characters of any state. We typically do not explicitly write that $\delta(q, \varepsilon) = \{q\}$ while defining the state transition function.

Definition 2.3 (ε-closure of states). Let $A = (Q, \mathbb{A}, \delta, q_0, F)$ be an NFA. The *ε-closure* $E(q)$ of a state $q \in Q$ is defined as

$$E(q) := \{\hat{q} \in Q \mid \exists p_1, \ldots, p_k \in Q : p_1 = q \wedge p_k = \hat{q} \wedge \forall i \;\; p_{i+1} \in \delta(p_i, \varepsilon)\}.$$

The ε-closure of a set of states $S \subseteq Q$ is simply the union of the individual closures: $E(S) = \bigcup_{q \in S} E(q)$.

Hence, $E(q)$ is the set of states which can be reached from q by reading only ε. Note that $E(q)$ always contains q itself.

Definition 2.4 (Extended transition function δ^*). Let $A = (Q, \mathbb{A}, \delta, q_0, F)$ be an Automaton and $\delta^*(q, \varepsilon) := E(q)$ for any $q \in Q$. Then

$$\delta^*(q, \boldsymbol{w}_{1:n}) := \bigcup_{\bar{q} \in E(\delta^*(q, \boldsymbol{w}_{1:n-1}))} \delta(\bar{q}, w_n)$$

for any $\boldsymbol{w}_{1:n} \in \mathbb{A}^*$ and $\boldsymbol{w}_{1:0} = \varepsilon$.

In other words: δ^* follows the transitions reading $\boldsymbol{w}_{1:n}$ possibly interrupted by ε-transitions.

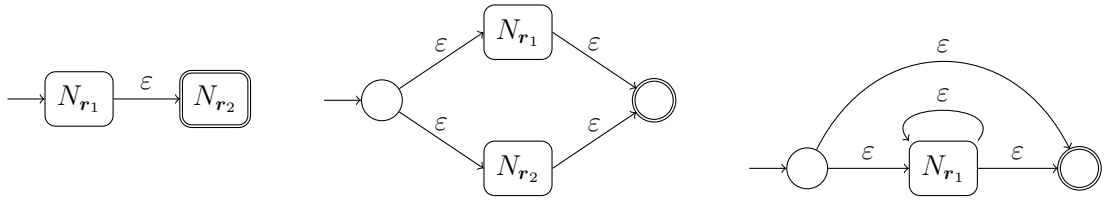
Definition 2.5 (Accepted word). An Automaton A is said to *accept* a word $\boldsymbol{w} \in \mathbb{A}^*$ iff

$$E(\delta^*(q_0, \boldsymbol{w})) \cap F \neq \emptyset.$$

For any state $q \in \delta^*(q_0, \boldsymbol{w})$, $\boldsymbol{w}$ is a prefix which can be completed to an accepted word $\boldsymbol{v}$ if there is a $\bar{\boldsymbol{w}}$ such that $\boldsymbol{v} = \boldsymbol{w}\bar{\boldsymbol{w}}$ and $E(\delta^*(q, \bar{\boldsymbol{w}})) \cap F \neq \emptyset$.

Theorem 2.6 (Thompson [1968]). For any Regular Expression $\boldsymbol{r}$ there is an NFA A accepting the corresponding language $\mathcal{L}(\boldsymbol{r})$ (i.e., A excepts exactly the words from $\mathcal{L}(\boldsymbol{r})$).

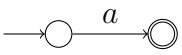
Thompson gave a construction method for this problem, commonly known as *Thompson's Construction Algorithm*. This algorithm exploits a hierarchical composition of



(a) $\boldsymbol{r}_1\boldsymbol{r}_2$ (b) $\boldsymbol{r}_1|\boldsymbol{r}_2$ (c) $\boldsymbol{r}_1{}^*$

Figure 2.1: Schematic representation of atomic NFAs resulting from Thompson's Algorithm. $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are Regular Expressions and $N_{\boldsymbol{r}_1}$ and $N_{\boldsymbol{r}_2}$ are the related NFAs. Other quantifiers or operators can be expressed by those three. Only if $N_{\boldsymbol{r}}$ is marked as initial (final), the initial (final) state is also initial (final) in the composition.

the Regular Expression. It starts by constructing NFAs for the elementary Regular Expressions a, ε and $\emptyset$: For any $a \in \mathbb{A}$, the NFA accepting $\mathcal{L} = \{a\}$ is



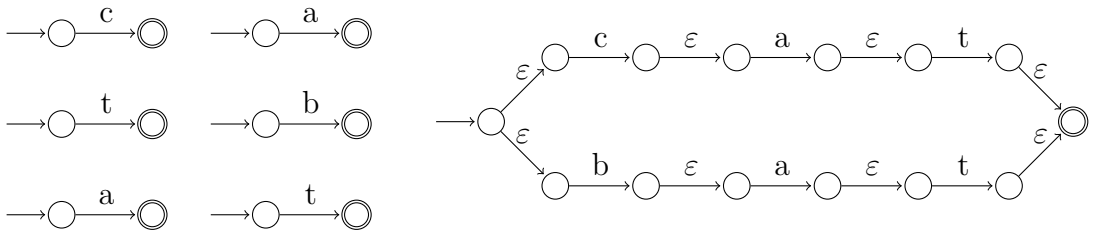
The arc without a start state marks the initial state and the double circles mark the final state. If $\mathcal{L} = \{\varepsilon\}$, replace a by ε. If $\mathcal{L} = \emptyset$, remove the arc. Commonly the arcs in Automata are denoted as transitions. Due to Definition 2.1, any more complex Regular Expression can be expressed by combining "smaller" expressions via concatenation, alternation and Kleene closure. Analogously, Thompson's Construction combines "smaller" NFAs corresponding to "smaller" expressions as depicted in Figure 2.1. Let $N_{\boldsymbol{r}_1}$ and $N_{\boldsymbol{r}_2}$ be two such smaller NFAs. Any transition from $N_{\boldsymbol{r}_1}$ to $N_{\boldsymbol{r}_2}$ connects the final state of $N_{\boldsymbol{r}_1}$ and the initial state of $N_{\boldsymbol{r}_2}$. The algorithm yields only one initial and one final state. Two intermediate steps of this construction are shown in Figure 2.2. A more detailed description may by found in [Hopcroft et al., 2001, Theorem 3.7].

There may be more than one Automaton accepting a Regular Language. For example, an equivalent[1] DFA is obtained by the *Subset Construction Algorithm* (see e.g. [Hopcroft et al., 2001, Theorem 2.11]). Analogously, there may be more than one Regular Expression describing the same language.

Generally, the Subset Construction Algorithm generates a DFA with 2^n states if n is the number of NFA states. Meyer and Fischer [1971] showed that there are languages which also require exactly 2^n DFA states, i.e., the NFA can be exponentially more succinct than the DFA. Therefore, we prefer NFAs over DFAs for infinite languages. For finite languages the minimal Automaton is given in Section 2.2.

Definition 2.7 (ε-closure of Automata). For any NFA $A = (Q, \mathbb{A}, \delta, q_0, F)$, the

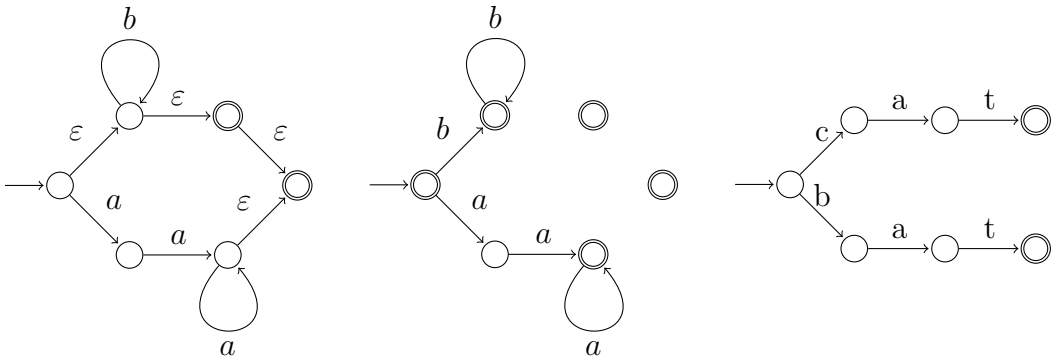
[1] Two Finite State Automata are equivalent if they accept the same language.



(a) Create Automata accepting the individual character.

(b) Combine these Automata using the combinations from Figure 2.1(b) and 2.1(a).

Figure 2.2: Thompson's Construction for the Regular Expression `bat|cat` .



(a) Automaton accepting `aaa*` and `b*`.

(b) ε-closure of the Automaton in Figure 2.3(a).

(c) ε-closure of the Automaton in Figure 2.2(b) without unreachable states.

Figure 2.3: Two examples of an ε-closure. The example in Figure 2.3(b) shows that the closure typically yields some unreachable states which can be removed. Figure 2.3(c) shows the ε-closure of the previous example of Figure 2.2(b).

ε-closure $E(A)$ of A is defined as $E(A) := (Q, \mathbb{A}, \bar{\delta}, q_0, \bar{F})$ where

$$\forall q \in Q, \forall a \in \mathbb{A} : \bar{\delta}(q, a) := \{q' \in Q \mid \exists r \in E(q) : q' = \delta(r, a)\}$$

and $\bar{F} := \{q \in Q \mid E(q) \cap F \neq \emptyset\}$.

The ε-closure of any Automaton A does not contain any transition between two distinct states reading an empty word ε. Especially, $E(A) = A$ for any DFA A. Sometimes the ε-closure yields non-reachable states which can be omitted (see Figure 2.3).

Neither [Sipser, 2006, Theorem 1.19] nor [Hopcroft et al., 2001, Section 2.5.5] introduce the ε-closure individually. Both treat it as a sub-procedure of creating a DFA from an NFA. Further, we also modified the transition slightly to omit redundant states. Thus, we have to prove that any A and $E(A)$ are equivalent:

Lemma 2.8. Any Automaton A is equivalent to $E(A)$, i.e., for any $\boldsymbol{w} \in \mathbb{A}^*$, A accepts $\boldsymbol{w}$ if and only if $E(A)$ accepts $\boldsymbol{w}$.

Proof. By definition, A accepts $\boldsymbol{w}_{1:n}$ iff $E(\delta^*(q_0, \boldsymbol{w}_{1:n})) \cap F \neq \emptyset$. Then, there is a sequence $q_0 = p_0, p_1, \ldots, p_n \in Q$, $\Big(\forall i \leq n, \exists r \in E(p_{i-1}) : p_i \in \delta(r, w_i)\Big)$ and $E(p_n) \cap F \neq \emptyset$. Then and only then it holds that $\Big(\forall i \leq n : p_i \in \bar{\delta}(p_{i-1}, w_i)\Big)$ and $p_n \in \bar{F}$, i.e., $E(A)$ accepts $\boldsymbol{w}$. □

Thus, $E(A)$ results from A by "substituting" all ε-transitions between distinct states by appropriate transitions such that $E(A)$ sometimes is called *Automaton without ε-transitions*. Subsequently, we only consider Automata without ε-transitions. Those Automata simplify several previous definitions since $E(q) = \{q\}$: Let $A = (Q, \mathbb{A}, \delta, q_0, F)$ be an Automaton without ε-transitions. Then for any q, $\delta^*(q, \varepsilon) = \{q\}$ and for $\boldsymbol{w} \in \mathbb{A}^*$

$$\delta^*(q_0, \boldsymbol{w}_{1:n}) = \bigcup_{q \in \delta^*(q_0, \boldsymbol{w}_{1:n-1})} \delta(q, w_n).$$

Further, A accepts $\boldsymbol{w}$ iff $\delta^*(q_0, \boldsymbol{w}) \cap F \neq \emptyset$.

Definition 2.9 (Path). A *path* denotes a sequence of states $p_0, \ldots, p_n \in Q$ such that there is a $\boldsymbol{w} \in \mathbb{A}^n$ with $p_i \in \delta(p_{i-1}, w_i)$ and $p_n \in F$.

2.2 Minimal Automata

2.2.1 Minimal NFAs

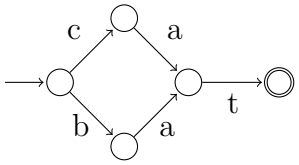
Definition 2.10 (Minimal Automaton). An Automaton $A = (Q, \mathbb{A}, \delta, q_0, F)$ accepting $\mathcal{L}$ is called *minimal* iff there is no Automaton with less states accepting $\mathcal{L}$.

For DFAs, efficient minimization techniques were proposed (see e.g. Hopcroft et al. [2001] Section 4.4.3). Unfortunately, for NFAs the minimization is hard:

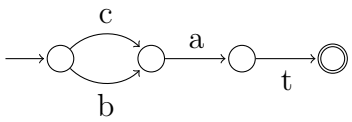
Theorem 2.11 (Jiang and Ravikumar [1993], Theorem 3.2). Converting an NFA into a minimal NFA is PSPACE-complete.

Regular Expressions can be formulated such that Thompson's Construction generates more or less states. Thus, we leave it up to the person which formulates the Regular Expression to find an efficient Automaton.

Example 2.12 (The number of states depends on the formulation of r). We construct an Automaton accepting the language $\mathcal{L} = \{\texttt{cat}, \texttt{bat}\}$. The naïve alternation `cat|bat` of both words leads to an Automaton with 7 states using Thompson's Construction and the ε-closure (see Figure 2.3(c)). We could save 2 states and transitions by alternating only the first letters. The Regular Expression `(c|b)at` will generate two states for `c` and `b` and share the states of `at`. Then, the Automaton will be:



If we aggregate the labels `c` and `b` like `[bc]at`, we could save one additional state and transition since also `c` and `b` are accepted by the same state:



Although a minimization of NFAs is combinatorial hard, a careful formulation of Regular Expressions may save many redundant operations.

2.2.2 Minimal Automata of Finite Languages

The minimum DFA accepting a finite set of words (such as a vocabulary) is well known. It was developed independently by several authors (see Daciuk et al. [2000]). We mainly follow Mihov [1998].

Definition 2.13 (Lexicographical order). Let $\mathbb{A}$ be a totally ordered alphabet. Given two sequences $\boldsymbol{a}_{1:n} \neq \boldsymbol{b}_{1:n} \in \mathbb{A}^*$ of the same length. Then $\boldsymbol{a}_{1:n} < \boldsymbol{b}_{1:n}$ iff there is an index i s.t. $a_i < b_i$ and $a_j = b_j$ for each $0 < j < i$ (otherwise $\boldsymbol{a}_{1:n} > \boldsymbol{b}_{1:n}$). If $\boldsymbol{a}$ and $\boldsymbol{b}$ are of different length, then a special symbol smaller than any other character from $\mathbb{A}$ is appended to the shorter sequence until the both sequences have the same length and can be compared.

Definition 2.14 (Equivalence of states). The *right language* $\overrightarrow{\mathcal{L}}(q)$ of $q \in Q$ is defined by

$$\overrightarrow{\mathcal{L}}(q) := \{\boldsymbol{w} \in \mathbb{A}^* \mid \delta^*(q, \boldsymbol{w}) \cap F \neq \emptyset\}.$$

Two states are *equivalent* (denoted by $\equiv$) if they have the same right language.

DFAs without cycles (except for ε-cycles) are denoted as *Deterministic Acyclic (Finite State) Automaton (DAFSA)* by Daciuk et al. [2000].

Definition 2.15 (Minimal except for $\boldsymbol{w}$). Let $A = (Q, \mathbb{A}, \delta, q_0, F)$ be a *DAFSA* with language $\mathcal{L}$. Then A is said to be *minimal except for* $\boldsymbol{w}$ iff the following conditions hold:

- For each state $q \in Q$, there are $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathbb{A}^*$ such that $q \in \delta^*(q_0, \boldsymbol{w}_1)$ and $\delta^*(q, \boldsymbol{w}_2) \cap F \neq \emptyset$. (Hence, q is reachable from q_0 and there is a final state reachable from q.)
- $\boldsymbol{w}$ is a prefix of the last word of $\mathcal{L}$ in lexicographical order. We denote: $q_0 = t_0, t_1, \ldots, t_k \in Q$ with $k = |\boldsymbol{w}|$, $\forall i > 0 : t_i \in \delta(t_{i-1}, w_i)$ and $T := \{t_0, \ldots, t_k\}$.
- There are no states $q, p \in Q \setminus T$ s.t. $q \neq p$ and $q \equiv p$.
- For any $q \in Q$, any $i \in \{1, \ldots, k\}$ and any $a \in \mathbb{A}$, $\delta(q, a) = t_i$ if and only if $i > 0$ and $q = t_{i-1}$ and $w_i = a$. (T is only reached by $\boldsymbol{w}$.)

In other words: If A is minimal except for $\boldsymbol{w}$, the only states which can potentially be saved are the elements of T (which form path that reads $\boldsymbol{w}$). The "rest" of A cannot further be reduced. Thus, A is minimal iff A is minimal except for ε.

Theorem 2.16 (Mihov [1998]). Let A be a minimal DAFSA except for $\boldsymbol{w}$, let $t_i \in \delta(t_{i-1}, w_i)$ for each $i \in \{1, \ldots, |\boldsymbol{w}|\}$ with $t_0 = q_0$ and $\bar{\boldsymbol{w}} > \boldsymbol{w} \geq \boldsymbol{v}$ in the lexicographical order for any $\boldsymbol{v}$ which is accepted by A. Then Algorithm 1 yields a minimal DAFSA $\bar{A}$ except for $\bar{\boldsymbol{w}}$ which accepts also the words accepted by A.

Proof. Is a direct consequence of Lemma 9, Lemma 10 and Theorem 11 of the cited article. □

For two equivalent states $q, p \in Q$, q is *replaced* by p if any transition ending in q is redirected to p, i.e, for any $r \in Q$ and for any $a \in \mathbb{A}$ with $q \in \delta(r, a)$ remove q from $\delta(r, a)$ and add p. According to Mihov [1998], a redirection of transitions starting in q is not necessary in Algorithm 1 since these transitions will be equal.

For any lexicographically sorted vocabulary $\mathcal{V} \subsetneq \mathbb{A}^*$, the following procedure will generate a minimal DAFSA:

- Start with a minimal DASFA A which accepts only the first word $\boldsymbol{w}_1$ of $\boldsymbol{V}$. Then A is minimal except for $\boldsymbol{w}_1$ with $T = Q$.
- For $i = \{2, \ldots, |\mathcal{V}|\}$, apply Algorithm 1 to A and $\boldsymbol{w}_i \in \mathcal{V}$ in the correct order.
- Finally, apply Algorithm 1 to A and the empty word ε.

In our tests, the number of arcs decreases dramatically compared to alternating the vocabulary words naïvely (i.e., the Automaton which is generated from the Regular Expression $\boldsymbol{z}_1 \,|\, \boldsymbol{z}_2 \,|\, \ldots \,|\, \boldsymbol{z}_n$ where $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ are all the vocabulary words, see Figure 2.3(c)).

Algorithm 1: Automation minimal except for $\bar{\boldsymbol{w}}$

input : $A = (Q, \mathbb{A}, \delta, q_0, F)$ minimal except for $\boldsymbol{w}$, $T = \{t_0, t_1, \ldots, t_k\}$, $\bar{\boldsymbol{w}}$
output: A minimal except for $\bar{\boldsymbol{w}}$, $T = \{\bar{t}_0, \bar{t}_1, \ldots, \bar{t}_k\}$

$c \leftarrow \max\{i \in \mathbb{N} \mid \boldsymbol{w}_{1:i} = \bar{\boldsymbol{w}}_{1:i}\}$; `// index of maximum common prefix`
for $i = |\boldsymbol{w}|$ **to** $c + 1$ **do**
  if $\exists p \in Q : t_i \equiv p$ **then**
    replace t_i by p in A;
    $Q \leftarrow Q \setminus \{t_i\}$;
for $i = 1$ **to** c **do**
  $\bar{t}_i \leftarrow t_i$
for $i = c + 1$ **to** $|\bar{\boldsymbol{w}}|$ **do**
  $Q \leftarrow Q \cup \{\bar{t}_i\}$; `//` $\bar{t}_i$ `is a new state s.t.` $Q \cap \{\bar{t}_i\} = \emptyset$
  $\delta(\bar{t}_{i-1}, \bar{w}_i) \leftarrow \{\bar{t}_i\}$;
$F \leftarrow F \cup \{\bar{t}_{|\bar{\boldsymbol{w}}|}\}$;

2.3 Weighted and Probabilistic Automata

The main ideas follow Mohri et al. [2008] and Dupont et al. [2005] in this section although the notation is adapted to our requirements.

Definition 2.17 (Weighted Automaton). A (time dependent) *Weighted Automaton (WA)* $W = (Q, \mathbb{A}, \lambda, q_0, F)$ is specified by a finite set of states Q, the alphabet $\mathbb{A}$, the initial state $q_0 \in Q$, the time dependent reward function $\lambda : Q \times \mathbb{A} \times Q \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ and $F \subseteq Q$ the set of final states.

The transition from q to q' reading a at time t gets a reward $\lambda(q, a, q', t) \geq 0$. We call $\lambda(q, a, q', t)$ reward instead of weight since we will define a maximization problem on the Weighted Automaton: Find the path with the highest reward from q_0 to any final state. In contrast, the term "weight" indicates a minimization problem. Additionally, the rewards are time dependent. For transitions which are not allowed, the reward is simply 0. The reward of a path is the product over the rewards of all its transitions such that a path containing a 0-reward transition imitatively yields reward 0.

Definition 2.18 (Greatest and Total Reward). Let W be a WA, then the reward function can be extended to $\lambda^+ : Q \times \mathbb{A}^* \times Q \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ for any $0 < n \in$

$\mathbb{N}, \boldsymbol{w} \in \mathbb{A}^n, q, q' \in Q$ and $t \in \mathbb{N}$ by

$$\lambda^+(q, \varepsilon, q', t) := \begin{cases} 1 & \text{if } q = q', \\ 0 & \text{otherwise,} \end{cases}$$

$$\lambda^+(q, \boldsymbol{w}, q', t) := \sum_{\substack{p_0,\dots,p_n \in Q \\ p_0 = q \\ p_n = q'}} \prod_{i=1}^{n} \lambda(p_{i-1}, w_i, p_i, t + i - 1).$$

Analogously, let $\lambda^{\max} : Q \times \mathbb{A}^* \times Q \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ with

$$\lambda^{\max}(q, \varepsilon, q', t) := \begin{cases} 1 & \text{if } q = q', \\ 0 & \text{otherwise,} \end{cases}$$

$$\lambda^{\max}(q, \boldsymbol{w}, q', t) := \max_{\substack{p_0,\dots,p_n \in Q \\ p_0 = q \\ p_n = q'}} \prod_{i=1}^{n} \lambda(p_{i-1}, w_i, p_i, t + i - 1)$$

for any $0 < n \in \mathbb{N}, \boldsymbol{w} \in \mathbb{A}^n, q, q' \in Q$ and $t \in \mathbb{N}$. For a certain $\boldsymbol{w} \in \mathbb{A}^*$ and WA A, the *total reward* $\rho_W^+(\boldsymbol{w})$ and the *greatest reward* $\rho_W^{\max}(\boldsymbol{w})$ of $\boldsymbol{w}$ and WA W are defined as

$$\rho_W^+(\boldsymbol{w}) := \sum_{q' \in F} \lambda^+(q_0, \boldsymbol{w}, q', 1) \quad \text{and} \quad \rho_W^{\max}(\boldsymbol{w}) := \max_{q' \in F} \lambda^{\max}(q_0, \boldsymbol{w}, q', 1).$$

Definition 2.19 (Support). Let $W = (Q, \mathbb{A}, \lambda, q_0, F)$ be a Weighted Automaton. An Automaton $A = (Q, \mathbb{A}, \delta, q_0, F)$ is called *Support of* W if

$$\forall q \in Q \, \forall a \in \mathbb{A} : \delta(q, a) = \{q' \in Q \mid \exists t \in \mathbb{N} : \lambda(q, a, q', t) \neq 0\}.$$

Lemma 2.20. If the Support of $W = (Q, \mathbb{A}, \lambda, q_0, F)$ is a DFA, then $\lambda^+(q, \boldsymbol{w}, q', t) = \lambda^{\max}(q, \boldsymbol{w}, q', t)$ for any $t \in \mathbb{N}, \boldsymbol{w} \in \mathbb{A}^*$ and $q, q' \in Q$.

Proof. Let $A = (Q, \mathbb{A}, \delta, q_0, F)$ be the Support of the WA W. If A is a DFA, there is at most one sequence $p_0, p_1, \dots, p_n$ with $p_i \in \delta(p_{i-1}, w_i)$ for any $\boldsymbol{w} \in \mathbb{A}^n$. Thus, $\lambda^{\max}(p_0, \boldsymbol{w}, p_n, t) = \prod_{i=1}^{n} \lambda^{\max}(p_{i-1}, w_i, p_i, t + i - 1) = \lambda^+(p_0, \boldsymbol{w}, p_n, t)$. If there is no such state sequence, $\lambda^+(p_1, \boldsymbol{w}, p_n, t) = \lambda^{\max}(p_1, \boldsymbol{w}, p_n, t) = 0$ anyway. □

Definition 2.21 (Probabilistic Automaton). A *Probabilistic Automaton (PA)* P is a 5-tuple $(Q, \mathbb{A}, \phi, \iota, \tau)$ where Q is a finite set of states, $\mathbb{A}$ is the alphabet, $\phi : Q \times \mathbb{A} \times Q \to [0, 1]$ is the transition probability function, $\iota : Q \to [0, 1]$ defines the initial probabilities and $\tau : Q \to [0, 1]$ define the final probabilities. Additionally,

for any $q \in Q$,

$$\sum_{q \in Q} \iota(q) = 1$$
$$\tau(q) + \sum_{a \in \mathbb{A}} \sum_{q' \in Q} \phi(q, a, q') = 1$$

and for any reachable q (i.e., $\iota(\hat{q})\phi(\hat{q}, \boldsymbol{w}, q) > 0$ for some $\hat{q}$)

$$\sum_{\boldsymbol{w} \in \mathbb{A}^*} \phi(q, \boldsymbol{w}, q')\tau(q') > 0$$

where $\phi(q, \varepsilon, q') = \delta_{q,q'}$ (Kronecker delta) and $\phi(q, \boldsymbol{w}a, q') = \sum_{\hat{q} \in Q} \phi(q, \boldsymbol{w}, \hat{q})\phi(\hat{q}, a, q')$ for any $a \in \mathbb{A}$ and $\boldsymbol{w} \in \mathbb{A}^*$.

Definition 2.22 (PA without final probabilities). A *Probabilistic Automaton without final probabilities (NFPA)* is a PA where $\tau(q) = 0$ for any $q \in Q$.

Remark 2.23. We additionally assume that any NFPA has only one initial state, i.e., $\iota(q) = 1$ iff $q = q_0$ and 0 otherwise. Such NFPA $P = (Q, \mathbb{A}, \phi, \iota, \tau)$ form a proper subclass of the set of WAs since there is $W = (Q, \mathbb{A}, \lambda, q_0, F)$ where $\iota(q_0) = 1$, $F = Q$ and $\lambda(q, a, q', t) = \phi(q, a, q')$.

Definition 2.24 (Probability). Let $W = (Q, \mathbb{A}, \lambda, q_0, Q)$ be a NFPA. The *total probability* $\overline{\mathrm{P}}_W(\boldsymbol{w})$ of $\boldsymbol{w} \in \mathbb{A}^*$ is defined by

$$\overline{\mathrm{P}}_W(\boldsymbol{w}) := \sum_{q' \in Q} \lambda^+(q_0, \boldsymbol{w}, q', 1) = \rho^+_W(\boldsymbol{w}).$$

The total probability of a set $W \subseteq \mathbb{A}^*$ of words is

$$\overline{\mathrm{P}}_W(W) := \sum_{\boldsymbol{w} \in W} \overline{\mathrm{P}}_W(\boldsymbol{w}).$$

Lemma 2.25 (Dupont et al. [2005]). Let A be a NFPA. For any integer n, we have

$$\overline{\mathrm{P}}_W(\mathbb{A}^n) = 1.$$

In contrast, PA define probability distributions over $\mathbb{A}^*$ such that

$$\mathrm{P}_W(\mathbb{A}^*) := \sum_{\boldsymbol{w} \in \mathbb{A}^*} \sum_{q, q' \in Q} \iota(q)\, \lambda(q, a, q', t)\tau(q') = 1$$

(Dupont et al. [2005]). We mainly consider WA in the following. Probabilistic Automata have strong connections to Hidden Markov Models which will appear in Section 4.2.

2.4 Composition

Definition 2.26 (Finite State Transducers). The *Finite State Transducer (FST)* T is a 6-tuple $(Q, \mathbb{A}, \mathbb{B}, \delta, I, F)$, where Q is the finite set of states, $\mathbb{A}$ is the input alphabet, $\mathbb{B}$ is the output alphabet, $\delta : Q \times \mathbb{A} \to \mathcal{P}(Q \times \mathbb{B})$ is the state transition function, $I \subset Q$ is the set of initial states and $F \subseteq Q$ is the set of final states.

If there is only one initial state $I = \{q_0\}$, FSTs can be seen as extension to Automata where each transition additionally outputs a value from $\mathbb{B}$. Sometimes we regard such an FST as an Automaton by ignoring its output. The other way around way: An Automaton can be regarded as an FST where each transition outputs only the empty word. The reason for introducing yet another kind of Automaton is that two simple FSTs can be combined to form a more complex one in a very natural way. This enables us to decompose a complex Automaton into different layers of abstraction.

Definition 2.27 (Composition of FSTs). Let $T_1 = (Q_1, \mathbb{A}, \mathbb{B}, \delta_1, I_1, F_1)$ and $T_2 = (Q_2, \mathbb{B}, \mathbb{C}, \delta_2, I_2, F_2)$ two FSTs. Then $\mathring{T} = (\mathring{Q}, \mathbb{A}, \mathbb{C}, \mathring{\delta}, \mathring{I}, \mathring{F})$ is the composition of T_1 and T_2 iff

- $\mathring{Q} \subseteq Q_1 \times Q_2$ and
- $\mathring{I} = I_1 \times I_2$ and
- $\mathring{F} = F_1 \times F_2 \cap \mathring{Q}$ and
- for any $q_1, p_1 \in Q_1$, for any $q_2, p_2 \in Q_2$ and for any $(a, b, c) \in \mathbb{A} \times \mathbb{B} \times \mathbb{C}$:

$$(p_1, b) \in \delta_1(q_1, a) \wedge (p_2, c) \in \delta_2(q_2, b) \Rightarrow \Big((p_1, p_2), c\Big) \in \mathring{\delta}\Big((q_1, q_2), a\Big).$$

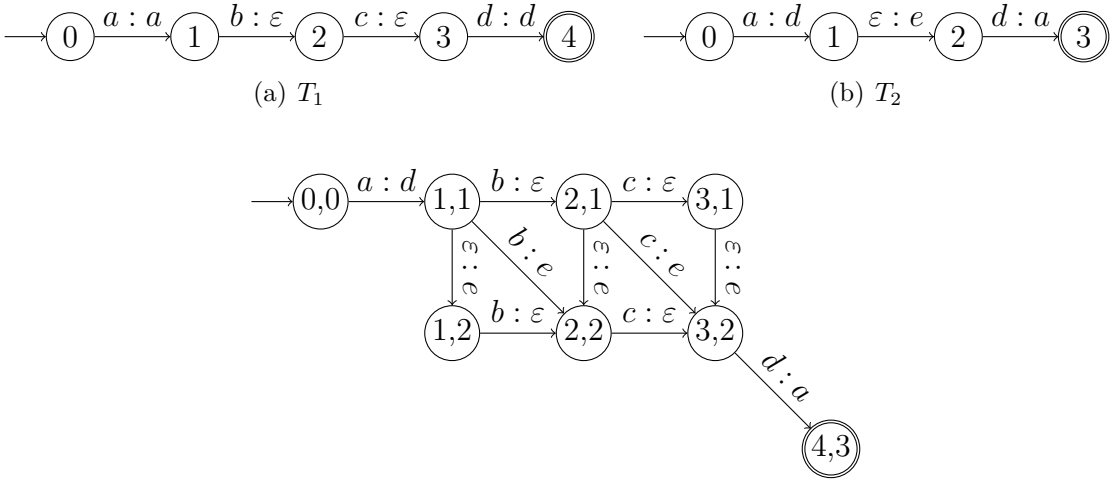
If Automata define feasible subsets of $\mathbb{A}^*$, FSTs define relations between $\mathbb{A}^*$ and $\mathbb{B}^*$ of the accepted words and their outputs, obviously. If the FSTs T_1 and T_2 define the relations $R_1 \subset \mathbb{A}^* \times \mathbb{B}^*$ and $R_2 \subset \mathbb{B}^* \times \mathbb{C}^*$, then the composition defines the relation $R_1 \circ R_2 = \{(\boldsymbol{a}, \boldsymbol{c}) \in \mathbb{A}^* \times \mathbb{C}^* \mid \exists \boldsymbol{b} \in \mathbb{B}^* : (\boldsymbol{a}, \boldsymbol{b}) \in R_1 \wedge (\boldsymbol{b}, \boldsymbol{c}) \in R_2\}$. In other words, the composition of T_1 and T_2 accepts a sequence $\boldsymbol{a} \in \mathbb{A}^*$ and outputs a corresponding sequence $\boldsymbol{c} \in \mathbb{C}^*$ if there is a sequence $\boldsymbol{b} \in \mathbb{B}^*$ which is output of T_1 while accepting $\boldsymbol{a}$ and if T_2 accepts $\boldsymbol{b}$ while outputting $\boldsymbol{c}$. Algorithm 2 provides the pseudo code of the composition which is a simplified version of the composition of Weighted FSTs published in Mohri et al. [2008].

Figure 2.4 shows two finite state transducers T_1, T_2 and their composition $T_1 \circ T_2$. In later chapters, we will also introduce FSTs which occasionally output the empty word ε. Although the higher level FST T_2 may have no explicit ε-transition, it is always possible to stay in a certain state while reading and outputting the empty word, as for example in Figure 2.4: The transition $(1, 1)$ to $(2, 1)$ in Figure 2.4(c) makes use of this property: T_2 stays in state 1 while reading the output ε of the transition from state 1 to state 2 of T_1.

Algorithm 2: Composition of two FST

input : $T_1 = (Q_1, \mathbb{A}, \mathbb{B}, \delta_1, I_1, F_1), T_2 = (Q_2, \mathbb{B}, \mathbb{C}, \delta_2, I_2, F_2)$
output: $T_1 \circ T_2 = (\mathring{Q}, \mathbb{A}, \mathbb{C}, \mathring{\delta}, I_1 \times I_2, \mathring{F})$

$\mathring{Q} \leftarrow I_1 \times I_2$;
$S \leftarrow I_1 \times I_2$;
while $S \neq \emptyset$ **do**
  $(q_1, q_2) \leftarrow$ get and remove first element of S;
  if $q_1 \in F_1$ *and* $q_2 \in F_2$ **then**
    $\mathring{F} \leftarrow \mathring{F} \cup \{(q_1, q_2)\}$;
  for $(a, b, c) \in A \cup \{\varepsilon\} \times B \cup \{\varepsilon\} \times C \cup \{\varepsilon\}$ **do**
    for $(\hat{q}_1, \hat{q}_2) \in Q_1 \times Q_2$ *with* $(\hat{q}_1, b) \in \delta_1(q_1, a)$ *and* $(\hat{q}_2, c) \in \delta_2(q_2, b)$ **do**
      if $(\hat{q}_1, \hat{q}_2) \notin \mathring{Q}$ **then**
        $\mathring{Q} \leftarrow \mathring{Q} \cup \{(\hat{q}_1, \hat{q}_2)\}$;
        $S \leftarrow S \cup \{(\hat{q}_1, \hat{q}_2)\}$;
      $\mathring{\delta}((q_1, q_2), a) \leftarrow \mathring{\delta}((q_1, q_2), a) \cup \{((\hat{q}_1, \hat{q}_2), c)\}$;



(a) T_1 (b) T_2

(c) $T_1 \circ T_2$

Figure 2.4: Two finite state transducers T_1, T_2 and their composition $T_1 \circ T_2$ (example from Mohri et al. [2008]). The colon separates the input (left) and the output (right) of a transition.

2.5 Conclusion

This section introduces Automata and related concepts which are necessary for this thesis. Automata accept a subset of $\mathbb{A}^*$, the so-called Regular Language. This language can be described by Regular Expressions, conveniently. We briefly show how to create a corresponding NFA and how to "delete" uninteresting ε-transitions. We also introduce Weighed Automata which additionally put weights or rewards on each transition. Later, we will search for paths of specific WA with maximum reward. We also mention that it is computationally hard to calculate the minimum NFA with respect to any Regular Expression. For the practically interesting case of finite sets of words, the minimum Automaton is known. Below, it will be helpful to divide different levels of abstraction, each of them represented by an Automaton. A composition of Automata is not defined but a composition of Finite State Transducers which additionally output a letter at each transition. We also provide the algorithm to compose FSTs. The composed FST accepts a sequence if and only if it is feasible according to the lower level FST and the output of the lower level FST is accepted by the higher level FST.

3 Dynamic Programming

Dynamic Programming is a method to reduce a complex calculation to a sequence of simpler calculations. Typical applications are special kinds of optimization problems which can be split into different stages and the value of each stage depends only on the previous stage. An example for such optimization problem is given in Section 3.1.

Many authors (e.g. Russell et al. [2003]) introduce Dynamic Programming while solving the objective function of a *Markov decision process*. The formulation of our optimization problems as Markov Decision Processes appears to be awkward. Since we apply Dynamic Programming to maximize the reward of Weighted Automata, we introduce Dynamic Programming only for this case although the basic idea is much more general. Additionally, we provide a classical application of Dynamic Programming in Section 3.3.

3.1 Maximum Reward

First, consider the maximization problem

$$\begin{aligned} \rho_W^{\max}(\boldsymbol{w}) &\to \max \\ \text{s.t. } \boldsymbol{w} &\in \mathcal{X}_1 \times \cdots \times \mathcal{X}_T \subseteq \mathbb{A}^T \end{aligned} \tag{3.1.1}$$

for any WA $W = (Q, \mathbb{A}, \lambda, q_0, F)$. If $\mathcal{X}_i := \{a_i\}$, (3.1.1) maximizes the reward of the sequence $\boldsymbol{a}_{1:T} \in \mathbb{A}^T$ which is $\rho_W^{\max}(\boldsymbol{a}_{1:T})$. If $\mathcal{X}_i := \mathbb{A}$, (3.1.1) yields the maximum reward of any sequence. Both cases will be important in later chapters.

A necessary condition is that the objective function is decomposable into a sequence of smaller objectives such that each of the smaller objectives only depends on the directly preceding objective values but not on other past or future values. This property allows to solve the objective function stagewise as the following lemma shows:

Lemma 3.1. Consider the optimization problem (3.1.1). For any $q' \in Q$, let

$$\alpha_0(q') := \begin{cases} 1 & \text{if } q' = q_0 \\ 0 & \text{else} \end{cases}$$

and for $t \geq 1$

$$\alpha_t(q') := \max_{q \in Q, a \in \mathcal{X}_t} \alpha_{t-1}(q)\, \lambda(q, a, q', t).$$

Then for any $1 \leq t \leq T$

$$\alpha_t(q') = \max_{\boldsymbol{w} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_t} \lambda^{\max}(q_0, \boldsymbol{w}, q', 1). \tag{3.1.2}$$

Especially,

$$\max_{\boldsymbol{w} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_T} \rho_W^{\max}(\boldsymbol{w}) = \max_{q \in F} \alpha_T(q).$$

Proof. Induction over t. For $t = 1$:

$$\max_{a \in \mathcal{X}_1} \lambda^{\max}(q_0, a, q', 1) = \max_{a \in \mathcal{X}_1} \max_{q \in Q} \alpha_0(q)\, \lambda(q, a, q', 1) = \alpha_1(q').$$

For $t > 1$: Let Eq. (3.1.2) be true for $t - 1$. Then

$$\begin{aligned}
&\max_{\boldsymbol{w} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_t} \lambda^{\max}(q_0, \boldsymbol{w}, q', 1) \\
=&\max_{\boldsymbol{w} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_t} \max_{\substack{p_0, \ldots, q_t \in Q \\ p_0 = q_0 \\ p_t = q'}} \prod_{i=1}^{t} \lambda(p_{i-1}, w_i, p_i, i) \\
=&\max_{a \in \mathcal{X}_t} \max_{q \in Q} \underbrace{\max_{\boldsymbol{w}' \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_{t-1}} \max_{\substack{p_0, \ldots, p_{t-1} \in Q \\ p_0 = q_0 \\ p_{t-1} = q}} \prod_{i=1}^{t-1} \lambda(p_{i-1}, w_i, p_i, i)}_{= \alpha_{t-1}(q)} \; \lambda(q, a, q', t) \\
=&\alpha_t(q')
\end{aligned}$$

□

Obviously, each $\alpha_t(q)$ is an optimal solution to the subproblem of finding the maximum path from q_0 to q reading any $w_1, \ldots, w_t \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_t$. This is known as *Bellman's Principle of Optimality* (Bellman and Dreyfus [1962]):

> *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

Translated to our situation: Independent of the choice of the t-th state, the first $t - 1$ states must be chosen optimally.

Algorithm 3 calculates the maximum reward for any Weighted Automaton. The complexity of the algorithm is in $\mathcal{O}(|Q|^2\, |\mathbb{A}|\, T)$ if $\mathcal{X}_i = \mathbb{A}$ for each i. For *Sparse Automata* (i.e., the number of transitions with $\lambda(q, a, q', t) > 0$ is bounded by a

Algorithm 3: Dynamic Programming for maximum reward.

input : WA $W = (Q, \mathbb{A}, \lambda, q_0, F)$
output: $\max_{q \in F} \alpha_T(q)$

$\alpha_0(q_0) \leftarrow 1$;
for $q \in Q \setminus \{q_0\}$ **do**
  $\alpha_0(q) \leftarrow 0$
for $t \leftarrow 1$ **to** T **do**
  for $q' \in Q$ **do**
    $\alpha_t(q') \leftarrow \max_{q \in Q} \max_{a \in \mathcal{X}_t} \lambda(q, a, q', t) \alpha_{t-1}(q)$

constant $c \ll |\mathbb{A}||Q|$ for any $q \in Q$) the algorithm might be optimized by omitting 0-reward transitions in advance. This leads to a complexity in $\mathcal{O}(|Q|T)$.

3.2 Total Reward

Dynamic Programming is sometimes also denoted as Dynamic Optimization. However, the basic idea is more general and can be applied to more than optimization problems. For example, Dynamic Programming provides a very efficient way to calculate the sum of the total rewards

$$\sum_{\boldsymbol{w} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_T} \rho_W^+(\boldsymbol{w})$$

given the WA $W = (Q, \mathbb{A}, \lambda, q_0, F)$.

Lemma 3.2. Let $A = (Q, \mathbb{A}, \lambda, q_0, F)$ be a WA. For any $q, q' \in Q$, let

$$\alpha_0^+(q) := \begin{cases} 1 & \text{if } q = q_0, \\ 0 & \text{else,} \end{cases} \qquad \beta_T^+(q') := \begin{cases} 1 & \text{if } q' \in F, \\ 0 & \text{else ,} \end{cases}$$

and for $t \geq 1$, $t' \leq T - 1$

$$\alpha_t^+(q') := \sum_{\bar{q} \in Q, a \in \mathcal{X}_t} \alpha_{t-1}^+(\bar{q})\, \lambda(\bar{q}, a, q', t) \;, \quad \beta_{t'}^+(q) := \sum_{\bar{q} \in Q, a \in \mathcal{X}_{t'+1}} \beta_{t'+1}^+(\bar{q})\, \lambda(q, a, \bar{q}, t' + 1).$$

Then for any $1 \leq t \leq T$

$$\alpha_t^+(q') = \sum_{\boldsymbol{w} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_t} \lambda^+(q_0, \boldsymbol{w}, q', 1) \tag{3.2.1}$$

and for $0 \leq t' \leq T-1$

$$\beta_{t'}^{+}(q) = \sum_{\boldsymbol{w} \in \mathcal{X}_{t'+1} \times \cdots \times \mathcal{X}_T} \sum_{q' \in F} \lambda^{+}(q, \boldsymbol{w}, q', t'+1).$$

Especially,

$$\sum_{\boldsymbol{w} \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_T} \rho_W^{+}(\boldsymbol{w}) = \sum_{q \in F} \alpha_T^{+}(q) = \beta_0^{+}(q_0).$$

Proof. The proof is analogous to the proof of Lemma 3.1 if we substitute max by the sum. □

Theorem 3.2 states that α^{+} and β^{+} calculate the sum of the total rewards of any feasible character sequence in opposite directions. Algorithm 4 calculates the total reward for any Weighted Automaton backwards starting from T to 0. An equivalent forward calculation can be derived from Algorithm 3 essentially by substituting max by the sum. Due to Eq. (3.2.1), $\alpha_t^{+}(q)$ is the sum of the rewards up to t of any feasible sequence $\boldsymbol{w}$ which is in state q at time t. Thus, $\alpha_t^{+}(q)$ can be interpreted as the reward of any feasible prefix from $\mathcal{X}_1 \times \cdots \times \mathcal{X}_t$ through q at time t. Analogously, $\beta_t^{+}(q)$ represents the reward of any feasible suffix through q at time t.

Algorithm 4: Dynamic programming for total reward.

input : WA $W = (Q, \mathbb{A}, \lambda, q_0, F)$
output: $\beta_0^{+}(q)$

for $q \in F$ **do**
  $\beta_T^{+}(q) \leftarrow 1$
for $q \notin F$ **do**
  $\beta_T^{+}(q) \leftarrow 0$
for $t \leftarrow T-1$ **to** 0 **do**
  for $q \in Q$ **do**
    $\beta_t^{+}(q) \leftarrow \sum_{q' \in Q} \sum_{a \in \mathcal{X}_{t+1}} \beta_{t+1}^{+}(q')\, \lambda(q, a, q', t+1)$

3.3 Distance of Character Sequences

Definition 3.3 (Levenshtein Distance). For any $n, m \in \mathbb{N}$, let $\boldsymbol{a}_{1:n}, \boldsymbol{b}_{1:m}$ be two sequences from the alphabet $\mathbb{A}$. Let $\mathrm{d} : \mathbb{A}^* \times \mathbb{A}^* \to \mathbb{N}$ be defined by

$$\mathrm{d}(\boldsymbol{a}_{1:i}, \boldsymbol{b}_{1:j}) := \begin{cases} i & \text{if } j = 0, \\ j & \text{if } i = 0, \\ \min\{\, \mathrm{d}(\boldsymbol{a}_{1:i}, \boldsymbol{b}_{1:j-1}) + 1, & \\ \qquad \mathrm{d}(\boldsymbol{a}_{1:i-1}, \boldsymbol{b}_{1:j}) + 1, & \text{otherwise,} \\ \qquad \mathrm{d}(\boldsymbol{a}_{1:i-1}, \boldsymbol{b}_{1:j-1}) + 1 - \delta_{a_i, b_j} \} & \end{cases}$$

for $i \in \{0, 1, \ldots, n\}$ and $j \in \{0, 1, \ldots, m\}$ where δ_{a_i, b_j} denotes the Kronecker-delta. The *Levenshtein Distance* of $\boldsymbol{a}_{1:n}, \boldsymbol{b}_{1:m}$ is defined as $\mathrm{d}(\boldsymbol{a}_{1:n}, \boldsymbol{b}_{1:m})$.

The Levenshtein Distance yields the minimal number of the operations: substitution, insertion and deletion, needed to convert one sequence into another where the *edit-operations* deletion, insertion and substitution are defined in a natural way, e.g. for any sequence $\boldsymbol{z} \in \mathbb{A}^*$, any $x \in \mathbb{A}$ and any position $i \leq n$ the substitution is defined as

$$\mathrm{subs}_i(\boldsymbol{z}, x) := z_1, \ldots, z_{i-1}, x, z_{i+1}, \ldots, z_n.$$

The Levenshtein Distance is widely used in HTR since it is a standard performance measure of a recognition:

Definition 3.4 (Character error rate). Let $\boldsymbol{t}$ be the true character sequence (typically called *ground truth*) and let $\boldsymbol{z}^*$ be the result of the decoding process defined by (OP). Let $\mathrm{d} : \mathbb{A}^* \times \mathbb{A}^* \to \mathbb{N}$ be the Levenshtein Distance on sequences of characters. Then the *character error rate (CER)* of $\boldsymbol{z}^*$ is defined by

$$\mathrm{CER}(\boldsymbol{z}^*, \boldsymbol{t}) := \frac{\mathrm{d}(\boldsymbol{z}^*, \boldsymbol{t})}{|\boldsymbol{t}|}.$$

The CER over a whole data set with ground truth $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n$ and the corresponding recognition results $\boldsymbol{z}_1^*, \ldots, \boldsymbol{z}_n^*$ is defined by the average

$$\mathrm{CER}\left((\boldsymbol{z}_1^*, \boldsymbol{t}_1), \ldots, (\boldsymbol{z}_n^*, \boldsymbol{t}_n)\right) := \frac{\sum_{i=1}^n \mathrm{d}(\boldsymbol{z}_i^*, \boldsymbol{t}_i)}{\sum_{i=1}^n |\boldsymbol{t}_i|}.$$

If not sequences of characters but sequences of words are compared, we obtain the word error rate:

Definition 3.5 (Word error rate). Let $\mathcal{V} \subset \mathbb{A}^*$ be the vocabulary of feasible words and $\Theta : \mathbb{A}^* \to \mathcal{V}^*$ the function which decomposes a character sequence in a sequence of words (typically called *tokenizer*). Let $\mathrm{d} : \mathcal{V}^* \times \mathcal{V}^* \to \mathbb{N}$ be the

Levenshtein Distance on sequences of words[1]. Then, the *word error rate (WER)* of $\boldsymbol{z}^*$ is defined by

$$\mathrm{WER}_\Theta(\boldsymbol{z}^*, \boldsymbol{t}) := \frac{\mathrm{d}(\Theta(\boldsymbol{z}^*), \Theta(\boldsymbol{t}))}{|\Theta(\boldsymbol{t})|}.$$

The WER over a whole data set with ground truth set $\boldsymbol{t}_1, \ldots, \boldsymbol{t}_n$ and the corresponding recognition results $\boldsymbol{z}_1^*, \ldots, \boldsymbol{z}_n^*$ is defined by the average

$$\mathrm{WER}_\Theta\left((\boldsymbol{z}_1^*, \boldsymbol{t}_1), \ldots, (\boldsymbol{z}_n^*, \boldsymbol{t}_n)\right) := \frac{\sum_{i=1}^n \mathrm{d}(\Theta(\boldsymbol{z}_i^*), \Theta(\boldsymbol{t}_i))}{\sum_{i=1}^n |\Theta(\boldsymbol{t}_i|)}.$$

3.4 Conclusion

This Chapter provides the basic calculation techniques of this thesis. Algorithms 3 and 4 immediately yield the maximum and total reward for any Weighted Automaton. Both algorithms will be fundamental for the decoding process. Only the Weighted Automata needs to be specified.

Furthermore, Dynamic Programming also yields the standard performance measure for handwritten text recognition, WER and CER.

[1] That means, the alphabet from Definition 3.3 is the set of feasible words $\mathcal{V}$ in this case.

4 Sequence Labeling Approaches

Given a sequence of observations, a sequence labeling algorithm assigns a label to each of the observations. A lot of real world tasks are sequence labeling tasks such as speech recognition or part-of-speech tagging where the words of a given sentence have to be classified. Also the main application of this thesis, the handwritten text recognition, is a sequence labeling task. Several techniques were proposed to solve such kind of problems. In this chapter, we introduce two of the most popular techniques: Hidden Markov Models and Neural Networks. Since practically all recognition systems for handwritten text recognition integrate a Language Model (LM), we start with a brief introduction of LMs. Afterwards, we introduce Hidden Markov Models and Neural Networks. In Section 4.4, a special approach of training Neural Networks for sequence labeling tasks is introduced. In contrast to the standard literature, we reformulate the known results in the notion of Weighted Automata.

4.1 Language Models

This section introduces the most popular Language Model estimation technique: n-grams.

Definition 4.1 (Language Model). Let $\mathcal{V} \subseteq \mathbb{A}^*$ define the set of feasible words (vocabulary). A *Language Model (LM)* assigns each sequence of words $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m \in \mathcal{V}$ a probability $\mathrm{P}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m) \in [0, 1]$.

Note that we did not define the term "word" precisely since there is an ongoing discussion which of the various definitions is correct in which situation (Manning and Schütze [1999]). In this thesis, we use two meanings of the term word: Unless specified otherwise, a *word* is an arbitrary sequence of characters over an alphabet $\mathbb{A}$. The second meaning of the term refers to the intuitive definition: A word is contained in an official dictionary such as Stevenson [2010]. In ambiguous situations, we will refer to these words as *vocables*.

Due to the chain rule of probability, $\mathrm{P}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m)$ can be expanded by

$$\mathrm{P}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m) = \mathrm{P}(\boldsymbol{z}_1) \prod_{i=2}^{m} \mathrm{P}(\boldsymbol{z}_i \mid \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{i-1}). \tag{4.1.1}$$

Definition 4.2 (*n*-gram). An *n-gram* over a certain set $\mathbb{D} \subset \mathcal{P}(\mathcal{V}^*)$ of word sequences is a Language Model which assumes the *Markov property of n-th order*, i.e., for $i \geq n$

$$\mathrm{P}(\boldsymbol{z}_i \mid \boldsymbol{z}_1, \ldots, \boldsymbol{z}_{i-1}) := \mathrm{P}(\boldsymbol{z}_i \mid \boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1})$$

The precise value of $\mathrm{P}(\boldsymbol{z}_i \mid \boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1})$ is calculated using the frequency of $\boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1}$ followed by $\boldsymbol{z}_i$. Let $c(\boldsymbol{z}_{k:m})$ be the number of appearances of $\boldsymbol{z}_{k:m} = z_k, \ldots, z_m$ as subsequences of any character sequence from $\mathbb{D}$. Then

$$\mathrm{P}(\boldsymbol{z}_i \mid \boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1}) := \begin{cases} \frac{c(\boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_i)}{c(\boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1})} & \text{if } c(\boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1}) \neq 0, \\ 0 & \text{else.} \end{cases}$$

For $i < n$, one typically uses i-grams in Eq. (4.1.1). However, one substitutes Eq. (4.1.1) for practical applications by

$$\mathrm{P}(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m) \approx \prod_{i=1}^{m+1} \mathrm{P}(\boldsymbol{z}_i \mid \boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1})$$

whereas $\boldsymbol{z}_j =$ `<BoS>` for $j < 1$ and $\boldsymbol{z}_{m+1} =$ `<EoS>` denote the Begin-of-Sentence token and the End-of-Sentence token, respectively. The advantage of the above approximation is that it extends the n-gram to model a probability distribution on $\mathcal{V}^*$ instead of $\mathcal{V}^n$ (Vidal et al. [2005]). Typically, this modified n-gram model is denoted as *extended n-gram model*. Equivalently, one could define LMs and n-grams on character sequences instead of word sequences.

The precise n-gram probability strongly depends on the training corpus $\mathbb{D}$ which requires a careful selection of $\mathbb{D}$. Moreover, many meaningful sequences of length n will practically not appear in the training corpus and thus get a zero probability. To circumvent this problem several *smoothing* techniques where proposed to also assign an (at least small) probability to unseen sequences. We refer to Chen and Goodman [1996] for a fundamental overview of classical smoothing techniques.

The most common Language Models are *n-grams*. Recently, Neural Networks (which will be introduced in Section 4.3) seem to outperform the classical n-grams (De Mulder et al. [2015]). Neural Networks are known to do a better smoothing than n-grams (Bengio et al. [2003]). However, the training is very time expensive. Fast stable training methods for a high number of output neurons is an active research area and evolved over the last few years (see Mnih and Teh [2012], Mnih and Kavukcuoglu [2013], Vincent et al. [2015]).

4.2 Hidden Markov Models

Hidden Markov Models (HMM) are the classical, state-of-the-art segmentation-free sequence labeling approach. In the following, we often compare our methods to standard HMM methods and thus we will give a brief introduction into Hidden Markov Models.

4.2.1 Definition

HMMs assume two discrete-time stochastic processes S_t and X_t. The first latent process S_t satisfies the Markov property of first order. Thus, the transition probability at time t only depends on S_t and S_{t-1}. The observable process X_t, basically the inputs to the model, is not directly modeled but depends on the latent process. HMMs are classified based on these observations: If the set of observations is finite, we call the HMM *discrete*. Otherwise, the HMM is *continuous*. We define discrete HMMs only. The extension to continuous HMMs is very natural.

Definition 4.3 (Hidden Markov Model). The (discrete) *Hidden Markov Model (HMM)* $\mathcal{H} = (S, \mathbb{X}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{p}_0)$ is a 5-tuple with the components

- $S = \{s_1, \ldots, s_n\}$ the set of hidden states,
- $\mathbb{X}$ the finite observation space,
- $a_{i,j} := \mathrm{P}(s_j \mid s_i)$ the probabilities of the transition from state s_i to s_j,
- $b_i(x) = \mathrm{P}(x \mid s_i)$ the emission probability of $x \in \mathbb{X}$ in state s_i and
- $\boldsymbol{p}_0 : S \to [0, 1]$ the initial distribution of the states

with the additional restrictions:

$$\begin{aligned} \forall s_i \in S : \sum_{s_j \in S} a_{i,j} = 1, \qquad & \forall s_i, s_j \in S : a_{i,j} \geq 0, \\ \forall s_i \in S : \sum_{x \in \mathbb{X}} b_i(x) = 1, \qquad & \forall s_j \in S, \forall x \in \mathbb{X} : b_j(x) \geq 0, \\ \sum_{s \in S} p_0(s) = 1, \qquad & \forall s \in S : p_0(s) \geq 0. \end{aligned}$$

The emission probabilities $b_i(x) = \mathrm{P}(x \mid s_i)$ model the observables given the hidden states. This is called a generative model in contrast to discriminative models which model the target variables given the observables ($\mathrm{P}(s_i \mid x)$). Since $\mathbb{X}$ and S are finite in the discrete case, these probabilities can be represented as a matrix $\boldsymbol{B}$ with entries $b_i(x)$. For continuous HMMs, $b_i(x)$ are functions $\mathbb{X} \times S \to [0, 1]$ which cannot be represented by a matrix anymore. At least the decoding presented in the following holds for the continuous HMMs in this general notation.

4.2.2 Decoding

Due to the dependence of the observables on the hidden states, we can draw some inference on the current state. The probability of the sequence of states $s_{i_1}, \ldots, s_{i_T} \in S$ given the sequence of observables $\boldsymbol{x}_{1:T}$ is:

$$\begin{aligned}
\mathrm{P}(s_{i_1}, \ldots, s_{i_T} \mid \boldsymbol{x}_{1:T}) &= \frac{\mathrm{P}(\boldsymbol{x}_{1:T} \mid s_{i_1}, \ldots, s_{i_T})\,\mathrm{P}(s_{i_1}, \ldots, s_{i_T})}{\mathrm{P}(\boldsymbol{x}_{1:T})} \\
&\propto \mathrm{P}(\boldsymbol{x}_{1:T} \mid s_{i_1}, \ldots, s_{i_T})\,\mathrm{P}(s_{i_1}, \ldots, s_{i_T}) \\
&= \mathrm{P}(\boldsymbol{x}_{1:T}, s_{i_1}, \ldots, s_{i_T}) \\
&= p_0(s_{i_1})\,\mathrm{P}(x_1 \mid s_{i_1}) \prod_{t=2}^{T} \mathrm{P}(s_{i_t} \mid s_{i_{t-1}})\,\mathrm{P}(x_t \mid s_{i_t}).
\end{aligned}$$

The denominator $\mathrm{P}(\boldsymbol{x}_{1:T})$ is just a scaling factor and constant for any choice of $s_{i_1}, \ldots, s_{i_T}$. Thus, the objective function to find the most likely state sequence is

$$\mathrm{O}(\boldsymbol{x}_{1:T}) = \max_{s_{i_1}, \ldots, s_{i_T}} \mathrm{P}(\boldsymbol{x}_{1:T}, s_{i_1}, \ldots, s_{i_T}) = p_0(s_{i_1}) b_{i_1}(x_1) \prod_{t=2}^{T} a_{i_{t-1}, i_t} b_{i_t}(x_t). \tag{4.2.1}$$

The optimal solution can be found by Dynamic Programming. The following theorem states how to obtain an equivalent Probabilistic Automaton.

Theorem 4.4. Let $\mathcal{H} = (S, \mathbb{X}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{p}_0)$ be an arbitrary discrete HMM and let $\boldsymbol{x}_{1:T} \in \mathbb{X}^T$ be the observation sequence. The NFPA $W = (S \cup \{s_0\}, \mathbb{X}, \lambda, s_0, S)$ with

$$\lambda(s_0, x_1, s_j, 1) = b_j(x_1) p_0(s_j)$$

and for $i > 1$

$$\lambda(s_i, x_t, s_j, t) = b_j(x_t) a_{i,j}$$

for any s_i, s_j and x_t is equivalent to $\mathcal{H}$ in the sense that $\mathrm{P}(\boldsymbol{x}_{1:t}, s_{i_1}, \ldots, s_{i_t}) = \prod_{j=1}^{t} \lambda(s_{i_{j-1}}, x_j, s_{i_j}, j)$ for any sequence $s_{i_1}, \ldots, s_{i_t} \in S$, $s_{i_0} = s_0$ and $\boldsymbol{x}_{1:t} \in \mathbb{X}^t$.

Proof. The claim follows directly from the definitions

$$\mathrm{P}(\boldsymbol{x}_{1:t}, s_{i_1}, \ldots, s_{i_t}) = p_0(s_{i_1}) b_{i_1}(\boldsymbol{x}_1) \prod_{j=2}^{t} a_{i_{j-1}, i_j} b_{i_j}(\boldsymbol{x}_j) = \prod_{j=1}^{t} \lambda(s_{i_{j-1}}, \boldsymbol{x}_j, s_{i_j}, j).$$

□

Thus, the set of observations $\mathbb{X}$ serves as alphabet for the corresponding NFPA. The emission probabilities together with the transition probabilities yield the rewards. In contrast to Dupont et al. [2005], the above NFPA yields the same reward for any path. Any path in $\mathcal{H}$ is prefixed by s_0 in W and thus one state longer. Dupont et al.

[2005] showed that there is an NFPA such that the probability $\overline{\mathrm{P}}_W(\boldsymbol{x}_{1:T})$ equals $\sum_{s_{i_1},\ldots,s_{i_T}} \mathrm{P}(\boldsymbol{x}_{1:T}, s_{i_1}, \ldots, s_{i_T})$. The PA suggested there cannot be used to decode the most likely state sequence, the so-called *Viterbi-Approximation* in Eq. (4.2.1) (Forney Jr [1973]).

Due to Lemma 2.25, $\sum_{\boldsymbol{x}\in\mathbb{X}^T} \mathrm{P}(\boldsymbol{x}) = \sum_{\boldsymbol{x}\in\mathbb{X}^T} \overline{\mathrm{P}}_W(\boldsymbol{x}) = 1$ for any HMM ($\mathrm{P}(\boldsymbol{x})$ is the marginal distribution). However, there are alternative definitions of HMMs which additionally introduce initial and final probabilities. These HMMs are equivalent to PAs such that $\sum_{\boldsymbol{x}\in\mathbb{X}^*} \mathrm{P}(\boldsymbol{x}) = 1$ (Dupont et al. [2005]).

Since any NFPA is also a WA, Algorithm 3 can be applied to the NFPA from Theorem 4.4 to calculate the optimal solution of Eq. (4.2.1) for $\mathcal{X}_i = \{x_i\}$ for each i. The joint probability can be split up into the state transition probabilities from the emission probabilities: $\mathrm{P}(\boldsymbol{x}_{1:T} \mid s_{i_1}, \ldots, s_{i_T})\,\mathrm{P}(s_{i_1}, \ldots, s_{i_T})$. This formulation yields a convenient way to integrate a LM as we will show in Section 4.2.4.

4.2.3 Parameter Optimization of HMMs

The most popular training method for HMMs is the Baum-Welch-Algorithm (Baum et al. [1970]), a special variant of the *expectation–maximization (EM) algorithm*. It was first described for discrete HMMs and for a single observation sequence. Soon, it was extended to continuous HMMs, to handle e.g. a mixture of Gaussian distributions, and multiple observation sequences. For sake of simplicity, we only introduce the simple classical method. The algorithm calculates the probabilities of being in state s_j at time t by summing up all the joint probabilities of sequences $\boldsymbol{x}_{1:t}$ and $s_{i_1}, \ldots, s_{i_t}$ ending in $s_{i_t} = s_j$.

The Baum-Welch-Algorithm introduces the so-called forward variables which denote the probability that the t-th state s_{i_t} is s_j. These forward variables correspond to the α^+ from Lemma 3.2:

$$\begin{aligned}\alpha_t^+(s_j) &= \mathrm{P}(\boldsymbol{x}_{1:t}, s_{i_t} = s_j) \\ &= \sum_{s_{i_1},\ldots,s_{i_{t-1}}\in S} \mathrm{P}(\boldsymbol{x}_t, s_j \mid s_{i_{t-1}})\,\mathrm{P}(\boldsymbol{x}_{1:t-1}, s_{i_1}, \ldots, s_{i_{t-1}})\end{aligned}$$

Let $W = (S, \mathbb{X}, \lambda, s_0, S)$ be the NFPA defined in Theorem 4.4. Then $\alpha_t^+(s_j)$ can be calculated efficiently using Lemma 3.2 ($\mathcal{X}_i = \{x_i\}$ for each i.). Obviously, $\mathrm{P}(\boldsymbol{x}_{1:T}) = \sum_s \alpha_T^+(s)$. Analogously, the backward variables β^+ of Lemma 3.2 calculate

$$\beta_t^+(s_j) = \mathrm{P}(\boldsymbol{x}_{t+1:T}, s_{i_t} = s_j).$$

In the HMM literature, the calculation of α^+ and β^+ is called the Forward-Backward-Algorithm. To calculate the new parameter, the algorithm makes use of two addi-

tional variables:

$$\begin{aligned}\xi_t(j,k) &= \mathrm{P}(s_{i_t} = s_j, s_{i_{t+1}} = s_k \mid \boldsymbol{x}_{1:T}) \\ &= \frac{\mathrm{P}(s_{i_t} = s_j, s_{i_{t+1}} = s_k, \boldsymbol{x}_{1:T})}{\mathrm{P}(\boldsymbol{x}_{1:T})} \\ &= \frac{\alpha_t^+(s_j)\,\mathrm{P}(s_k \mid s_j)\,\mathrm{P}(\boldsymbol{x}_{t+1} \mid s_k)\beta_{t+1}^+(s_k)}{\sum_s \alpha_T^+(s)}\end{aligned}$$

$$\begin{aligned}\gamma_t(j) &= \mathrm{P}(s_{i_t} = s_j \mid \boldsymbol{x}_{1:T}) \\ &= \frac{\alpha_t^+(s_j)\beta_t^+(s_j)}{\sum_s \alpha_T^+(s)}\end{aligned}$$

Instead of a direct optimization of the likelihood, HMMs usually optimize the intermediate function of the following theorem.

Theorem 4.5 ([Baum et al., 1970, Theorem 2.1]). Let $\Lambda := (\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{p}_0)$ be a configuration of model parameters, let $\overline{\Lambda}$ be another configuration and let $\boldsymbol{x}_{1:T} \in \mathbb{X}^T$ be an observation sequence. Let

$$Q(\Lambda, \overline{\Lambda}) := \sum_{s_{i_1},\ldots,s_{i_T} \in S} \mathrm{P}(s_{i_1}, \ldots, s_{i_T}, \boldsymbol{x}_{1:T} \mid \Lambda) \ln \mathrm{P}(s_{i_1}, \ldots, s_{i_T}, \boldsymbol{x}_{1:T} \mid \overline{\Lambda}).$$

If $Q(\Lambda, \overline{\Lambda}) \geq Q(\Lambda, \Lambda)$, then

$$\sum_{s_{i_1},\ldots,s_{i_T} \in S} \mathrm{P}(s_{i_1}, \ldots, s_{i_T}, \boldsymbol{x}_{1:T} \mid \overline{\Lambda}) \geq \sum_{s_{i_1},\ldots,s_{i_T} \in S} \mathrm{P}(s_{i_1}, \ldots, s_{i_T}, \boldsymbol{x}_{1:T} \mid \Lambda).$$

In other words: Increasing the intermediate function $Q(\Lambda, \overline{\Lambda})$ increases $\mathrm{P}(\boldsymbol{x}_{1:T} \mid \Lambda)$. The above theorem is a simplification to discrete $\mathbb{X}$ (which is sufficient for discrete HMMs) of the original theorem.

Theorem 4.6 ([Baum et al., 1970, p. 8]). The unique maximum of $Q(\Lambda, \overline{\Lambda})$ is given by $\overline{\Lambda} = (\overline{\boldsymbol{A}}, \overline{\boldsymbol{b}}, \overline{\boldsymbol{p}}_0)$ where for any i, j

$$\begin{aligned}\overline{a}_{i,j} &= \mathrm{P}(s_i \mid s_j) \\ &= \frac{\sum_t \xi_t(i,j)}{\sum_t \gamma_t(i)} \\ \overline{b}_i(x) &= \frac{\sum_{t=1}^T \delta_{x,x_t}\gamma_t(i)}{\sum_{t=1}^T \gamma_t(i)} \\ \overline{p}_0(i) &= \gamma_1(i).\end{aligned}$$

The proof is a straight forward minimization of $Q(\Lambda, \overline{\Lambda})$ with Lagrange Multipliers to constrain the resulting parameters according to Definition 4.3.

The computationally expensive part, the Forward-Backward-Algorithm, makes effectively use of the Dynamic Programing approach. It calculates the α^+ and β^+ values in $\mathcal{O}(Tn^2)$. A naïve summation of all possible state sequences of the objective needs $\mathcal{O}(Tn^T)$ operations[1].

Dempster et al. [1977] prove the convergence to the optimal value of $\mathrm{P}(\boldsymbol{x}_{1:T} \mid \Lambda)$ of the "classical" Baum-Welch-Algorithm. The theoretical analysis and the impressive experimental results made it the most popular training method for HMMs. However, it is very limited in the type of estimator for the posterior probabilities it can train. Hence, several iterative methods are published to train a broader class of posterior probability estimators such as Neural Networks. A detailed survey can be found in Khreich et al. [2012].

4.2.4 HMMs in Handwritten Text Recognition

Speech recognition was one of the first applications of HMMs (Rabiner [1989]). They became very popular because of the automatic time alignment and the probabilistic formulation. These two characteristics are also desirable in HTR. Due to the automatic alignment the plain transcription is sufficient in contrast to a position-wise labeling as it is required for segmentation-based methods.

Like speech recognition, handwritten text recognition requires a lot of domain knowledge. The input undergoes several preprocessing steps before it is applied to HMMs: Slant correction, baseline normalization, binarization or contrast enhancement, feature generation among others. These aspects are very complex and there are various approaches published. Since this is out of scope of this thesis, we refer to Plötz and Fink [2009] and Bluche [2015].

HMM systems typically distinguish between at least two different levels:

- The character level: Character HMMs model the various feasible characters. They work directly with features resulting from the input image. Transition probabilities are related to the expected length of a specific character. Fig. 4.1 shows the shape of such a character HMM.
- The character sequence level: A Weighted Automaton accepts feasible character sequences. The transition probabilities typically result from external language resources such as vocabularies or n-grams.

The lowest level is typically covered by the actual HMMs which recognize the most basic shapes of the model. Instead of characters, the basic shape may be *graphem* models. Graphems are the smallest writing units. The corresponding concept in speech is called phoneme. Since HMMs can be interpreted as Weighted Automata / Weighted FSTs, two levels can be combined by the Weighted FST composition (see Mohri [2009]). For instance, a composition of such graphem HMMs and a character

[1] Recall that $n = |S|$.

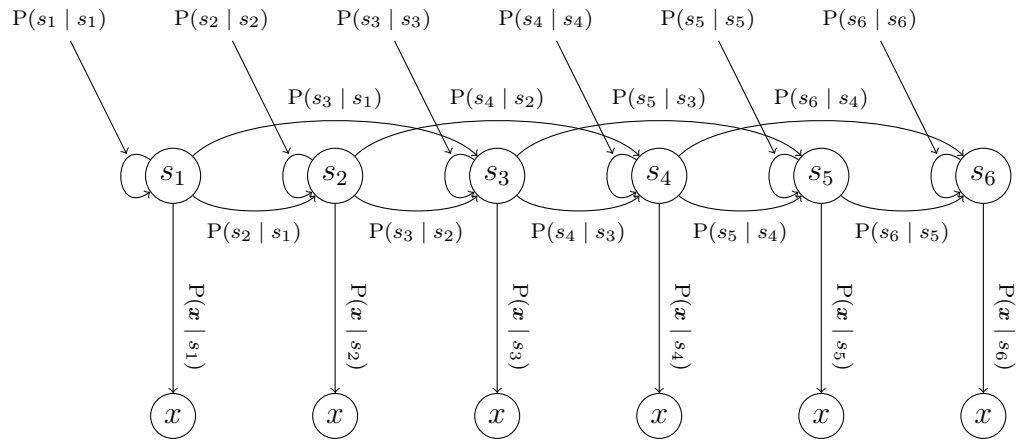


Figure 4.1: Possible shape of a left-to-right character HMM. Each state generates a probability of the current observation. Thus, a valid interpretation is that the detection of a character is split into the detection of a sequence of parts of the character.

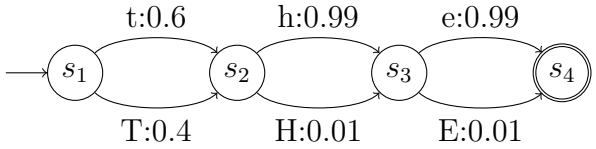


Figure 4.2: Possible shape of the Weighted Automaton at word level accepting different spellings of "the". The colon seperates the emitted character and its character bigram probability.

FST which reads graphems and outputs a character is able to recognize characters.

Either the character sequences are modeled directly using an n-gram at character level to determine the transition weights or the character sequence level is subdivided into a word level (also called lexical level) and language level (also called syntactical level): Different feasible character sequences of the same word constitute a word level Automaton (e.g. the, The and THE) representing the different spellings (see Figure 4.2). For the syntactical level, the corresponding Automaton accepts sequences of words and weights them according to a Language Model (typically an n-gram).

As already mentioned, the different levels are formulated as Finite State Transducers. The composition of the layers yields the entire system. The transition probabilities of each of the levels sum to one such that the underlying Automaton is an NFPA. Many authors, especially in the handwritten text and speech recognition such as Vidal et al. [2005] or Mohri et al. [2008], propose additional initial and finial probabilities such that the corresponding Automata of these FSTs are PA instead of NFPAs. This enables to model extended n-grams correctly, for example.

According to Plötz and Fink [2009] "practically all current recognizers are derived by applying (variants of) Baum–Welch training on sample data [...] resulting in most cases in (semi-) continuous output models. They are usually based on Gaussian mixture densities". *Gaussian mixtures* imply continuous HMMs with the emission probability

$$b_j(\boldsymbol{x}) := \mathrm{p}(\boldsymbol{x} \mid s_j) = \sum_i c_{j,i} \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_{j,i}, \boldsymbol{\Sigma}_{j,i})$$

where $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is the density of the multivariate normal distribution

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right).$$

The coefficients $c_{j,i} \in [0, 1]$ must sum to 1, i.e., $\sum_i c_{j,i} = 1$. Among the best HMM handwriting systems there are also combinations of HMMs and Neural Networks: Hybrid systems use Neural Networks as posterior probability estimators (Doetsch et al. [2014]). Tandem systems usually rely on Gaussian mixtures but the features are generated by Neural Networks. Well performing tandem systems were proposed in e.g. Bluche et al. [2013] (Convolutional Neural Network-GMM-HMM tandem system) or Kozielski et al. [2013a] (BLSTM-GMM-HMM tandem system).

It is a great advantage that HMMs include Language Model probabilities into the probability of the final label sequence in an inherent way. However, it was reported that some scaling is necessary in order to optimize the performance: For $\boldsymbol{z}_{1:n}$ words, HMM decoders practically calculate

$$\boldsymbol{s}^*_{1:T} = \underset{s_{i_1}, \ldots, s_{i_t}}{\arg\max}\, \mathrm{p}(\boldsymbol{x}_{1:T} \mid s_{i_1}, \ldots, s_{i_T}) \frac{\mathrm{P}(s_{i_1}, \ldots, s_{i_T})^\rho}{\gamma^n}.$$

The parameter ρ is called *Grammar Scale Factor* and γ is know as *Word Insertion Penalty* (e.g. Plötz and Fink [2009], Zimmermann and Bunke [2004]). Although Dynamic Programming is applied, the search space can be exhausting especially for higher-order Language Models. Pruning techniques such as Beam Search reduce the search space to keep the decoding practically tractable. A Beam Search variant is introduced in Section 6.3.2.

4.3 Neuronal Networks

Neural Networks are an abstract model of the brain. They are successfully used for regression and classification tasks. Here we introduce the basic concepts which are necessary to understand the experiments of the following chapters.

4.3.1 Definition and Architecture

Classical Models

The first article dealing with Artificial Neural Networks is commonly cited as McCulloch and Pitts [1943]. It investigates neural behavior of an ideal simplified neural model and counts also as inception of *Finite State Automata*. Further research of these Neural Networks by Kleene (Kleene [1956]) lead to the formulation of *Regular Expressions* which we introduced in Chapter 2.

The modern neural model as we know it today is commonly attributed to Rosenblatt [1958]. In contrast to previous works its synaptic connections were weighted. The following definition is closer to Minsky and Papert [1969] although they used the threshold function ($\mathbf{1}_{>0}(x) := 1$ if $x > 0$ and 0 else) instead of a (piecewise) continuous, monotonously increasing function which is commonly used nowadays.

Definition 4.7 (Perceptron). For a given function $\psi : \mathbb{R} \to \mathbb{R}$, $\boldsymbol{w} \in \mathbb{R}^d$ and $\theta \in \mathbb{R}$ the function

$$\Psi : \mathbb{R}^d \to \mathbb{R}; \quad \boldsymbol{x} \mapsto \psi(\boldsymbol{w}^T \boldsymbol{x} + \theta)$$

with the standard scalar product $\boldsymbol{w}^T \boldsymbol{x}$ is called *Perceptron* and ψ is called the *activation function*.

The Perceptron is the classical model of a neuron. However, there is a huge amount of neural models such that there is no precise definition of Neural Networks. In this thesis, a *(Artificial) Neural Network (NN)* consists of a set of Perceptrons which process input and generate an output. In the simplest case, several Perceptrons processing the same input form a layer, several consecutive layers form the Multi-Layer Perceptron:

Definition 4.8 (Multi-Layer Perceptron). For a given number $n \in \mathbb{N}$ of *layers* Λ_l and for each $1 \leq l \leq n$, let $d_l \in \mathbb{N}$ be the number of Perceptrons in layer l. For each $i \leq d_l$, $\Psi_{l,i}(\boldsymbol{x}) := \psi_{l,i}(\boldsymbol{w}_{l,i}^T \boldsymbol{x} + \theta_{l,i})$ denotes the i-th Perceptron in layer l with the parameters $\boldsymbol{w}_{l,i} \in \mathbb{R}^{d_{l-1}}$ and $\theta_{l,i} \in \mathbb{R}$. A *Multi-Layer Perceptron (MLP)* with $n \in \mathbb{N}$ computed layers is a function $\Lambda_n : \mathbb{R}^{d_0} \to \mathbb{R}^{d_n}$ such that the l-th layer is calculated by

$$\Lambda_l(\boldsymbol{x}) := \Big(\Psi_{l,1}(\Lambda_{l-1}(\boldsymbol{x})), \ldots, \Psi_{l,d_l}(\Lambda_{l-1}(\boldsymbol{x}))\Big),$$

where $\Lambda_0(\boldsymbol{x}) := \boldsymbol{x}$. The *layer* Λ_0 is called *input layer*, Λ_n is called *ouput layer* and $\boldsymbol{x}$ is the input.

An exemplary Multi-Layer Perceptron with one hidden layer is given by Figure 4.3.

Definition 4.9 (Squashing function). An activation function $\psi : \mathbb{R} \to \mathbb{R}$ is called *squashing function* (also sigmoid function) if it is bounded, monotonically increasing and continuous.

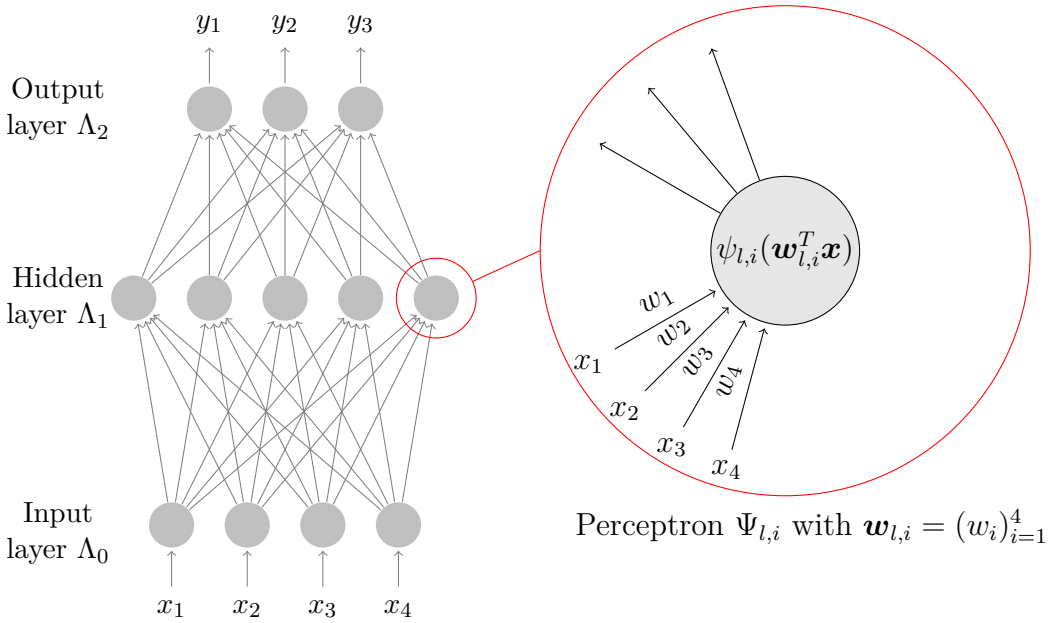


Figure 4.3: Multi-Layer Perceptron with 2 computed layer. $\boldsymbol{d} = (4, 5, 3)$

Typical examples are $\psi(x) = \tanh(x)$, $\psi(x) = \arctan(x)$ or $\psi(x) = (1 + e^{-x})^{-1}$. Besides squashing functions there are other popular activation functions e.g. $\psi(x) = \max\{0, x\}$ (the so-called *rectifier*). A natural extension of the ordinary activation function of a Perceptron is a layer-wise activation function. A popular example is the following function:

Definition 4.10 (Softmax function). The *Softmax function* $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ is defined by

$$\sigma(\boldsymbol{x})_j := \frac{\exp(x_j)}{\sum_{i=1}^n \exp(x_i)}.$$

Softmax is very popular as an activation function since the values sum to one and are all positive such that NNs with Softmax activation function of the last layer are able to model probability distributions.

Up to now, we introduced only forward structured Neural Networks (Feed-Forward Networks) without circles and independent of the previous input. A simple recurrent extension of Definition 4.8 is the following:

Definition 4.11 (Recurrent Multi-Layer Perceptron). Assume the notation of Definition 4.8. For a given time series $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots, \boldsymbol{x}(T) \in \mathbb{R}^{d_0}$ of inputs, a *Recurrent Multi-Layer Perceptron* Λ_n is a function $\Lambda_n : \mathbb{R}^{d_0 \times T} \to \mathbb{R}^{d_n \times T}$. It is calculated iteratively: Let $\Lambda_l(0) := \mathbf{0}$ [2] for any $1 \leq l \leq n$ and $\Lambda_0(t) = \boldsymbol{x}(t)$ for

[2] The initialization with zero is quite common. Nevertheless, any other initial $\Lambda_l(0) \in \mathbb{R}^{d_l}$ is possible.

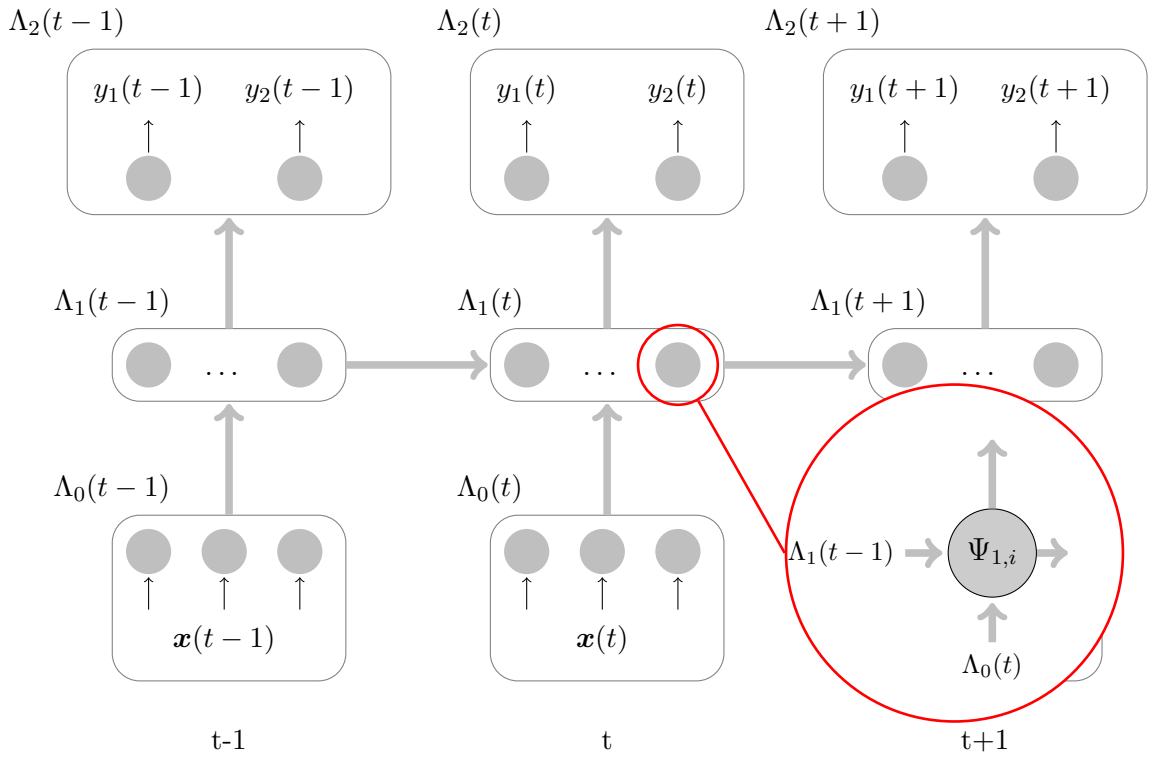


Figure 4.4: Recurrent Multi-Layer Perceptron with 2 computed layer, 2 output Perceptrons and 3 input Perceptrons unfolded in time. Bold gray arcs indicate multi-connections between many Perceptrons.

$1 \leq t \leq T$. With $\boldsymbol{x}_l(t) := (\Lambda_{l-1}(t), \Lambda_l(t-1))$,

$$\Lambda_l(t) := \Big(\Psi_{l,1}(\boldsymbol{x}_l(t)), \ldots, \Psi_{l,d_l}(\boldsymbol{x}_l(t))\Big).$$

Compared to a Multi-Layer Perceptron, the Recurrent Multi-Layer Perceptron has additional connections (i.e., dependencies) to the same layer of the previous time step. Figure 4.4 shows a scheme of a Recurrent Multi-Layer Perceptron. Generally, a NN with recurrent connections is typically denoted as Recurrent Neural Network (RNN).

Modern Approaches

Recently, several neural models with many layers were proposed under the common label *Deep Learning*. Many articles confirm a high capability of approximation for a large class of functions. Various approaches were proposed like Convolutional Neural Networks, Autoencoders or Restricted Boltzmann Machines which are out of scope of this thesis. The parameter optimization of such "deep" structures is not trivial and requires advanced learning techniques. A survey on Deep Learning reviewing

the above mentioned models and many others can be found in Schmidhuber [2015]. In this thesis, we focus on Neural Networks containing so-called Cells. These neurons are more advanced compared to Perceptrons and became popular in HTR:

Definition 4.12 (Cell). A *Cell* is a function $c : \mathbb{R}^d \to \mathbb{R}^2$. We denote $c(\boldsymbol{x}) := \Big(s(\boldsymbol{x}), o(\boldsymbol{x}, s(\boldsymbol{x}))\Big)$. The functions s and o are composed of k Perceptrons $\Psi_i(\boldsymbol{x})$ $(i = 1, \ldots, k)$, the so-called *gates*: For two specific functions f, g with $f : \mathbb{R}^k \to \mathbb{R}$ and $g : \mathbb{R}^{k+1} \to \mathbb{R}$, let

$$s(\boldsymbol{x}) := f(\Psi_1(\boldsymbol{x}), \ldots, \Psi_k(\boldsymbol{x})) \text{ and } o(\boldsymbol{x}, s(\boldsymbol{x})) := g(\Psi_1(\boldsymbol{x}), \ldots, \Psi_k(\boldsymbol{x}), s(\boldsymbol{x})).$$

The first component $s(\boldsymbol{x})$ is typically referred to as *internal state*, the second component is typically referred to as *output value*.

Cells are usually defined as recurrent neurons such that the input $\boldsymbol{x}$ consists of the output values from a predecessor layer, in addition to values (internal state and output values) from the same layer one time step before.

Definition 4.13 (Recurrent Layer of Cells). Let $c_{l,i} = (s_{l,i}, o_{l,i})$ $(i = 1, \ldots, d_l)$ be the Cells of the l-th *Recurrent Layer of Cells* Λ_l within a Neural Network. Then

$$\Lambda_l(t) := \Big(o_{l,1}(t), \ldots, o_{l,d_l}(t)\Big)$$

where

$$\begin{aligned} s_{l,i}(t) :=& s_{l,i}\Big(\boldsymbol{x}_{i,l}(t)\Big), \\ o_{l,i}(t) :=& o_{l,i}\Big(\boldsymbol{x}_{i,l}(t), s_i(t)\Big) \end{aligned}$$

where $\boldsymbol{x}_{i,l}(t) := \Big(\Lambda_{l-1}(t), \Lambda_l(t-1), s_i(t-1)\Big)$ and $\Lambda_0(t) := \boldsymbol{x}(t)$ provides the external input $(t = 1, \ldots, T)$.

That means, the Cell input also comprises its own internal state $s_{l,i}(t-1)$ besides the output $\Lambda_l(t-1)$ of layer l and the output $\Lambda_{l-1}(t)$ of layer $l-1$. Figure 4.5 shows an exemplary structure of such Recurrent Layer of Cells.

Example 4.14. To better understand the specific structure of the Cells it is reasonable to introduce $\boldsymbol{u}_l(t) := (\Lambda_{l-1}(t), \Lambda_l(t-1))$ since typically the outputs of the layer below and the outputs of the same layer at the previous time step are processed in the same way while $s_i(t)$ has a different functionality.

- The *Long Short Term Memory Cell (LSTM)* from Hochreiter and Schmidhuber [1997] is the classical Cell with the ability to memorize, forget and output information:

$$\begin{aligned} s_i(t) &= \Psi_1^i(\boldsymbol{u}_l(t))\ \Psi_2^i(\boldsymbol{u}_l(t)) + \Psi_3^i(\boldsymbol{u}_l(t))\ s_i(t-1) \\ o_i(t) &= \psi_4^i(s_i(t))\ \Psi_5^i(\boldsymbol{u}_l(t)) \end{aligned}$$

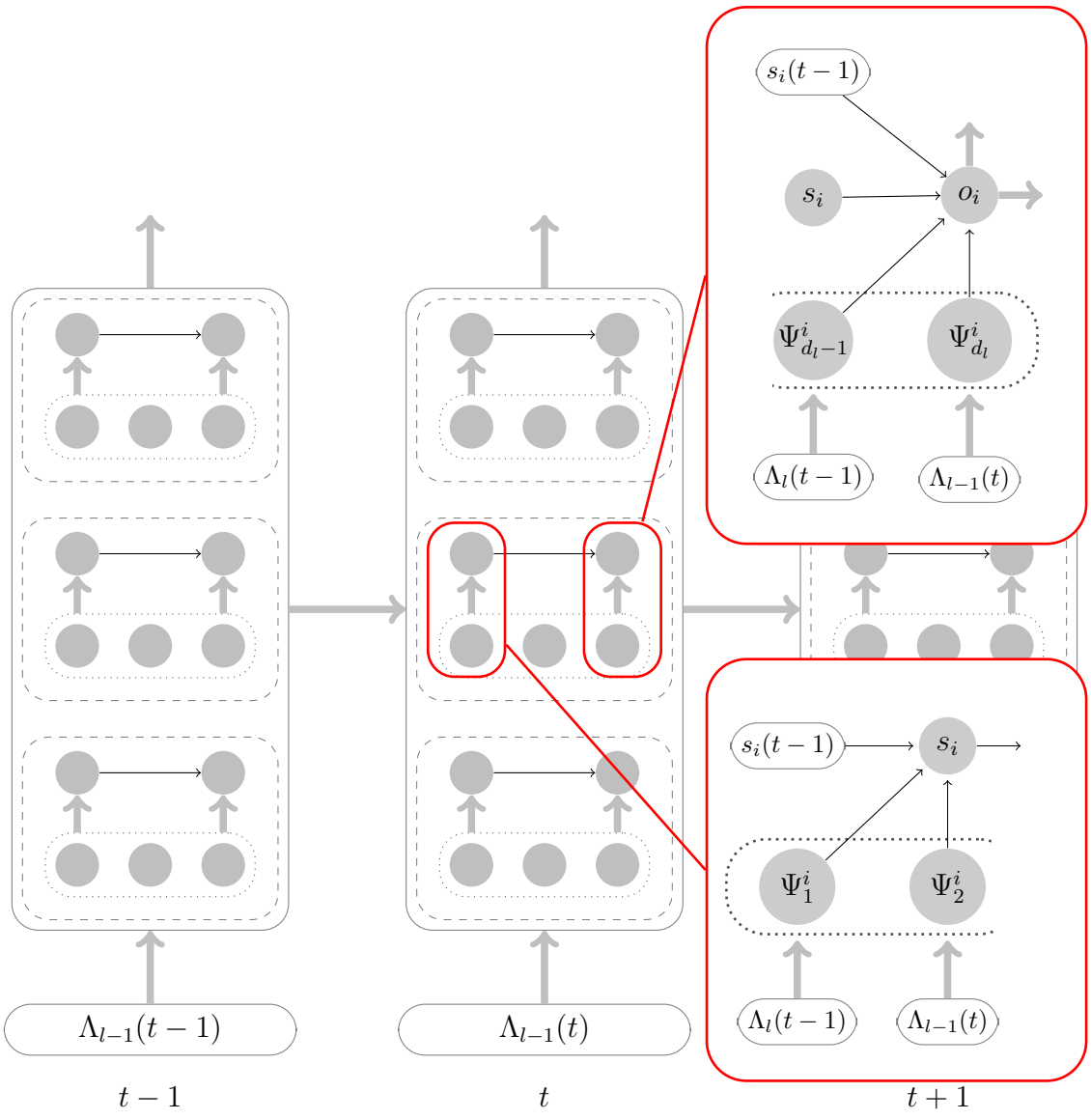


Figure 4.5: Recurrent Layer of Cells unfolded in time. Broad arcs indicate multi-connections.

where Ψ_1^i is the input, Ψ_2^i is the input gate, Ψ_3^i is the forget gate, Ψ_5^i is the output gate which are standard Perceptrons. ψ_4^i is a squashing function. The activation functions ψ_1^i and ψ_4^i are usually chosen to be tanh. ψ_2^i, ψ_3^i and ψ_5^i are usually the logistic function $f(x) = (1 + \exp(-x))^{-1}$.

- For the *LeakyLP Cell (LeakyLP)* from Leifert et al. [2014b], the new internal state is a convex combination of the new information $\Psi_1^i(\boldsymbol{u}(t))$ and the internal state $s_i(t-1)$ (from a previous time):

$$\begin{aligned} s_i(t) &= \Big(1 - \Psi_2^i(\boldsymbol{u}_l(t))\Big)\Psi_1^i(\boldsymbol{u}_l(t)) + \Psi_2^i(\boldsymbol{u}_l(t))s_i(t-1) \\ o_i(t) &= \psi_3^i\Big(s_i(t)\Psi_4^i(\boldsymbol{u}_l(t)) + s_i(t-1)\Psi_5^i(\boldsymbol{u}_l(t))\Big) \end{aligned}$$

  where Ψ_1^i is the input, Ψ_2^i is the forget gate, Ψ_4^i and Ψ_5^i are the two output gates. Again, all gates are standard Perceptrons. The activation functions ψ_1^i and ψ_3^i are defined as $\tanh(x)$. ψ_2^i, ψ_4^i and ψ_5^i are usually the logistic function.
- The *Gated Recurrent Unit*, introduced in Cho et al. [2014], is a popular Cell which does not follow the Definition 4.12 since it needs an additional intermediate Feed-Forward layer to compute the internal state. Let $r_1(t), \ldots, r_{d_l}(t)$ denote the output of this additional layer. Then

$$r_j(t) = \Psi_j^r(\boldsymbol{u}_l(t)).$$

  The output r_i scales the output $o_i(t-1)$:

$$\begin{aligned} \overline{\boldsymbol{x}}_l(t) &= \Big(\Lambda_{l-1}(t), r_1(t)o_1(t-1), \ldots, r_{d_l}(t)o_{d_l}(t-1)\Big) \\ s_i(t) &= \Psi_1^i(\overline{\boldsymbol{x}}_l(t)) \\ o_i(t) &= \Big(1 - \Psi_2^i(\boldsymbol{u}_l(t))\Big)o_i(t-1) + \Psi_2^i(\boldsymbol{u}_l(t))s(t) \end{aligned}$$

  where Ψ_j^r $(j = 1, \ldots, d)$ are the reset gates, Ψ_2^i is called the update gate. The activation function ψ_1^i is usually chosen to be tanh. ψ_j^r and ψ_2^i are usually the logistic function.

The above Cells are the most popular in the literature. Note that for all Cells there are further details which are practically relevant but not introduced. We provide only the conceptional background here.

We will use the first two and extend them to multi-dimensional input sequences.

Remark 4.15 (Multidimensional Cells). In many practical applications, the input time series is q-dimensional with $q > 1$, i.e., the time (or equivalently position) $\boldsymbol{p} \in \mathbb{N}^q$ is q-dimensional instead of a scalar t. Hence, there are q directly preceding time steps or positions instead of one which directly contribute to the input of the

Cell input via recurrent connections. That means,

$$\begin{aligned}\boldsymbol{x}_{l,i}(\boldsymbol{p}) :=& \Big(\Lambda_{l-1}(\boldsymbol{p}), \Lambda_l(\boldsymbol{p}_1^-), \ldots, \Lambda_l(\boldsymbol{p}_q^-), s_i(\boldsymbol{p}_1^-), \ldots, s_i(\boldsymbol{p}_q^-)\Big)\\ s_{l,i}(\boldsymbol{p}) :=& s_{l,i}\Big(\boldsymbol{x}_{l,i}(\boldsymbol{p})\Big),\\ o_{l,i}(\boldsymbol{p}) :=& o_{l,i}\Big(\boldsymbol{x}_{l,i}(\boldsymbol{p}), s_i(\boldsymbol{p})\Big)\end{aligned}$$

where $\boldsymbol{p}_j^-$ is the position $\boldsymbol{p}$ minus 1 in dimension j. The LSTM and the LeakyLP Cells generalize in a natural way to higher dimensions (see Graves et al. [2007] and Leifert et al. [2014b]). Multidimensional LSTM Cells are commonly referred to as *MD-LSTM* Cells.

Definition 4.16 (Truncated Gradient). Let $\boldsymbol{x}(\boldsymbol{p})$ be the input of the j-th Perceptron Ψ_j^i of Cell c_i at position $\boldsymbol{p}$ and $o_k(\boldsymbol{p}_l^-)$ any previous output activation from the same layer. The *Truncated Gradient* differs from the exact gradient only by setting recurrent weighted gradient propagation $\frac{\partial \Psi_j^i(\boldsymbol{x}(\boldsymbol{p}))}{\partial o_k(\boldsymbol{p}_l^-)}$ to zero.

For two positions $p, p' \in \mathbb{N}^q$, $p' < p$ iff $p'_i \leq p_i$ for any i and $p'_j < p_j$ for at least one j.

Theorem 4.17 (Leifert et al. [2014b]). For MD-LSTM Cells, the Truncated Gradient of an internal state at position $\boldsymbol{p}$ with respect to the internal state at position $\boldsymbol{p}'$ with $\boldsymbol{p}' < \boldsymbol{p}$ is not necessarily in $[0, 1]$. For LeakyLP cells, the Truncated Gradient is bounded by $[0, 1]$.

See Theorem 15, Theorem 20 and the remark after Definition 21 of the cited article for the proof.

If the gradient is bounded, the influence of the internal states cannot grow over time. Thus, the internal memory is stable. For multi-dimensional LSTM Cells (MD-LSTM) the memory is not bounded. Thus, the impact of the internal state at a certain position may increase for subsequent positions such that it superimposes all new information. We prefer the LeakyLP Cells especially for lower layers and MD-LSTMs for higher layers as it is suggested in Leifert et al. [2014b].

In addition, many systems introduce a so-called *subsampling* from some layer Λ_{l-1} to Λ_l. In a one dimensional case that means, the output of $s > 1$ time steps of the $(l-1)$-th layer form the input of the l-th layer. For example, a *subsampling rate* of $s = 2$ means $\Lambda_{l-1}(1)$ and $\Lambda_{l-1}(2)$ serve as input for $\Lambda_l(1)$, $\Lambda_{l-1}(3)$ and $\Lambda_{l-1}(4)$ serve as input for $\Lambda_l(2)$, etc. So, the number of time steps in which the l-th layer generates an output is down sampled to only $\lceil T/s \rceil$ if T is the output number of the $(l-1)$-th layer.

Another very important concept for HTR is the multi-directional RNN. Such an RNN contains a layer Λ_l which comprises 2^q sublayers where q is the dimension of the time. In case of a one-dimensional time, let us denote these two sublayers by Λ_l^1 and

Λ_l^2. Then one sublayer, w.l.g. $\Lambda_l^1(t)$, depends on $\Lambda_l^1(t-1)$ via recurrent connections as sketched in Figure 4.5. The other sublayer $\Lambda_l^2(t)$ depends on the $\Lambda_l^2(t+1)$ in positive time direction such that it is able to remember and process information from the opposite direction. But that means $\Lambda_l^2(t+1)$ has to be calculated before $\Lambda_l^2(t)$. This concept obviously assumes a finite input. The output of layer $\Lambda_l(t)$ is the combined output of both sublayers $(\Lambda_l^1(t), \Lambda_l^2(t))$ and contains information from both directions. NNs with such an LSTM Cell layer are called *bi-directional LSTM Networks (BLSTM)*. For time dimensions greater than 1, this concept transfers in a natural way to 2^q possible directions. Then such NNs are called *multi-directional LSTM Networks*[3].

Remark 4.18. In HTR, the two dimensional input sequence is finite and all sequence elements are known in advance. We represent them as a matrix $\boldsymbol{X} \in \bigcup_{m,n} \mathbb{R}^{m \times n}$ in the following.

4.3.2 Parameter Optimization of Neural Networks

Traditionally, the parameters $\boldsymbol{W}$ of a NN[4] are optimized iteratively using Gradient Descent. Other learning techniques could not compete in general and are limited to specific conditions (Echo-State-Networks Jaeger [2002], Hebb's rule Hebb [2005]). A good overview of gradient learning techniques is given by Ruder [2016] which we follow here. Let $\mathbb{D}$ be the data set (of input samples $\boldsymbol{X}$ and corresponding target samples $\boldsymbol{z} \in \mathbb{A}^*$) used for training. Assume the following optimization problem:

$$\mathcal{E}(\mathbb{D}, \boldsymbol{W}) \to \min$$

where $\mathcal{E}$ is the so-called error function. To apply Gradient Descent, $\mathcal{E}$ has to be at least piecewise continuously differentiable. The so-called *Batch Gradient Descent* updates the parameter by

$$\begin{aligned} \boldsymbol{\Delta}_n &:= \eta \, \nabla_{\boldsymbol{W}} \mathcal{E}(\mathbb{D}, \boldsymbol{W}_n) \\ \boldsymbol{W}_{n+1} &:= \boldsymbol{W}_n - \boldsymbol{\Delta}_n, \end{aligned}$$

where η denotes the *learning rate*. That means the gradient is calculated over the whole data set which can be exhausting for huge data sets. The opposite is *Stochastic Gradient Decent* where the gradient is calculated and $\boldsymbol{W}$ is updated only for a single date instead of the whole data set $\mathbb{D}$. A compromise of both variants is the *Mini-Batch Gradient Descent* where the gradient is calculated for a reasonable small but sufficient large subset of $\mathbb{D}$.

The gradient itself is calculated by extensive exploit of the chain rule for differ-

[3] Do not confuse multi-directional LSTM with multi-dimensional LSTM which both may be abbreviated by MD-LSTM. However, MD-LSTM mainly abbreviates multi-dimensional LSTM in the literature (and this thesis).

[4] The parameters are basically the weight vectors $\boldsymbol{w}_{l,i}$ of the Perceptrons.

entiation which is called *Backpropagation*. Extensions for RNNs incorporating the additional time-dependency are called *Backpropagation Through Time* or *Realtime Recurrent Learning*.

Besides these elementary approaches there were several adaptations proposed: The *Momentum* method also takes previous directions into account to prevent for fluctuation in the gradient: $\boldsymbol{\Delta}_n := \gamma\, \boldsymbol{\Delta}_{n-1} + \eta\, \nabla_{\boldsymbol{W}} \mathcal{E}$ for some $\gamma, \eta \in [0, 1)$. *Nesterov Accelerated Gradient* builds up on the Momentum method: Since the major part of the direction is typically given by the Momentum $\boldsymbol{\Delta}_{n-1}$, $\overline{\boldsymbol{W}}_n := \boldsymbol{W}_n - \gamma\, \boldsymbol{\Delta}_{n-1}$ can be used as an estimate for the future direction. Hence, the gradient $\nabla_{\boldsymbol{W}} \mathcal{E}$ is calculated at $\overline{\boldsymbol{W}}_n$ instead of $\boldsymbol{W}_n$. *Adagrad*, *Adadelta*, *RMSprop* and *Adam* are advanced optimization techniques which change the learning rate of each component $\frac{\partial \mathcal{E}}{\partial w_{i,j}}$ of the gradient individually depending on past gradients of the specific component. This is an approach to tackle the problem of *vanishing gradients* which states that the gradient of lower layers is almost zero such that the training of those layers lasts longer. A more detailed introduction to these techniques can be found in Ruder [2016]. This article also covers a basic introduction to parallelization techniques.

A very important aspect of learning the parameters of a NN is the regularization: The problem of adapting to the training data without generalization is commonly referred to as *overfitting*. Different suggestions are proposed to tackle this problem: Small transformations of the input (e.g. a morphological transformation in case of HTR) or adding small random perturbations to the activations or weights are the classical approaches. Recently, several authors investigated a concept called *Dropout*. There a proportion of the hidden neurons (i.e., not from the input or output layer) is removed randomly during the training phase (Srivastava et al. [2014]). It has been shown to improve the generalization and the training speed. Equivalently, one could remove a random percentage of the weights which is called *DropConnect* (see Wan et al. [2013]).

4.3.3 Estimating Probabilities by Neural Networks

The problem of estimating conditional probabilities may refer to two tasks: Either there is a given parameterized family of distributions whose parameter are unknown and have to be estimated by the NN. Or second, the output of the NN itself is interpreted as probability. We only consider the second case. We first show that NNs are capable of estimating probabilities and show how to train them subsequently.

Neural Networks are Universal (Probability) Approximators

Minsky and Papert [1969] showed that a single layer of Perceptrons is not able to reproduce a simple XOR function which lead to the *abandonment of connectionism* until the early 1980s. The following result states that a Multi-Layer Perceptron with

one hidden layer and identity function as output activation function is sufficient to reproduce any function from the practically relevant class of continuous functions with arbitrary exactness on compact subsets of the $\mathbb{R}^r$:

Theorem 4.19 (Hornik et al. [1989]). For any $r, s \in \mathbb{N}$ and any squashing function ψ, the set

$$\Sigma^{r,s}(\psi) := \left\{ f : \mathbb{R}^r \to \mathbb{R}^s \;\middle|\; \forall \boldsymbol{u} \in \mathbb{R}^r : f(\boldsymbol{x})_i = \sum_{j=1}^{N} \beta_{i,j} \Psi_j(\boldsymbol{x}), \quad N \in \mathbb{N}, \beta_{i,j} \in \mathbb{R} \right\}$$

(where the activation function of any Ψ_j is ψ) is ρ_K dense in the set $C^{r,s}$ of continuous functions $f : \mathbb{R}^r \to \mathbb{R}^s$ for any compact set $K \subsetneq \mathbb{R}^r$. That means, for any $\epsilon > 0$ and for any $g \in C^{r,s}$ there is a $\varphi \in \Sigma^{r,s}$ such that $\rho_K(g, \phi) := \sup_{\boldsymbol{x} \in K} \|g(\boldsymbol{x}) - \varphi(\boldsymbol{x})\|_\infty < \epsilon$.

The same result also holds for NNs with squashing functions at the output layer (Castro et al. [2000]). To the best of our knowledge, the result was not extended to MLPs with Softmax activation function at the output layer. Let $C_1^{r,s}$ be the set of discrete (finite) conditional probability distributions which depend continuously on the conditioned argument:

$$C_1^{r,s} := \left\{ f \in C^{r,s} \;\middle|\; \forall \boldsymbol{x} \in \mathbb{R}^r : \sum_{i=1}^{s} f(\boldsymbol{x})_i = 1 \wedge \forall i : f(\boldsymbol{x})_i \geq 0 \right\}.$$

Let further for $L \subset [0,1]^s$

$$C_1^{r,s}(L) := \{ f \in C_1^{r,s} \mid \forall \boldsymbol{x} \in \mathbb{R} : f(\boldsymbol{x}) \in L \}.$$

Corollary 4.20. For any squashing function ψ and any $r, s \in \mathbb{N}$, the set

$$\Upsilon^{r,s}(\psi) := \left\{ g : \mathbb{R}^r \to \mathbb{R}^s \;\middle|\; \exists f \in \Sigma^{r,s}(\psi) : \; g = f \circ \sigma \right\}$$

of MLPs with Softmax output activation function is ρ_k dense in $C_1^{r,s}$ for any compact set $K \subsetneq \mathbb{R}^r$.

Proof. Let

$$L_\delta := \left\{ \boldsymbol{x} \in [\delta, 1-\delta]^s \;\middle|\; \sum_{i=1}^{s} x_i = 1 \right\}.$$

For any $\boldsymbol{y} \in [0,1]^s$ with $\sum_i y_i = 1$,

$$\min_{\boldsymbol{x} \in L_\delta} \|\boldsymbol{y} - \boldsymbol{x}\|_\infty \leq \min_{\boldsymbol{x} \in L_\delta} \|\boldsymbol{e}_i - \boldsymbol{x}\|_\infty = \min\{1 - (1-\delta), \delta\} = \delta$$

where $\boldsymbol{e}_i$ is the i-th unit vector or equivalently the i-th column of the identity matrix. Hence, for any ϵ there is a δ such that for any $\boldsymbol{y} \in [0,1]^s$ with $\sum_i y_i = 1$

$$\min_{\boldsymbol{x} \in L_\delta} \|\boldsymbol{y} - \boldsymbol{x}\|_\infty \leq \frac{\epsilon}{2}.$$

This means in particular that for any compact $K \subsetneq \mathbb{R}$ for any $\epsilon > 0$ and for any $g \in C_1^{r,s}$ there are a $\delta > 0$ and a $h \in C_1^{r,s}(L_\delta)$ such that

$$\sup_{\boldsymbol{x} \in K} \|g(\boldsymbol{x}) - h(\boldsymbol{x})\|_\infty < \frac{\epsilon}{2}.$$

Note that $h \circ \ln \in C^{r,s}$ for $h \in C_1^{r,s}(L_\delta)$. Since $\Sigma^{r,s}(\psi)$ is ρ_k dense in $C^{r,s}$, for any $h \in C_1^{r,s}(L_\delta)$ and any $\bar{\epsilon} > 0$ there is $\varphi \in \Sigma^{r,s}(\psi)$ such that

$$\bar{\epsilon} > \sup_{\boldsymbol{x} \in K} \| \ln(h(\boldsymbol{x})) - \varphi(\boldsymbol{x})\|.$$

Hence, for any $\boldsymbol{x} \in K$ and $i \in \{1, \ldots, s\}$

$$\exp(-\bar{\epsilon}) < \frac{h(\boldsymbol{x})_i}{\exp(\varphi(\boldsymbol{x})_i)} < \exp(\bar{\epsilon}).$$

Elementry operations lead to

$$\begin{aligned}
\max_i \sup_{\boldsymbol{x} \in K} |h(\boldsymbol{x})_i - \sigma(\varphi(\boldsymbol{x}))_i| &= \max_i \sup_{\boldsymbol{x} \in K} \left| h(\boldsymbol{x})_i - \frac{\exp(\varphi(\boldsymbol{x})_i)}{\sum_j \exp(\varphi(\boldsymbol{x})_j)} \right| \\
&< \max_i \sup_{\boldsymbol{x} \in K} \left| h(\boldsymbol{x})_i - \frac{h_i(\boldsymbol{x}) \exp(-\bar{\epsilon})}{\exp(\bar{\epsilon}) \sum_j h(\boldsymbol{x})_j} \right| \\
&= \max_i \sup_{\boldsymbol{x} \in K} |h(\boldsymbol{x})_i| (1 - \exp(-2\bar{\epsilon})) \\
&\leq (1 - \delta)(1 - \exp(-2\bar{\epsilon})).
\end{aligned}$$

Hence for sufficiently small $\bar{\epsilon}$, we find a φ such that $\epsilon/2 > \sup_{\boldsymbol{x} \in K} \|h(\boldsymbol{x}) - \sigma(\varphi(\boldsymbol{x}))\|$. Finally, for any compact $K \subsetneq \mathbb{R}^r$, for any $g \in C_1^{r,s}$ there is a $\varphi \in \Upsilon^{r,s}(\psi)$ such that

$$\sup_{\boldsymbol{x} \in K} \|g(\boldsymbol{x}) - \sigma(\varphi(\boldsymbol{x}))\| < \sup_{\boldsymbol{x} \in K} \|g(\boldsymbol{x}) - h(\boldsymbol{x})\| + \sup_{\boldsymbol{x} \in K} \|h(\boldsymbol{x}) - \sigma(\varphi(\boldsymbol{x}))\| < \epsilon.$$

□

This especially means, that any discrete conditional probability distribution from $C_1^{r,s}$ can be approached by an MLP with one hidden layer and Softmax output layer activation function on any compact $K \subsetneq \mathbb{R}^r$. By setting all recurrent connections to 0 the above corollary also holds for Recurrent Multi-Layer Perceptrons. Even LSTM and LeakyLP Cells are able to cover the traditional Perceptron behavior which allows to extend the statement to such NNs.

Neural Probability Estimators

Corollary 4.20 states that NNs are theoretically capable of approximating conditional probability distributions and in fact Recurrent Neural Networks are also successfully

trained to maximize the conditional likelihood in practice.

The classical approach to fit the underlying distribution of the data $\mathbb{D}$ is the *maximum likelihood estimation*. For a Neural Network with parameters $\boldsymbol{W}$ this means

$$\mathcal{E}(\mathbb{D}, \boldsymbol{W}) = \prod_{(\boldsymbol{X},\boldsymbol{u})\in\mathbb{D}} \mathrm{P}(\boldsymbol{u} \mid \boldsymbol{X}; \boldsymbol{W}) \to \max \tag{4.3.1}$$

where $\boldsymbol{u}$ is the target label sequence and $\boldsymbol{X}$ is the corresponding input (observation in terms of HMMs). For given $\boldsymbol{X}$, the Neural Network's likelihood estimate is

$$\mathrm{P}(\boldsymbol{u} \mid \boldsymbol{X}, \boldsymbol{W}) = \prod_{t=1}^{T} y_{t,u_t}$$

where $y_{i,j}$ is the output at position i and label j and the number of rows of $\boldsymbol{Y}$ (i.e., the number of positions) T equals $|\boldsymbol{u}|$. The plain maximum likelihood estimation requires position-wise labeled training data. Section 4.4 introduces an approach how to apply maximum likelihood without explicit alignment which is a typical scenario for speech recognition or HTR.

Since we exclusively deal with probabilities given by the family defined by the network, we omit the parameters $\boldsymbol{W}$ below and abbreviate $\mathrm{P}(\boldsymbol{u} \mid \boldsymbol{X}; \boldsymbol{W})$ by $\mathrm{P}(\boldsymbol{u} \mid \boldsymbol{X})$.

4.4 Sequence Labeling by Connectionist Temporal Classification

This section introduces the Connectionist Temporal Classification (CTC) training scheme for Neural Networks and some basic aspects of its decoding. The results of this section are well known in the literature but we reformulate them as an optimization problem on Weighted Automata. Here, terms are introduced which are fundamental for the rest of this thesis. We mainly follow the notation of Graves et al. [2006].

Some of the currently most successful HTR systems were trained with CTC as shown in several competitions. To give just one example, one of the most challenging real world tasks is the Maurdor project which was won by A2iA in 2014 using CTC to train LSTM-Cell RNNs (see Moysset et al. [2014]). CTC is not limited to text recognition. Recently the performance of several speech recognition systems trained with CTC equaled those of other state-of-the-art methods (e.g. Graves and Jaitly [2014], Sak et al. [2015]).

Definition 4.21 (not-a-character symbol, extended alphabet). Let $\mathbb{A}$ be the alphabet. Let $\oslash \notin \mathbb{A}$ denote an artificial garbage label called *not-a-character (NaC)* (also denoted as blank). Then, the *extended alphabet* is $\mathbb{A}' := \mathbb{A} \cup \{\oslash\}$. An element

of $\mathbb{A}$ is called *character*. Sequences from $\mathbb{A}^*$ are called *words*. Elements of $\mathbb{A}'$ are *labels*.

The NNs described below predict likelihoods of the labels from $\mathbb{A}'$ at any position for a specific input $\boldsymbol{X}$. The NaC has two functionalities: It indicates that none of the labels from $\mathbb{A}$ is present and it separates consecutive identical letters as we will see below.

As already mentioned, we assume a Neural Network with Softmax activation in the last layer such that the output $\boldsymbol{Y} \in \bigcup_{T=1}^{\infty}[0,1]^{T\times|\mathbb{A}'|}$ is a ConfMat which satisfies the laws of probability at each position t. That means, for any t, $\sum_{\ell\in\mathbb{A}'} y_{t,\ell} = 1$ and $y_{t,\ell} \geq 0$ for any t and any $\ell \in \mathbb{A}'$. Thus, $y_{t,\ell}$ is interpreted as the probability of label ℓ at time t. The probability of the label sequence $\boldsymbol{\pi} \in (\mathbb{A}')^*$ is simply the product of the individual probabilities:

$$\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X}) := \begin{cases} \prod_{t=1}^{T} y_{t,\pi_t} & \text{if } |\boldsymbol{\pi}| = T, \\ 0 & \text{otherwise} \end{cases}$$

which assumes conditional independence of any two labels $\ell_1, \ell_2 \in \mathbb{A}'$ at distinct time steps t_1 and t_2.

If the Automaton A accepts any feasible label sequence, the corresponding Weighted Automaton defined in the next lemma yields the reward $\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ for any $\boldsymbol{\pi} \in (\mathbb{A}')^T$ if and only if $\boldsymbol{\pi}$ is accepted by A and 0 otherwise.

Lemma 4.22. For any NFA $A = (Q, \mathbb{A}', \delta, q_0, F)$, let $W_{\boldsymbol{X}}(A)$ denote the WA $W_{\boldsymbol{X}}(A) := (Q, \mathbb{A}', \lambda, q_0, F)$ with

$$\lambda(q, \ell, q', t) := \begin{cases} y_{t,\ell} & \text{if } q' \in \delta(q,\ell) \wedge t \leq T, \\ 0 & \text{else} \end{cases}$$

for $\ell \in \mathbb{A}'$. Then for $|\boldsymbol{\pi}| \leq T$,

$$\max_{q\in F} \lambda^{\max}(q_0, \boldsymbol{\pi}, q, 1) = \begin{cases} \prod_{t=1}^{|\boldsymbol{\pi}|} y_{t,\pi_t} & \text{if } \delta^*(q_0,\boldsymbol{\pi}) \cap F \neq \emptyset, \\ 0 & \text{else.} \end{cases}$$

Proof. Let $\boldsymbol{\pi} \in (\mathbb{A}')^*$ be accepted by A and $|\boldsymbol{\pi}| \leq T$. Then there is a sequence $\boldsymbol{p}_{0:|\boldsymbol{\pi}|}$ in Q s.t. $p_0 = q_0$, $p_{|\boldsymbol{\pi}|} \in F$, for each i, $p_i \in \delta(p_{i-1}, \pi_i)$ and $\lambda^{\max}(p_0, \boldsymbol{\pi}, p_{|\boldsymbol{\pi}|}, 1) = \prod_{t=1}^{|\boldsymbol{\pi}|} \lambda(p_{t-1}, \pi_t, p_t, t) = \prod_{t=1}^{|\boldsymbol{\pi}|} y_{t,\pi_t}$.

For any $\boldsymbol{\pi} \in (\mathbb{A}')^*$ with $|\boldsymbol{\pi}| > T$ or which is not accepted by A, any state sequence $\boldsymbol{p}_{0:|\boldsymbol{\pi}|}$ with $p_0 = q_0$, $p_{|\boldsymbol{\pi}|} \in F$ and $p_i \in \delta(p_{i-1}, \pi_i)$ contains at least one p_t with $\lambda(p_{t-1}, \pi_t, p_t, t) = 0$ (since either $\delta(p_t, \pi_t) = \emptyset$ or $t > T$) such that $\lambda^{\max}(p_0, \boldsymbol{\pi}, p_{|\boldsymbol{\pi}|}, 1) = \prod_{t=1}^{|\boldsymbol{\pi}|} \lambda(p_{t-1}, \pi_t, p_t, t) = 0$. □

This means, Algorithm 3 applied to $W_{\boldsymbol{X}}(A)$ calculates the likelihood $\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ of any $\boldsymbol{\pi} \in (\mathbb{A}')^T$ accepted by A. Lemma 4.22 will be the standard procedure to create a WA from any Automaton which accepts the specific feasible label sequences. The most likely label sequence for any $\boldsymbol{z} \in \mathbb{A}^*$ is called *best path ($\boldsymbol{\beta}$)*.

To map a label sequence $\boldsymbol{\pi}$ to a word $\boldsymbol{z}$, one merges consecutive identical π_t and deletes the NaCs. More precisely, we define

Definition 4.23 (Collapse function ($\mathcal{F}$)). For any $\boldsymbol{\pi} \in (\mathbb{A}')^*$, let $\{s_1, \ldots, s_n\}$ be the largest set of indices where $\pi_{s_i} \notin \{\oslash, \pi_{s_i-1}\}$. W.l.g. let $s_i < s_{i+1}$ for any i. Then $\mathcal{F}(\boldsymbol{\pi}) := (\pi_{s_1}, \ldots, \pi_{s_n})$.

Definition 4.24 (Extended word). For any $\boldsymbol{z} \in \mathbb{A}^*$, let $\boldsymbol{z}' \in (\mathbb{A}')^*$ be the *extended word* which has an additional NaC before $\boldsymbol{z}$, after $\boldsymbol{z}$ and between each pair of characters.

Example 4.25. For example, $\mathcal{F}(\oslash aa \oslash aabb \oslash) = aab$ with the corresponding indices $(s_1, s_2, s_3) = (2, 5, 7)$. The corresponding extended word is $\oslash a \oslash a \oslash b \oslash$.

Definition 4.26 ($T_{\mathcal{F}}$). Let $T_{\mathcal{F}} = (Q, \mathbb{A}', \mathbb{A} \cup \{\varepsilon\}, \delta, q_\oslash, Q)$ be an FST with $Q := \{q_\ell \mid \ell \in \mathbb{A}'\}$ and

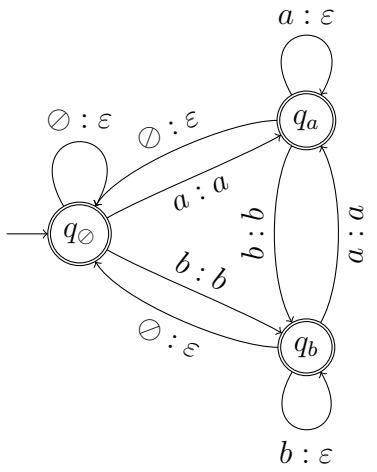
$$\delta(q, \ell) := \begin{cases} \{(q_\ell, \ell)\} & \text{if } \ell \in \mathbb{A} \wedge q \neq q_\ell, \\ \{(q_\ell, \varepsilon)\} & \text{if } \ell \in \mathbb{A} \wedge q = q_\ell, \\ \{(q_\oslash, \varepsilon)\} & \text{if } \ell = \oslash. \end{cases}$$

The states of $T_{\mathcal{F}}$ correspond to the labels $\mathbb{A}'$. The initial state is the "NaC-state" ($q_\oslash$). Any state is final. Reading any character $a \in \mathbb{A}$ for the first time yields an output a and forces a transition to q_a. A consecutive identical read of a generates no output and $T_{\mathcal{F}}$ stays in its current state. Also, reading a NaC generates no output. Thus, $T_{\mathcal{F}}$ defines the relation $\{(\boldsymbol{\pi}, \mathcal{F}(\boldsymbol{\pi})) \mid \boldsymbol{\pi} \in (\mathbb{A}')^*\}$. Figure 4.6(a) shows $T_{\mathcal{F}}$ for the alphabet $\mathbb{A} = \{a, b\}$. The Automaton given in the next lemma is a composition of $T_{\mathcal{F}}$ and the ε-free Automaton which accepts $\boldsymbol{z} \in \mathbb{A}^*$.

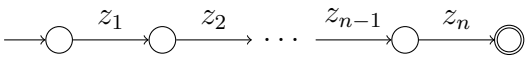
Lemma 4.27. For any $\boldsymbol{z} \in \mathbb{A}^*$, let $\mathring{\mathrm{A}}_{\boldsymbol{z}} = (Q, \mathbb{A}', \delta, q_1, F)$ be an Automaton with $Q := \{q_1, \ldots, q_{|\boldsymbol{z}'|}\}$ and $F := \{q_{|\boldsymbol{z}'|-1}, q_{|\boldsymbol{z}'|}\}$. The accepted language of $\mathring{\mathrm{A}}_{\boldsymbol{z}}$ is $\mathcal{F}^{-1}(\boldsymbol{z})$ if for any $q \in Q$

$$\delta(q_i, a) = \begin{cases} \{q_i\} & \text{if } z'_i = a \\ \{q_{i+1}\} & \text{if } i < |\boldsymbol{z}'| \wedge z'_{i+1} = a \\ \{q_{i+2}\} & \text{if } i < |\boldsymbol{z}'| - 1 \wedge z'_{i+2} = a \neq z'_i \\ \emptyset & \text{else} \end{cases}.$$

Proof. Let $n := |\boldsymbol{z}|$. Let $\{\hat{q}_0, \hat{q}_1, \ldots, \hat{q}_n\}$ be the states of the ε-free Automaton $A_{\boldsymbol{z}}$ from Figure 4.6(b) where q_0 is the initial state and the transition from q_{i-1} to q_i reads z_i. Let $\{\bar{q}_\ell \mid \ell \in \mathbb{A}'\}$ be the states of $T_{\mathcal{F}}$. Then $\mathring{\mathrm{A}}_{\boldsymbol{z}}$ results directly from



(a) FST $T_\mathcal{F}$ modeling the releation $\{(\boldsymbol{\pi}, \mathcal{F}(\boldsymbol{\pi})) \mid \boldsymbol{\pi} \in (\mathbb{A}')^*\}$ for $\mathbb{A} = \{a, b\}$.



(b) $A_{\boldsymbol{z}}$ accepting $\boldsymbol{z} \in \mathbb{A}^n$.

Figure 4.6: FST $T_\mathcal{F}$ and Automaton $A_{\boldsymbol{z}}$.

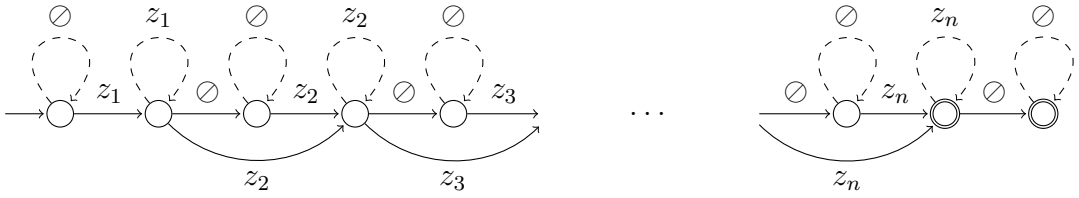


Figure 4.7: Automaton $\mathring{\mathrm{A}}_{\boldsymbol{z}}$ accepting $\mathcal{F}^{-1}(\boldsymbol{z})$ for single words $\boldsymbol{z}$. Dashed arcs indicate a repetition of labels.

the composition $T_\mathcal{F} \circ A_{\boldsymbol{z}}$ (see Algorithm 2) with $q_1 = (\bar{q}_\oslash, \hat{q}_0)$, $q_{2k} = (\bar{q}_{z_k}, \hat{q}_k)$ and $q_{2k+1} = (\bar{q}_\oslash, \hat{q}_k)$ (for $k \in \{1, \ldots, n\}$). This is a straight forward procedure: $T_\mathcal{F}$ reads $\ell \in \mathbb{A}'$ and outputs either ℓ (only if $\ell \in \mathbb{A}$) or ε. The result is read by $A_{\boldsymbol{z}}$. $A_{\boldsymbol{z}}$ remains in its current state by reading ε or moves one state forward by reading ℓ. Exemplarily, we show that there is no connection between $q_{2k} = (\bar{q}_{z_k}, \hat{q}_k)$ and $q_{2k+2} = (\bar{q}_{z_{k+1}}, \hat{q}_{k+1})$ if $z_k = z_{k+1}$. The reason is that $\{(\bar{q}_{z_k}, \varepsilon)\} = \delta_\mathcal{F}(\bar{q}_{z_k}, z_{k+1})$ where $\delta_\mathcal{F}$ is the transition function of $T_\mathcal{F}$. Since $A_{\boldsymbol{z}}$ has no ε-transitions, $A_{\boldsymbol{z}}$ remains in $\hat{q}_k$ by reading ε. Thus, the only transition from $(\bar{q}_{z_k}, \hat{q}_k)$ by reading $z_k = z_{k+1}$ is the loop. □

A visual representation of the Automaton is given in Figure 4.7. It shows the allowed transitions and which labels they read. Loops are dashed to indicate a repetition of the previous label. For the moment there is no difference between dashed and solid

arcs. The dashed arcs will have a special role in the Weighted Automaton when different labels reach the same state as shown in Chapter 6 below. Note that the states are numbered according to the index of the accepted label in $\boldsymbol{z}'$. This means q_i is reached from q_1 only by $\boldsymbol{\pi} \in (\mathbb{A}')^*$ with $\mathcal{F}(\boldsymbol{\pi}) = \mathcal{F}(\boldsymbol{z}'_{1:i})$ and ends on $\pi_{|\boldsymbol{\pi}|} = z'_i$. Obviously, q_j is reachable from q_i iff $i < j$.

Furthermore, the $\mathring{\mathrm{A}}_{\boldsymbol{z}}$ is obviously a DFA. Recall that according to Lemma 2.20, $\lambda^{\max}(q, \boldsymbol{\pi}, q', t) = \lambda^{+}(q, \boldsymbol{\pi}, q', t)$ for WA whose support is a DFA. Then the maximum and the total reward of the WA $W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})$ are equal: $\rho^{\max}_{W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})}(\boldsymbol{\pi}) = \rho^{+}_{W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})}(\boldsymbol{\pi})$ and yield $\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X}) = \prod_{t=1}^{T} y_{t,\pi_t}$ for any label sequence $\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z})$ with $|\boldsymbol{\pi}| = T$. The rewards of any other label sequence are zero.

Since $\sum_{\ell \in \mathbb{A}'} y_{t,\ell} = 1$ for any t,

$$\sum_{\boldsymbol{\pi} \in (\mathbb{A}')^T} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X}) = \sum_{\boldsymbol{\pi} \in (\mathbb{A}')^T} \prod_{t=1}^{T} y_{t,\pi_t} = \prod_{t=1}^{T} \sum_{\ell \in \mathbb{A}'} y_{t,\ell} = 1. \tag{4.4.1}$$

Thus, $\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ yields a probability distribution of $(\mathbb{A}')^T$.

4.4.1 Training

The naïve approach to train a NN by the maximum likelihood estimation is to maximize $\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ for a precise labeling $\boldsymbol{\pi} \in (\mathbb{A}')^*$. Unfortunately, the training set typically contains only the transcription $\boldsymbol{z} \in \mathbb{A}^*$ of an image $\boldsymbol{X}$ of a writing but not position-wise labeling $\boldsymbol{\pi}$. Analogously to the Baum-Welch-Training, CTC maximizes the sum over all the label sequences collapsing to $\boldsymbol{z}$ instead:

$$\mathcal{E}(\mathbb{D}, \boldsymbol{W}) := \sum_{(\boldsymbol{X},\boldsymbol{z}) \in \mathbb{D}} \mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X}) := \sum_{(\boldsymbol{X},\boldsymbol{z}) \in \mathbb{D}} \sum_{\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X}) \to \max. \tag{4.4.2}$$

Taking the logarithm of the maximum likelihood error function of Eq. (4.4.2) is a typical way to convert the product into a sum and keeping the maxima: Let $\mathbb{D}$ be the data set, then we take the logarithm of the original error function such that $\ln \mathcal{E}(\mathbb{D}, \boldsymbol{W}) = \sum_{(\boldsymbol{X},\boldsymbol{z})} \ln \mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$. Now the gradient can be calculated for each sample $(\boldsymbol{X}, \boldsymbol{z}) \in \mathbb{D}$, individually. Then, $\ln \mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$ is the contribution of the sample $(\boldsymbol{X}, \boldsymbol{z})$ to the logarithmic error with the derivative:

$$\frac{\partial \ln \mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})}{\partial w_{i,j}} = \sum_{t,\ell} \frac{\partial \ln \mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})}{\partial y_{t,\ell}} \frac{\partial y_{t,\ell}}{\partial w_{i,j}}.$$

It is well known how to calculate $\frac{\partial y_{t,\ell}}{\partial w_{i,j}}$ (e.g. by Backpropagation Through Time for RNNs). The first term $\frac{\partial \ln \mathrm{P}}{\partial y_{t,\ell}}$ is given by the following theorem:

Theorem 4.28. For any $\boldsymbol{z} \in \mathbb{A}^*$, let $\mathring{\mathrm{A}}_{\boldsymbol{z}}$ be the Automaton from Lemma 4.27 and

$W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})$ be the corresponding WA as in Lemma 4.22. Then with α^+ and β^+ as in Lemma 3.2 ($\mathcal{X}_i = \mathbb{A}'$),

$$\frac{\partial \ln \mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})}{\partial y_{t,\ell}} = \frac{1}{\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})} \sum_{\substack{i=1 \\ z'_i=\ell}}^{|\boldsymbol{z}'|} \frac{\alpha_t^+(q_i)\beta_t^+(q_i)}{y_{t,\ell}}.$$

Proof.

$$\begin{aligned}
\frac{\partial \ln \mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})}{\partial y_{t,\ell}} &= \frac{1}{\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})} \sum_{\substack{\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z}) \\ \pi_t = \ell \\ |\boldsymbol{\pi}| = T}} \prod_{\substack{t'=1 \\ t' \neq t}}^{T} y_{t',\pi_{t'}} \\
&= \frac{1}{\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})} \sum_{\substack{\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z}) \\ \pi_t = \ell \\ |\boldsymbol{\pi}| = T}} \sum_{q \in F} \frac{\lambda^+(q_0, \boldsymbol{w}, q, 1)}{y_{t,\ell}} \\
&= \frac{1}{\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})} \sum_{\substack{\boldsymbol{\pi} \in (\mathbb{A}')^T \\ \pi_t = \ell}} \sum_{q \in F} \frac{\lambda^+(q_0, \boldsymbol{w}, q, 1)}{y_{t,\ell}} \\
&= \frac{1}{\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})} \sum_{\substack{i=1 \\ z'_i=\ell}}^{|\boldsymbol{z}'|} \left(\sum_{\bar{\boldsymbol{\pi}} \in (\mathbb{A}')^t} \frac{\lambda^+(q_0, \bar{\boldsymbol{\pi}}, q_i, 1)}{y_{t,\ell}} \right) \left(\sum_{\hat{\boldsymbol{\pi}} \in \mathbb{A}^{T-t}} \sum_{q' \in F} \lambda^+(q_i, \hat{\boldsymbol{\pi}}, q', t) \right) \\
&= \frac{1}{\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})} \sum_{\substack{i=1 \\ z'_i=\ell}}^{|\boldsymbol{z}'|} \frac{\alpha_t^+(q_i)\beta^+(q_i)}{y_{t,\ell}}
\end{aligned}$$

where $\mathbb{A}^0 := \{\varepsilon\}$ contains only the empty word for which $\lambda^+(q, \varepsilon, q', t) = 1$ if $q = q'$ and 0 else. □

Hence, $\frac{\partial \ln \mathrm{P}}{\partial y_{t,l}}$ can be calculated efficiently using the intermediate steps α^+ and β^+ from Algorithm 4. A standard Backpropagation algorithm propagates errors into the network and optimizes its parameters. The original description without Automata can be found in Graves et al. [2006].

4.4.2 Decoding

To decode the label at position t which is most likely encoded, it is obvious to take the label π_t^* with the greatest output confidence. If we decode for each position $\beta_t = \arg\max_{\ell \in \mathbb{A}'} y_{t,\ell}$ and collapse the best path to $\boldsymbol{z}^* = \mathcal{F}(\boldsymbol{\beta})$, this is called *best path decoding*. This obviously may lead to infeasible results since typically only a sparse subset of $\mathbb{A}^*$ is feasible.

Due to Lemma 4.22, we know that for any specific $\boldsymbol{z} \in \mathbb{A}^*$ the WA $W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})$ accepts any label sequence $\boldsymbol{\pi}$ in $\mathcal{F}^{-1}(\boldsymbol{z})$ with probability $\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ and any other sequence with 0 probability. Thus, Algorithm 3 calculates

$$\mathrm{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}) := \max_{\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$$

for $\mathcal{X}_i = \mathbb{A}'$ for each i. This approach returns the likelihood of the optimal label (or state sequence, equivalently) sequence which collapses to $\boldsymbol{z}$. Thus, this corresponds to the Viterbi approximation of HMMs.

However, it is also very natural to decode the character sequence with the highest training objective value: $\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X}) = \sum_{\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ which can be calculated analogously using Algorithm 4 ($\mathcal{X}_i = \mathbb{A}'$ for each i). Note, that the corresponding Automaton has to be a DFA in order to sum any label sequence only once.

Definition 4.29. For any $\boldsymbol{z} \in \mathbb{A}^*$ and any input $\boldsymbol{X}$, we call $\mathrm{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$ the *path probability* and $\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$ is called the *CTC (word) probability*.

The path probability yields an alignment of positions and class labels, it speeds up the calculation and – since there is typically one dominant label sequence – it is a reasonable approximation to $\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$ as we will see in the next chapter.

4.4.3 Decoding with Language Models

Up to now, we composed $T_{\mathcal{F}}$ and the $A_{\boldsymbol{z}}$ which accepts a word $\boldsymbol{z} \in \mathbb{A}^*$ to accept any feasible label sequence collapsing to $\boldsymbol{z}$. This corresponds to the word level decoding. Let $A_{\mathcal{V}}$ be the Automaton which is composed of each $A_{\boldsymbol{z}}$ where $\boldsymbol{z} \in \mathcal{V}$ by sharing the initial state (such as in Figure 2.3(c)). Let $T_{\mathcal{V}}$ be an FST of the Kleene closure (Figure 2.1(c) and ε-closure subsequently) of $A_{\mathcal{V}}$ which outputs the word when reaching the corresponding final state and the empty word ε for other transitions. Thus, $T_{\mathcal{V}}$ relates a character sequence from $\mathbb{A}^*$ to the corresponding word sequence from $\mathcal{V}^*$. Thus, $T_{\mathcal{F}} \circ T_{\mathcal{V}}$ is sufficient to model label sequences collapsing to feasible word sequences.

Consider a Language Model which assumes the Markov property with a history of $n-1$ words. A WA which models such a n-gram must be able to "remember" the last $n-1$ words while reading the next word:

Definition 4.30. Let $A_{\mathrm{LM}} = (Q, \mathbb{A}, \delta, q_0, F)$ where $Q = \bigcup_{i=0}^{n-1} \mathcal{V}^i$, $F = Q$, $q_0 = \varepsilon$ and for any $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m) \in Q$ and $\boldsymbol{z} \in \mathcal{V}$

$$\delta\Big((\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m), \boldsymbol{z}\Big) = \begin{cases} \{(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m, \boldsymbol{z})\} & \text{if } m < n-1, \\ \{(\boldsymbol{v}_2, \ldots, \boldsymbol{v}_m, \boldsymbol{z})\} & \text{otherwise.} \end{cases}$$

The composition $\text{Å}_{\text{LM}} := T_{\mathcal{F}} \circ T_{\mathcal{V}} \circ A_{\text{LM}}$ still accepts any label sequence which collapses to a sequence of feasible words. At the same time, any state corresponds to a history of the last $n-1$ words such that a LM can be applied.

At this point, the theory of Weighted FSTs yields an elegant way to integrate also the weights / rewards into the composition (see Mohri [2009]). Instead of introducing yet another theory, we define the rewards "manually": The Weighted Automaton $W_{\boldsymbol{X}}(\text{Å}_{\text{LM}})$ only makes use of the NN output $y_{t,\ell}$ for the specific label ℓ at position t such that Algorithm 4 on $W_{\boldsymbol{X}}(\text{Å}_{\text{LM}})$ yields $\max_{\boldsymbol{z}_{1:n} \in \mathcal{V}^*} \mathrm{P}(\boldsymbol{z}_{1:n} \mid \boldsymbol{X})$.

To integrate also LM probabilities, Graves et al. [2008] suggested a reestimation of the likelihood $\mathrm{P}(\boldsymbol{z}_{1:n} \mid \boldsymbol{X})$ given by the NN using the prior probability $\mathrm{P}(\boldsymbol{z}_{1:n})$ defined by the Language Model:

$$\mathrm{P}(\boldsymbol{z}_{1:n} \mid \boldsymbol{X}) \, \mathrm{P}(\boldsymbol{z}_{1:n}) \tag{4.4.3}$$

for any word sequence $\boldsymbol{z}_{1:n} \in \mathcal{V}^n$. They obtained this "score" as a consequence of Bayes' Law and as an omission of terms which are constant for all label sequences by assuming that $\boldsymbol{X}$ and the Language Model are independent. They mentioned that this is obviously not true and thus the score is an approximation. We will derive a generalization of the above formula in Section 7.1.

The score of Eq. (4.4.3) can easily be integrated into the Weighted Automaton: The states of Å_{LM} have the form $(q, \bar{q}, \hat{q})$ where q is a state of $T_{\mathcal{F}}$, $\bar{q}$ is a state of $T_{\mathcal{V}}$ and $\hat{q}$ is a state of T_{LM}. We obviously reached a new word through the transition from $r_1 := (q_1, \bar{q}_1, \hat{q}_1)$ to $r_2 := (q_2, \bar{q}_2, \hat{q}_2)$ if

- $(q_1, \bar{q}_1) \neq (q_2, \bar{q}_2)$,
- $\bar{q}_2$ corresponds to the first character of any word and
- $q_2 \neq q_{\oslash}$.

The latter two points ensure that $(q_2, \bar{q}_2)$ is a first character state of any word in $T_{\mathcal{F}} \circ T_{\mathcal{V}}$ and the first item states that it is not the same state as before. Only for these transitions, we replace the rewards $\lambda(r_1, \ell, r_2, t) := y_{t,\ell}$ of $W_{\boldsymbol{X}}(\text{Å}_{\text{LM}})$ by $\lambda(r_1, \ell, r_2, t) := y_{t,\ell} \, \mathrm{P}(\boldsymbol{z}_i \mid \boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1})$ if $\hat{q}_1$ represents the history $\boldsymbol{z}_{i-n+1}, \ldots, \boldsymbol{z}_{i-1}$ and the transition $\hat{q}_1$ to $\hat{q}_2$ reads $\boldsymbol{z}_i$.

The resulting Automaton reads any accepted label sequence only once (since $(q, \bar{q}, \hat{q})$ corresponds to a unique label of a unique word with a unique history) and thus Algorithm 4 calculates

$$\max_{\boldsymbol{z}_{1:n} \in \mathcal{V}^*} \mathrm{P}(\boldsymbol{z}_{1:n} \mid \boldsymbol{X}) \, \mathrm{P}(\boldsymbol{z}_{1:n}).$$

Thus, it is equivalent to the *CTC-Token-Passing Algorithm* proposed in Graves et al. [2008] if $\mathrm{P}(\boldsymbol{z}_{1:n})$ is given by a bigram. The authors gave an explicit Dynamic Programming scheme to solve the above optimization problem.

4.5 Conclusion

In this chapter, we introduce the state-of-the-art sequence labeling methods: HMMs and Neural Networks. First, the basic concepts of Language Models are presented since they are an essential part of the decoding process of the sequence labeling methods in case of HTR. The Hidden Markov Models yet provide the fundamental ideas of CTC which is a discriminative version of the Forward-Backward Algorithm. Thus, also the decoding follows similar approaches: It can be split up into different layers of abstraction each of them represented by an FST which are composed to obtain a Automaton which models all the feasible label sequences. Training and decoding of HMMs and NNs can be reduced to the same two algorithms on WA.

Besides these decoding concepts, we also define the basic architectures of HMMs and Neural Networks which will be used in the experiments hereafter. Furthermore, it is shown that these Neural Networks theoretically are able to reproduce any conditional probability which is the fundamental justification to model conditional probabilities with Neural Networks.

5 Single-Word Decoding

The aim of this part is to explore elementary decoding methods for single words, i.e., the result of the decoding is only a single word. Typically, we also assume that the input writing to the Neural Network contains only a single word. However, we will also introduce a method to search effectively for a single word within a whole line of text. The results of the current section are partially published in Strauß et al. [2012, 2013]. Section 5.2 contains a result of Frinken et al. [2010]. Again, the previous results are reformulated as optimization problems on Weighted Graphs.

We start with an investigation of the objective functions with respect to the decoding error they produce.

5.1 Objective Functions

In this section, we compare different "confidence functions" which result in different decoding times and performance.

5.1.1 Probabilistic Objectives

Already in Section 4.4.3, we introduced the path probability

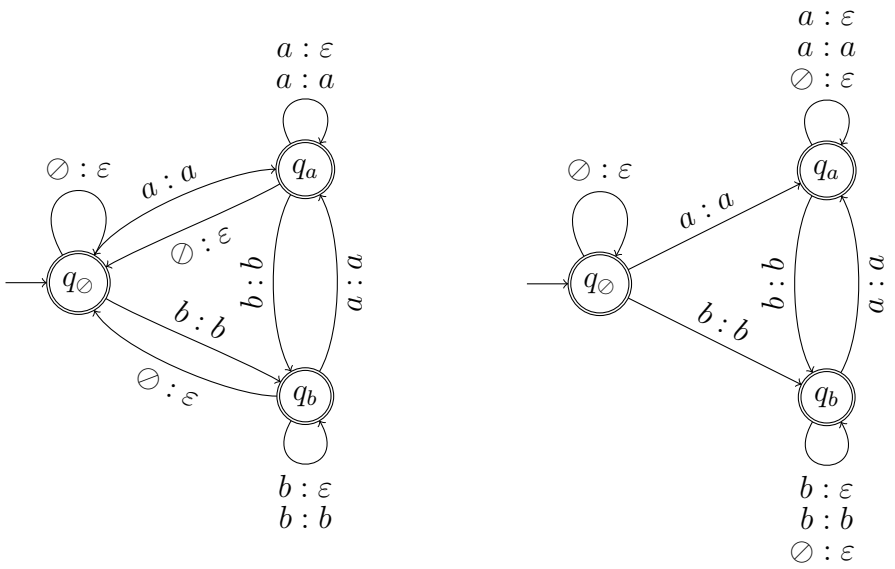
$$\mathrm{O}_{\mathcal{F}}(\boldsymbol{Y}, \boldsymbol{z}) := \mathrm{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}) = \max_{\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$$

and CTC probability

$$\mathrm{O}_{\mathrm{CTC}}(\boldsymbol{Y}, \boldsymbol{z}) := \mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X}) = \sum_{\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$$

as possible objectives of the optimization problem (OP). In the following, we will modify the objectives and constraints.

All label sequences which contribute to $\mathrm{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$ and $\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$ are elements of $\mathcal{F}^{-1}(\boldsymbol{z})$. According to $\mathcal{F}$, the NaC is necessary and sufficient to separate consecutive identical letters. In case of high subsampling rate, there may be not enough ConfMat positions to separate consecutive identical letters of the ground truth by a NaC. Hence, the objectives will yield zero probability for the true label sequence. To



(a) $T_{\mathcal{G}}$. (b) $T_{\mathcal{H}}$.

Figure 5.1: Finite State Transducers $T_{\mathcal{G}}$ and $T_{\mathcal{H}}$ modeling the releation $\{(\boldsymbol{\pi}, \boldsymbol{z}) \in (\mathbb{A}')^* \times \mathbb{A}^* \mid \boldsymbol{z} \in \mathcal{G}(\boldsymbol{\pi})\}$ and $\{(\boldsymbol{\pi}, \boldsymbol{z}) \in (\mathbb{A}')^* \times \mathbb{A}^* \mid \boldsymbol{z} \in \mathcal{H}(\boldsymbol{\pi})\}$, respectively, for $\mathbb{A} = \{a, b\}$.

circumvent this problem, we define a new collapse function $\mathcal{G}$. Since the claims are very similar to those of the Section 4.4, we omit the proofs.

For sake of convenience, we write $\Phi^{-1}(\boldsymbol{b}) := \{\boldsymbol{a} \in A \mid \boldsymbol{b} \in \Phi(\boldsymbol{a})\}$ for any function $\Phi : A \to \mathcal{P}(B)$.

Definition 5.1. For any $\boldsymbol{\pi} \in (\mathbb{A}')^*$, let $\{s_1, \ldots, s_n\}$ be a set of indices which contains at least any i with $\oslash \neq \pi_i \neq \pi_{i-1}$ and which does not contain indices j with $\oslash = \pi_j$. W.l.g. let $s_i < s_{i+1}$ for any i. Then $\mathcal{G} : (\mathbb{A}')^* \to \mathcal{P}(\mathbb{A}^*)$ where $\mathcal{G}(\boldsymbol{\pi})$ contains $(\pi_{s_1}, \ldots, \pi_{s_n})$ iff $(s_1, \ldots, s_n)$ satisfies the above conditions.

Loosely spoken: $\mathcal{G}$ may or may not delete some (but not all) of the consecutive identical labels from $\boldsymbol{\pi}$ and delete all NaCs. For example,

$$\mathcal{G}(\oslash aa \oslash aabb \oslash) = \{aaaabb, aaabb, aabb, aaaab, aaab, aab\}.$$

Thus, NaCs are sufficient but not necessary to separate letters. The corresponding FST $T_{\mathcal{G}}$ has additional transitions compared to $T_{\mathcal{F}}$: $(q_a, a) \in \delta(q_a, a)$ for any $a \in \mathbb{A}$. Figure 5.1(a) shows $T_{\mathcal{G}}$ for a binary alphabet.

Definition 5.2. Analogously to Lemma 4.27, let $\mathring{\mathrm{A}}_{\boldsymbol{z}}^{\mathcal{G}}$ be the composition of $T_{\mathcal{G}} \circ A_{\boldsymbol{z}}$.

$\mathring{\mathrm{A}}_{\boldsymbol{z}}^{\mathcal{G}}$ differs from $\mathring{\mathrm{A}}_{\boldsymbol{z}}$ (see Lemma 4.27) only in one type of transition: Consecutive identical labels (not the NaC) may change the state such that they are interpreted

as different letters since the loop of q_a in $T_{\mathcal{G}}$ can output a or ε while reading $a \in \mathbb{A}$. Thus, $\mathring{\mathrm{A}}_{\boldsymbol{z}}^{\mathcal{G}}$ is not a DFA anymore if $\boldsymbol{z}$ contains consecutive identical letters.

If NaCs are ignored in the way that they do not change the state, we can even save the NaC-states which will reduce the running time:

Definition 5.3. For any $\boldsymbol{\pi} \in (\mathbb{A}')^*$, let $S := \{s_1, \ldots, s_n\}$ be a set of indices which satisfies:

- S contains the minimum i with $\pi_i \in \mathbb{A}$,
- S contains the smallest index $i > s_j$ with $\pi_i \in \mathbb{A}$ and $\pi_i \neq \pi_{s_j}$ for any $s_j \in S$ and
- S does not contain any index j with $\oslash = \pi_j$.

W.l.g. let $s_i < s_{i+1}$ for any i. Then $\mathcal{H} : (\mathbb{A}')^* \to \mathcal{P}(\mathbb{A}^*)$ where $\mathcal{H}(\boldsymbol{z})$ contains $(\pi_{s_1}, \ldots, \pi_{s_n})$ iff $(s_1, \ldots, s_n)$ satisfies the above conditions.

Loosely spoken: $\mathcal{H}$ first deletes all NaCs and afterwards it may or may not delete consecutive identical labels from $\boldsymbol{\pi}$. For example,

$$\mathcal{H}(\oslash aa \oslash aabb \oslash) = \{aaaabb, aaabb, aabb, abb, aaaab, aaab, aab, ab\}.$$

Thus, NaCs are neither sufficient nor necessary to separate letters. In other words: A NaC has no influence on the number of letters which are decoded. Thus, we could save some states:

Definition 5.4. Let $\mathring{\mathrm{A}}_{\boldsymbol{z}}^{\mathcal{H}}$ be the the composition of $T_{\mathcal{H}} \circ A_{\boldsymbol{z}}$.

Once $T_{\mathcal{H}}$ reaches a state q_a corresponding to a character $a \in \mathbb{A}$, it never returns to $q_{\oslash}$. Thus, $\mathring{\mathrm{A}}_{\boldsymbol{z}}^{\mathcal{H}}$ has only $|\boldsymbol{z}|$ states besides the initial state since any character of $\boldsymbol{z}$ corresponds to only one state in the composition. Once $\mathring{\mathrm{A}}_{\boldsymbol{z}}^{\mathcal{H}}$ is in the state corresponding to z_i, $\mathring{A}_{\boldsymbol{z}}^{\mathcal{H}}$ moves to the next state only if it reads z_{i+1}. Figure 5.1(b) shows $T_{\mathcal{H}}$ for a binary alphabet.

Let $\mathring{\mathrm{A}} := \mathring{\mathrm{A}}_{\boldsymbol{z}}^{\mathcal{G}}$ or $\mathring{\mathrm{A}} := \mathring{\mathrm{A}}_{\boldsymbol{z}}^{\mathcal{H}}$ be the Automaton accepting $\mathcal{G}^{-1}(\boldsymbol{z})$ or $\mathcal{H}^{-1}(\boldsymbol{z})$, respectively. Let further $W_{\boldsymbol{X}}(\mathring{\mathrm{A}})$ be the corresponding Weighted Automaton from Lemma 4.22. Then, Algorithm 3 applied to $W_{\boldsymbol{X}}(\mathring{\mathrm{A}})$ calculates the path probability

$$\mathrm{O}_{\mathcal{G}}(\boldsymbol{Y}, \boldsymbol{z}) := \max_{\boldsymbol{\pi} \in \mathcal{G}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$$

or

$$\mathrm{O}_{\mathcal{H}}(\boldsymbol{Y}, \boldsymbol{z}) := \max_{\boldsymbol{\pi} \in \mathcal{H}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X}),$$

respectively. However, for both mappings several words share some common label sequences such that there will be confusions if the decoded label sequence belongs to two distinct, feasible words. As a consequence, $\sum_{\boldsymbol{z} \in \mathbb{A}^*} \sum_{\boldsymbol{\pi} \in (\mathbb{A}')^*} \rho^{+}_{W_{\boldsymbol{X}}(A)}(\boldsymbol{\pi}) =$

$\sum_{\boldsymbol{z}\in\mathbb{A}^*}\sum_{\boldsymbol{\pi}} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ sums several label sequences multiple times which may yield a sum greater than 1 in contrast to Eq. (4.4.1). Thus Algorithm 4 is not meaningful for $W_{\boldsymbol{X}}(A)$.

5.1.2 Non-Probabilistic Objectives

The overall performance is measured in CER and WER which both are based on the Levenshtein Distance $\mathrm{d}(\boldsymbol{z}_1, \boldsymbol{z}_2)$ (see Section 3.3). Hence, it is very natural to measure the confidence that a word $\boldsymbol{z}$ is encoded in the ConfMat $\boldsymbol{Y}$ by the negative Levenshtein Distance between the collapsed best path $\mathcal{F}(\boldsymbol{\beta})$ and $\boldsymbol{z}$:

$$\mathrm{O}_{\mathrm{Lev}}(\boldsymbol{Y}, \boldsymbol{z}) := -\,\mathrm{d}(\mathcal{F}(\boldsymbol{\beta}), \boldsymbol{z}).$$

Compared to the other decoding measures, Levenshtein is extremely fast. It is also calculated by Dynamic Programming but the number of calculations depends on the length of the two character sequences which are compared. In contrast, the other measures compare label sequences with character sequences. Since there are typically much more ConfMat positions than encoded characters, the comparison of character sequences only is faster.

Definition 5.5 (Hamming Distance). Let $\boldsymbol{\pi}, \hat{\boldsymbol{\pi}} \in (\mathbb{A}')^T$ be label sequences. The *Hamming Distance* is the number of distinct components of $\boldsymbol{\pi}$ and $\hat{\boldsymbol{\pi}}$ at any position:

$$\mathrm{ham}(\boldsymbol{\pi}, \hat{\boldsymbol{\pi}}) := \sum_{t=1}^{T} 1 - \delta_{\pi_t, \hat{\pi}_t}$$

where $\delta_{i,j}$ is the Kronecker delta.

The objective function calculates the minimum Hamming Distance for any label sequence from $\mathcal{F}^{-1}(\boldsymbol{z})$ and the best path $\boldsymbol{\beta}$:

$$\mathrm{O}_{\mathrm{Ham}}(\boldsymbol{Y}, \boldsymbol{z}) := -\min_{\substack{\boldsymbol{\pi}\in\mathcal{F}^{-1}(\boldsymbol{z}) \\ |\boldsymbol{\pi}|=|\boldsymbol{\beta}|}} \mathrm{ham}(\boldsymbol{\beta}, \boldsymbol{\pi}).$$

In contrast to the probabilistic objective functions, the Hamming Distance is additive such that we have to define an additive reward:

Lemma 5.6. Let $\mathring{\mathrm{A}}_{\boldsymbol{z}} = (Q, \mathbb{A}', \delta, q_0, F)$ be the Automaton accepting any label sequence collapsing to $\boldsymbol{z}$. Let the Weighted Automaton be defined as $W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}}) := (Q, \mathbb{A}', \lambda, q_0, F)$ where

$$\lambda(q, a, \bar{q}, t) := \begin{cases} 0 & \text{if } a = z'_t \wedge \bar{q} \in \delta(q, a) \wedge t \leq T, \\ -1 & \text{else.} \end{cases}$$

Then for any $\bar{q} \in Q$, let

$$\hat{\alpha}_0(\bar{q}) := \begin{cases} 0 & \text{if } \bar{q} = q_0, \\ -\infty & \text{else,} \end{cases}$$

and for $t \geq 1$

$$\hat{\alpha}_t(\bar{q}) := \max_{q \in Q, a \in \mathbb{A}} \hat{\alpha}_{t-1}(q) + \lambda(q, a, \bar{q}, t).$$

Then for any $1 \leq t \leq T$, the reward of a path is the sum of the transitions:

$$\hat{\alpha}_t(q) = \max_{\boldsymbol{\pi} \in \mathbb{A}^t} \max_{\substack{p_0, \ldots, p_t \in Q \\ p_0 = q_0 \\ p_t = q}} \sum_{j=1}^{t} \lambda(p_{j-1}, \pi_j, p_j, j).$$

The proof is analogous to the proofs of Lemma 3.1 and 4.22.

A slightly modified version of Algorithm 3, which updates and initializes α according to Lemma 5.6, calculates the minimum Hamming Distance between the best path and any label sequence from $\mathcal{F}^{-1}(\boldsymbol{z})$ very efficiently.

5.1.3 Experimental Comparison of the Objective Functions

In this section, we compare the decoding methods on two datasets published at ICDAR in 2009. The entire software is written in JAVA. All experiments are executed on a laptop with an i7 2.8 GHz CPU and 8 GByte memory.

Neural Networks with Conservative Sampling Rates

Setup: We test on the IFN/ENIT database of handwritten Arabic words (see Pechwitz et al. [2002]) as well as on a subset of the RIMES database of handwritten mail snippets (see Grosicki and Abed [2009]) published at ICDAR 2009. We choose the vocabulary item with the best score:

$$\boldsymbol{z}^* := \arg\max_{\boldsymbol{z} \in \mathcal{V}} \mathrm{O}(\boldsymbol{Y}, \boldsymbol{z})$$

where $\mathrm{O}(\boldsymbol{Y}, \boldsymbol{z})$ is one of $\mathrm{O}_{\mathrm{Lev}}(\boldsymbol{Y}, \boldsymbol{z})$, $\mathrm{O}_{\mathrm{Ham}}(\boldsymbol{Y}, \boldsymbol{z})$, $\mathrm{O}_{\mathcal{H}}(\boldsymbol{Y}, \boldsymbol{z})$ or $\mathrm{O}_{\mathrm{CTC}}(\boldsymbol{Y}, \boldsymbol{z})$. Additionally, $\mathrm{O}_{\mathcal{F}}(\boldsymbol{Y}, \boldsymbol{z})$ and $\mathrm{O}_{\mathcal{G}}(\boldsymbol{Y}, \boldsymbol{z})$ are calculated for RIMES. The Neural Network consists of three recurrent MD-LSTM Layers and two Feed-Forward Layers. The architecture was provided by Graves (see [Graves, 2012, Section 9.2]). We omit a detailed description and refer to the cited work where the interested reader will find all important information. We train 10 randomly initialized Neural Networks on a training set. The tests are only performed on a disjoint test set. Note that we use

	O_{Lev}	O_{Ham}	$O_{\mathcal{H}}$	O_{CTC}
average error	18.32%	17.68%	11.22%	**10.77%**
minimum error	13.84%	15.73%	7.95%	7.62%
maximum error	35.71%	34.23%	23.92%	22.83%

Table 5.1: Word error rate on the IFN/ENIT dataset. The average, minimum and maximum error rates are calculated among 10 randomly initialized NNs.

		O_{Lev}	O_{Ham}	$O_{\mathcal{F}}$	$O_{\mathcal{G}}$	$O_{\mathcal{H}}$	O_{CTC}
$\mathcal{V}_{test}$	average error	19.84%	19.78%	11.06%	11.11%	11.86%	**10.80%**
	minimum error	17.05%	17.30%	9.36%	9.40%	10.04%	9.23%
	maximum error	21.97%	21.45%	12.61%	12.65%	13.48%	12.34%
$\mathcal{V}_{tot}$	average error	23.59%	23.86%	13.70%	13.74%	15.36%	**13.36%**
	minimum error	20.53%	21.28%	11.51%	11.31%	13.14%	11.34%
	maximum error	25.76%	25.64%	15.65%	15.66%	17.33%	15.30%

Table 5.2: Word error rate for the RIMES handwriting recognition tests. The average, minimum and maximum error rates are calculated among 10 randomly initialized NNs.

exactly the same input to all decoding methods.

Data: The IFN/ENIT database contains 32492 different handwritten names and zip codes of Tunisian places. Unlike the original task, we decode only the names of the places and do not incorporate the zip codes. We divide the data into a training set of 30,000 images and a test set containing 2492 items. The error measure is the WER. The vocabulary was created from all words occurring in the whole data set (union of training and test set). Thus, any word in the text lines is known which is usually referred to as *closed vocabulary*. It contains 1508 different words.

The RIMES database contains French handwritten faxes and postal mails. Only the data of the subtask is considered which targets at the recognition of snippets of handwritten words. We divide the data into a training set of 44,196 images and a test set which contains 7542 writings. Similarly to the original task, we test with two different vocabularies: The first one ($\mathcal{V}_{test}$) contains all elements from the test set (1636 words) and the second one ($\mathcal{V}_{tot}$) was created from all words occurring in the whole data set (4936 words). Hence, both are closed vocabularies.

Results: Table 5.1 and 5.2 show the average, minimum and maximum word error rates over the 10 Neural Networks on the test set. Levenshtein and Hamming

Distance yield the greatest error. Increasing the set of feasible sequences from $\mathcal{F}^{-1}$ to $\mathcal{G}^{-1}$ seems to have only slight impact on the error rate (Table 5.2, only). The WER for $\mathrm{O}_{\mathcal{H}}$ increases compared to $\mathrm{O}_{\mathcal{F}}$ or $\mathrm{O}_{\mathcal{G}}$. Calculating the objectives $\mathrm{O}_{\mathrm{CTC}}$, $\mathrm{O}_{\mathcal{F}}$, $\mathrm{O}_{\mathcal{G}}$ and $\mathrm{O}_{\mathrm{Ham}}$ needs about 30% more computation time than $\mathrm{O}_{\mathcal{H}}(\boldsymbol{z})$ since the number of values α which are calculated is about twice as big but due to some overhead we are only able to save 50% of that time. The $\mathrm{O}_{\mathrm{Lev}}$ is the objective with the lowest decoding time as expected which needs not even 10% of the time of $\mathrm{O}_{\mathrm{CTC}}$, $\mathrm{O}_{\mathcal{F}}$, $\mathrm{O}_{\mathcal{G}}$ and $\mathrm{O}_{\mathrm{Ham}}$.

Independently from the decoding scheme, the RIMES experiment shows the impact of a perfectly adapted vocabulary to the decoding process. The error of *CTC* with $\mathcal{V}_{tot}$ for example increases by 24% compared to $\mathcal{V}_{test}$.

Decoding Neural Networks with High Subsampling Rate

Setup: The decoding strategies introduced in Section 5.1.1 differ in the set of label sequences belonging to the same word. While $\mathcal{F}$ maps any $\boldsymbol{\pi} \in \mathbb{A}^*$ unambiguously to one word, $\mathcal{G}$ and $\mathcal{H}$ map $\boldsymbol{\pi}$ to a set of words which may result in confusions if this set contains two feasible words. The following experiment investigates the effect of "fuzzy" mapping in case of high subsampling rate.

We train and test on the same partitions of the RIMES data as before and average any result over 10 randomly initialized NNs. The vocabulary consists of 4936 words. The NN architecture is similar to that one used in Graves [2012] but with a higher subsampling rate (18 instead of 8). This decreases the time complexity by factor ≈ 2 which could be important in practical applications. Additionally, we compare 10 Neural Networks used in Leifert et al. [2013], again with the same subsampling rate.

Results: Table 5.3 shows the results of the experiments. Due to the high subsampling rate, it seems to be better to take some more label sequences into account: $\mathrm{O}_{\mathcal{G}}$ seems to be a better score than $\mathrm{O}_{\mathrm{CTC}}$ and $\mathrm{O}_{\mathcal{F}}$ for both RNNs. The latter decoding methods yield similar error rates for this tests. $\mathrm{O}_{\mathcal{H}}$ performs worst but also has a notably smaller running time.

If the subsampling rate is too high, the number of positions T could be smaller than the number of labels of any $\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z})$ of the ground truth $\boldsymbol{z}$. Although the theoretical bound is never violated for the test set, $\mathrm{O}_{\mathcal{G}}$ yields the lowest error rates. Obviously, this is due to the high subsampling rate of the NNs (on average 2.1 positions per character) and the high proportion of duplicated letters per word (around 10% of all words contain consecutive identical letters). If these conditions are less pronounced, the exact decoding methods (restricted to $\mathcal{F}^{-1}$) will work better.

Objective functions	WER	
	Graves	Leifert at al
O_{CTC}	16.34%	12.24%
$O_{\mathcal{F}}$	16.36%	12.25%
$O_{\mathcal{G}}$	**16.02%**	**12.05%**
$O_{\mathcal{H}}$	17.46%	12.93%

Table 5.3: Word error rates on RIMES data set for different objective functions. Besides the architecture described in Graves [2012], we also tested a NN described in Leifert et al. [2013] both with a higher subsampling rate. Error rates are averaged over 10 different initializiations.

Discussion

If only speed is important Levenshtein decoding should be the method of choice since it is by far faster than all other decoding techniques (more than 10 times faster than $O_{\mathcal{F}}$ or O_{CTC}). The objective $O_{\mathcal{G}}$ seems to be an acceptable compromise between speed and accuracy (30% less decoding time). Especially, if the decoding is accomplished on saved confidence matrices, the running time of the decoding process is very important.

In case of a complete transcription incl. Neural Network update and image preprocessing, the decoding usually consumes only a fraction of the running time. Thus, we use $O_{\mathcal{F}}$ as the decoding objective in the following. We prefer it over *CTC* since it allows an assignment of positions to characters. Thus, we are able to determine the begin and the end of a word, for example. The experiments also indicate that the error does not increase much from O_{CTC} to $O_{\mathcal{F}}$.

To decode ConfMats of NNs with high subsampling rate, it might be beneficial to use $\mathcal{G}$ since consecutive identical labels might cause problems while using O_{CTC} or $O_{\mathcal{F}}$.

5.2 Find Any Occurrence in the Output

One elementary question of information retrieval is whether or not the input image contains a specific word. Frinken et al. [2010] proposed an efficient algorithm to tackle this problem by calculating the probability

$$\max_{s \leq e \leq T} P_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$$

where

$$\mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}) := \max_{\substack{\boldsymbol{\pi}\in\mathcal{F}^{-1}(\boldsymbol{z}) \\ |\boldsymbol{\pi}|=e-s+1}} \prod_{t=s}^{e} y_{t,\pi_{t-s+1}}$$

for any given ConfMat. Thus, the resulting probability is the greatest probability of any label sequence from $\mathcal{F}^{-1}(\boldsymbol{z})$ in any sub-ConfMat from s to e. They derive an algorithm from a simplification of the Token Passing Algorithm proposed in Graves et al. [2008]. They show that their method is capable of doing keyword search in handwritten documents.

Alternatively, the algorithm can be derived by a slight modification of the Weighted Automaton $W_{\boldsymbol{X}}$ from Lemma 4.22:

Lemma 5.7. For any $\boldsymbol{z} \in \mathbb{A}^*$, let $\mathring{\mathrm{A}}_{\boldsymbol{z}} = (Q, \mathbb{A}', \delta, q_0, F)$ be the Automaton corresponding to $\boldsymbol{z}$ from Lemma 4.27. Let the WA $W^O_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}}) = (Q, \mathbb{A}', \lambda, q_0, F)$ with

$$\lambda(q, l, \bar{q}, t) := \begin{cases} 1 & \text{if } q = \bar{q} = q_0, \\ y_{t,l} & \text{if } q_0 \neq \bar{q} \in \delta(q, a) \wedge t \leq T, \\ 0 & \text{else.} \end{cases}$$

Then

$$\max_{\boldsymbol{\pi}\in(\mathbb{A}')^*} \max_{q\in F} \lambda^{\max}(q_0, \boldsymbol{\pi}, q, 1) = \max_{s\leq e\leq T} \mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}).$$

Proof. Since $\lambda^{\max}(q_0, \ell, q_0, t) = 1$,

$$\begin{aligned} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^*} \max_{q\in F} \lambda^{\max}(q_0, \boldsymbol{\pi}, q, 1) &= \max_{\boldsymbol{\pi},\hat{\boldsymbol{\pi}}\in(\mathbb{A}')^*} \max_{q\in F} \lambda^{\max}(q_0, \boldsymbol{\pi}, q_0, 1)\, \lambda^{\max}(q_0, \hat{\boldsymbol{\pi}}, q, |\boldsymbol{\pi}| + 1) \\ &= \max_{\boldsymbol{\pi},\hat{\boldsymbol{\pi}}\in(\mathbb{A}')^*} \max_{q\in F} \underbrace{\lambda^{\max}(q_0, \boldsymbol{\pi}, q_0, 1)}_{=1} \lambda^{\max}(q_0, \hat{\boldsymbol{\pi}}, q, |\boldsymbol{\pi}| + 1) \end{aligned}$$

with $s = |\boldsymbol{\pi}| + 1$ and considering only $\hat{\boldsymbol{\pi}}$ which yield non-zero reward

$$\begin{aligned} &= \max_{s} \max_{\substack{\hat{\boldsymbol{\pi}}\in\mathcal{F}^{-1}(\boldsymbol{z}) \\ |\hat{\boldsymbol{\pi}}|\leq T-s+1}} \max_{q\in F} \lambda^{\max}(q_0, \hat{\boldsymbol{\pi}}, q, s) \\ &= \max_{s\leq e\leq T} \mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}) \end{aligned}$$

□

In contrast to $W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})$, the support of $W^O_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})$ is not $\mathring{\mathrm{A}}_{\boldsymbol{z}}$ since $\lambda(q_0, \ell, q_0, 1) = 1$ for any ℓ. That means, the support of $W^O_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})$ stays in q_0 reading any label, including z_1 which also moves to q_1. Thus, the support is not a DFA anymore.

Due to Lemma 5.7, the intermediates $\alpha_t(q)$ of Algorithm 3 ($\mathcal{X}_i = \mathbb{A}'$) calculate

$\max_{s\le e\le T} \mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$ efficiently: Since

$$\begin{aligned}\max_{s\le e\le T} \mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}) &= \max_{\boldsymbol{\pi}\in(\mathbb{A}')^*} \max_{\bar{q}\in F} \lambda^{\max}(q_0, \boldsymbol{\pi}, \bar{q}, 1)\\ &= \max_{t\le T} \max_{\bar{q}\in F} \alpha_t(\bar{q}),\end{aligned}$$

we have to maximize $\alpha_t(q)$ over each $q \in F$ and each $t \le T$ instead of fixing $t = T$. $W^O_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{z}})$ has two finial states corresponding to the last character and an optional NaC afterwards. The following lemma shows that the NaC-state will not take the maximum. We need to introduce two new terms first:

Definition 5.8 (Predecessor, successor). Let $A = (Q, \mathbb{A}, \delta, q_0, F)$ be an Automaton. We denote the predecessors of any state $\bar{q} \in Q$ by:

$$\mathrm{pre}(\bar{q}) := \{q \in Q \mid \exists \boldsymbol{\pi} \in (\mathbb{A}')^* : \bar{q} \in \delta^*(q, \boldsymbol{\pi})\}.$$

Analogously, the successors of $q \in Q$ are defined by:

$$\mathrm{succ}(q) := \{\bar{q} \in \delta^*(q, \boldsymbol{\pi}) \mid \boldsymbol{\pi} \in (\mathbb{A}')^*\}.$$

Definition 5.9 (Mandatory Predecessor). Let $A = (Q, \mathbb{A}, \delta, q_0, F)$ be an Automaton. A state q is a *mandatory predecessor for his successors* iff for any $\hat{q} \in \mathrm{succ}(q)$ any path from q_0 to $\hat{q}$ also contains q.

Especially, any character state[1] of $\mathring{\mathrm{A}}_{\boldsymbol{z}}$ is a mandatory predecessor for his successors.

Lemma 5.10. Let $(Q, \mathbb{A}', \lambda, q_0, F)$ be a WA with $0 \le \lambda(q, \ell, \hat{q}, t) \le 1$ for any $q, \hat{q} \in Q$, for any $\ell \in \mathbb{A}'$ and for any t. Let $q \in Q$ be a mandatory predecessor for his successors. Then,

$$\max_{\bar{t}<t} \alpha_{\bar{t}}(q) \ge \max_{\bar{q}\in\mathrm{succ}(q)\setminus\{q\}} \alpha_t(\bar{q}).$$

Proof. Obviously, $0 \ge \lambda^{\max}(q, \boldsymbol{\pi}, \bar{q}, t) \le 1$. Thus,

$$\begin{aligned}\max_{\bar{t}<t} \alpha_{\bar{t}}(q) &= \max_{\bar{t}<t} \max_{\boldsymbol{\pi}\in\mathbb{A}^{\bar{t}}} \lambda^{\max}(q_0, \boldsymbol{\pi}, q, 1)\\ &\ge \max_{\bar{t}<t} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^{\bar{t}}} \lambda^{\max}(q_0, \boldsymbol{\pi}, q, 1) \max_{\bar{q}\in\mathrm{succ}(q)\setminus\{q\}} \max_{\hat{\boldsymbol{\pi}}\in(\mathbb{A}')^{t-\bar{t}}} \lambda^{\max}(q, \hat{\boldsymbol{\pi}}, \bar{q}, \bar{t})\end{aligned}$$

and since any path from q_0 to $\bar{q}$ contains q s.t.

$$\begin{aligned}&\ge \max_{\bar{q}\in\mathrm{succ}(q)\setminus\{q\}} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^t} \lambda^{\max}(q_0, \boldsymbol{\pi}, \bar{q}, 1)\\ &= \max_{\bar{q}\in\mathrm{succ}(q)\setminus\{q\}} \alpha_t(\bar{q}).\end{aligned}$$

□

[1] That means, $q_{2k} = (q_\ell, q_a)$ with $\ell \ne \oslash$ where q_ℓ is a state of $T_{\mathcal{F}}$ and q_a is a state of $A_{\boldsymbol{z}}$.

Thus, it is sufficient to consider only $\alpha_t(q_{|z'|-1})$ of the second last state and return the maximum over all time steps t:

$$\max_{s \leq e \leq T} \mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}) = \max_{t \leq T} \alpha_t(q_{|z'|-1}).$$

The corresponding maximizing bounds (s^*, e^*) yield an alignment of ConfMat and keyword.

Small modifications lead to variants of the above algorithm searching a word at the beginning ($s^* = 1$) or at the end ($e^* = T$) of a ConfMat to search for words explicitly starting or completing the line. We derive a more flexible algorithm in Section 6 which will be applied to keyword search in Section 7.2.

A modification of the above algorithm such that it returns the CTC probability is more elaborate. The Weighted Automaton from Lemma 5.7 is designed for the maximum reward (i.e., the path probability) only since the support is no DFA. In fact, a CTC version which returns $\sum_\pi \mathrm{P}_{s:e}(\boldsymbol{\pi} \mid \boldsymbol{X})$ is not meaningful since CTC cannot align labels and positions to determine s and e. An alternative target function, which calculates the probability of the entire line, could be:

$$\sum_{\substack{\boldsymbol{\pi} \in (\mathbb{A}')^T \\ \mathcal{F}(\boldsymbol{\pi}) \in \mathcal{L}(.*\boldsymbol{z}.*)}} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X}).$$

However, the corresponding DFA which accepts $\mathcal{L}(.*\boldsymbol{z}.*)$ has more connections which directly influences the number of multiplications which are necessary to calculate α. Furthermore, the result will be a probability of the whole line instead of only the part containing the keyword.

5.3 Speed-Up Methods

In this section, we exploit redundancies and limiting effects occurring during the calculation of

$$\max_{\boldsymbol{z} \in \mathcal{V}} \mathrm{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$$

for any vocabulary $\mathcal{V} \subsetneq \mathbb{A}^*$.

5.3.1 Vocabulary Automaton

The minimum Automaton accepting the finite vocabulary $\mathcal{V}$ is described in Section 2.2. However, an assignment of the winning state to the best vocabulary item is not possible without tracking back the maximum state sequence. Furthermore, a top-n list of the most likely words is not possible. For practical applications, it is typically

necessary to return a list of the top-n vocabulary items to provide alternatives for example for the user or a rescoring LM.

To enable a top-n list, we omit the substitution of equivalent states compared to Algorithm 1. The resulting Automaton finally has more states but each accepting state represents an individual word in the vocabulary. Algorithm 5 creates such a tree-like Automaton where common prefixes are read by the same transitions.

Algorithm 5: Vocabulary Automaton

input : Vocabulary $\mathcal{V}$ in lexicographical order
output: $A = (Q, \mathbb{A}, \delta, t_1^\varepsilon, F)$

$F \leftarrow \{\}$;
$\bar{\boldsymbol{w}} \leftarrow \varepsilon$;
$Q \leftarrow \{t_1^\varepsilon\}$;
for $\boldsymbol{w} \in \mathcal{V}$ **do**
  $c \leftarrow \max\{i \in \mathbb{N} \mid \boldsymbol{w}_{1:i} = \bar{\boldsymbol{w}}_{1:i}\}$; `// index of maximum commen prefix`
  for $i = 1$ **to** c **do**
    $t_i^{\boldsymbol{w}} \leftarrow t_i^{\bar{\boldsymbol{w}}}$
  for $i = c + 1$ **to** $|\bar{\boldsymbol{w}}|$ **do**
    $Q \leftarrow Q \cup \{\bar{t}_i^{\boldsymbol{w}}\}$; `//` $\bar{t}_i^{\boldsymbol{w}}$ `is a new state s.t.` $Q \cap \{\bar{t}_i^{\boldsymbol{w}}\} = \emptyset$
    $\delta(\bar{t}_{i-1}^{\boldsymbol{w}}, \bar{w}_i) \leftarrow \{\bar{t}_i^{\boldsymbol{w}}\}$;
  $F \leftarrow F \cup \{\bar{t}_{|\bar{\boldsymbol{w}}|}^{\boldsymbol{w}}\}$;
  $\bar{\boldsymbol{w}} \leftarrow \boldsymbol{w}$;

The composition of $T_{\mathcal{F}}$ and $A_{\mathcal{V}}$ which is created by Algorithm 5 yields the Automaton $\mathring{\mathrm{A}}_{\mathcal{V}}$ which accepts the label sequences that collapse to any $\boldsymbol{z} \in \mathcal{V}$ since $T_{\mathcal{F}}$ relates labels to words and $A_{\mathcal{V}}$ accepts only feasible words.

5.3.2 Limit

Most vocabulary items fit very poorly to the ConfMat such that the calculation of certain branches of the search space can be pruned. This is the case, e.g., if we already found a more likely word or if we are not interested in results with probabilities smaller than a certain threshold. We give three different criteria when to stop the calculation based on a threshold θ which represents the minimum likelihood of interest. First of all, Lemma 5.10 yields such a criterion. Other criteria to detect that the probability will not exceed θ will be given by the following lemma:

Lemma 5.11. Let $(Q, \mathbb{A}', \lambda, q_0, F)$ be a WA with $0 \leq \lambda(q, \ell, \hat{q}, t) \leq 1$ for any $q, \hat{q} \in Q$, for any $\ell \in \mathbb{A}'$ and for any t.

(a) The value $\max_{\hat{q} \in \mathrm{pre}(q)} \alpha_t(\hat{q})$ decreases monotonously in t for any $q \in Q$.

(b) If q is a mandatory predecessor for its successors, for any $t < T$,

$$\max_{\hat{q}\in\mathrm{succ}(q)} \alpha_t(\hat{q}) \geq \max_{\hat{q}\in\mathrm{succ}(q)\setminus\{q\}} \alpha_{t+1}(\hat{q}).$$

Proof. (a)

$$\begin{aligned}
\max_{\hat{q}\in\mathrm{pre}(q)} \alpha_t(\hat{q}) &= \max_{\hat{q}\in\mathrm{pre}(q)} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^t} \lambda^{\max}(q_0, \boldsymbol{\pi}, \hat{q}, 1) \\
&\geq \max_{\hat{q}\in\mathrm{pre}(q)} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^t} \left(\lambda^{\max}(q_0, \boldsymbol{\pi}, \hat{q}, 1) \max_{(\bar{q},\ell)\in Q\times\mathbb{A}'} \lambda(\hat{q}, \ell, \bar{q}, t+1) \right) \\
&\geq \max_{\hat{q}\in\mathrm{pre}(q)} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^{t+1}} \lambda^{\max}(q_0, \boldsymbol{\pi}, \hat{q}, 1) = \max_{\hat{q}\in\mathrm{pre}(q)} \alpha_{t+1}(\hat{q})
\end{aligned}$$

(b)

$$\begin{aligned}
\max_{\hat{q}\in\mathrm{succ}(q)} \alpha_t(\hat{q}) &= \max_{\hat{q}\in\mathrm{succ}(q)} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^t} \lambda^{\max}(q_0, \boldsymbol{\pi}, \hat{q}, 1) \\
&\geq \max_{\hat{q}\in\mathrm{succ}(q)} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^t} \max_{(\bar{q},\ell)\in Q\times\mathbb{A}'} \lambda^{\max}(q_0, \boldsymbol{\pi}, \hat{q}, 1)\, \lambda(\hat{q}, \ell, \bar{q}, t+1)
\end{aligned}$$

since q is a mandatory predecessor

$$\geq \max_{\bar{q}\in\mathrm{succ}(q)\setminus\{q\}} \max_{\boldsymbol{\pi}\in(\mathbb{A}')^{t+1}} \lambda^{\max}(q_0, \boldsymbol{\pi}, \bar{q}, 1) = \max_{\bar{q}\in\mathrm{succ}(q)\setminus\{q\}} \alpha_{t+1}(\bar{q})$$

□

Remark 5.12. Let $W_{\boldsymbol{X}}(A)$ be the WA for $A = \mathring{\mathrm{A}}_{\boldsymbol{z}}$ or $A = \mathring{\mathrm{A}}_{\mathcal{V}}$. Assume that any result with probability smaller than θ is immediately rejected.

(a) Using Lemma 5.11 (a), the calculation of Algorithm 3 can be canceled if $\max_{q\in Q} \alpha_t(q) < \theta$ for any t since

$$\max_{\hat{q}\in F} \alpha_T(\hat{q}) \leq \max_{q\in Q} \alpha_T(q) \leq \max_{q\in Q} \alpha_t(q) < \theta.$$

(b) Let q be character state. Due to Lemma 5.10, the calculation of $\alpha_t(\hat{q})$ for any $\hat{q} \in \mathrm{succ}(q)$ can be stopped if $\max_{t\leq T} \alpha_t(q) < \theta$ since

$$\max_{\hat{q}\in F} \alpha_T(\hat{q}) \leq \max_{\bar{\mathrm{succ}}(q)} \alpha_T(\bar{q}) \leq \max_{t\leq T} \alpha_t(q) < \theta.$$

We exploit the above remark during the calculation of a whole vocabulary by updating the threshold iteratively. If we are interested in the n most likely words, θ is equal to the likelihood of the n-th most likely word up to the current vocabulary item. The limit value θ is updated whenever the list of top-n words is updated. Obviously, the above abort criteria are applied in opposite situations. While Remark 5.12(a) is applied if $\alpha_t(q)$ is calculated for each q before moving to $t + 1$, Remark

5.12(b) is applied if $\alpha_t(q)$ is calculated for each t before moving to the next q. For $\text{Å}_{\boldsymbol{z}}$ both variants are more or less equivalent. For $\text{Å}_{\mathcal{V}}$, we prefer Remark 5.12(b) since whole branches can be cut at once.

Remark 5.13. Let $W^{O}_{\boldsymbol{X}}(\text{Å}_{\boldsymbol{z}})$ be the WA of Lemma 5.7. If $\max_{\hat{q}\in\text{succ}(q)} \leq \theta$, then $\alpha_{t+1}(\hat{q}) \leq \theta$ for any $\hat{q} \in \text{succ}(q)$ due to Lemma 5.11(b). Since the first states q are relatively likely, the conditions of the previous criteria will not be satisfied at an early stage. Lemma 5.11 (b) does not yield a criterion to stop the calculation of Algorithm 3 for entire branches but it could reduce the number of values $\alpha_t(q)$ which are calculated.

Initial Limit for the Vocabulary Automaton One way to quickly get a reasonable initial limit value for the most likely vocabulary item is to search for the item with the smallest Levenshtein Distance to the collapsed best path $\mathcal{F}(\boldsymbol{\beta})$ and use the item's probability to initialize the threshold. In our experiments the gain resulting from this search was negligible or negative since this operation consumed more time than it saved. Alternatively, we search for the position, $\mathcal{F}(\boldsymbol{\beta})$ would appear in the vocabulary according to a lexicographical order and set the threshold to the maximum of the probabilities of the neighboring items.

5.3.3 Experimental Validation of Speed-Up Strategies

The precise gain of the above strategies strongly depends on the given vocabulary. To give an impression on the effects of the previous methods, we test on the RIMES test dataset with the same Neural Networks as in the first experiment of Section 5.1.3. Again, the results are averaged over 10 different initializations.

We compare the decoding times of iteratively calculating $\text{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$ using $\text{Å}_{\boldsymbol{z}}$ for any $\boldsymbol{z}$ without any stopping criterion (referred to as "one-by-one"), using $\text{Å}_{\boldsymbol{z}}$ with stopping the calculation q if $\alpha_t(q)$ cannot reach θ ("one-by-one & limit") and using $\text{Å}_{\mathcal{V}}$ with limit (Vocabulary Automaton & limit). The calculation is canceled if $\max_{t\leq T} \alpha_t(q) < \theta$ (Remark 5.12(b)).

Table 5.4 shows the average time needed to decode the whole data set. Obviously, many calculations can be avoided by stopping the calculation if there is no chance to improve the result. The running time of the decoding with limit decreases to one third of the decoding time without limit. The Vocabulary Automaton from Subsection 5.3.1 saves almost half of the computation time compared to one-by-one.

As already reported, the above time reduction is obtained by searching for the most likely element of the vocabulary. If we search for the top-n results, these time savings will decrease. For $n = 10$, the limit-version of one-by-one takes 63.9% (instead of 33.7%) of the original time.

Applying the limit is always practical since the results do not differ and the decoding

	time (in ms)	
	Graves	Leifert at al
One-by-one	162764.1	156537.4
One-by-one & limit	54932.9	52481.5
Vocabulary Automaton & limit	28683.9	27737.5

Table 5.4: Decoding times (milli seconds) of path probability without limit, with limit and Voabulary Automaton averaged over 10 different initializations. Net architectures from Graves [2012] and Leifert et al. [2014b]

time decreases dramatically. Even for a top-n search the time saving persists but decreases if n increases. Further, the Vocabulary Automaton saves the calculation of 60% of the $\alpha_t(q)$ compared to the calculation without caching but with limit. The Vocabulary Automaton needs some overhead such that we cannot transfer the savings totally to the running time.

5.4 Conclusion

In this chapter, we consider single-word decoding. First, we investigated different objective functions (and constraints) which differ in performance and running time. We gave a precise description of the corresponding Automata which in combination with Algorithm 3 calculate the maximum objective value. The most exact method is CTC as long as the subsampling rate of the NN is not too high. In case of a highly downsampled ConfMat, we suggested an alternative decoding approach. The path version of CTC (which corresponds to the Viterbi approximation for HMMs) yields error rates close to those of CTC but has other advantages such as an Automatic assignment of labels to positions. This assignment is of interest, e.g. in case of keyword spotting. The method provided in Section 5.2 yields such a keyword spotting method. In the last section, we gave criteria to improve the running time of the decoding process for $O_{\mathcal{F}}$. In the experiments, we could decrease the running time down to 20%.

6 Decoding Constrained by Regular Expressions

Handwritten text recognition is a complex task which requires advanced decoding methods. For example, a typical subproblem in full text recognition is structuring the recognizer's output into a sequence of regions of words, punctuations and numbers to calculate the word error rate or to apply an LM. In many cases, the most likely label sequence yields an acceptable segmentation. However, it happens that this label sequence is not feasible, i.e., it does not match the expected structure and has to be corrected. Finding the optimal feasible structure is one of many applications of this chapter. For this purpose, we describe feasible structures by Regular Expressions and give the corresponding Automaton which accepts any label sequence which collapses to the corresponding Regular Language.

Regular expressions can be very complex and the calculation of the probability of all feasible sequences can be very time consuming. We give an approximation of the most likely label sequence which we motivate theoretically and verify experimentally. That means, we give a pruning heuristic which yields the optimal path under certain conditions and we show that these conditions are satisfied in practical scenarios.

Beyond finding the optimal feasible label sequence, previously specified parts of the Regular Expression can be aligned to the positions of the ConfMat immediately. In the case of keyword spotting, this feature simplifies the computation of, e.g., the likelihood of the keyword and the likelihood of the separating space.

The remainder of this chapter is organized as follows: First, we have a look at previous decoding methods based on Regular Expressions. In Section 6.2, we analyze the Automaton which accepts feasible label sequences. We introduce the RegEx-Decoder in Section 6.4. Afterwards, we stress test the algorithm in an experimental section. We combine Beam Search and the developed algorithm to handle larger vocabularies in Section 6.6.

6.1 Previous Work

We already introduced connections between HMMs and Finite State Automata in the Section 4.2 (Mohri [2009], Dupont et al. [2005], Vidal et al. [2005]). Although the equivalence of HMMs and Automata is well known, there are only few approaches

which link HMMs and Regular Expressions for practical applications.

Some links between Regular Expressions, their corresponding Automata and HMMs are given in Krogh et al. [1998]. The authors showed how to create HMMs from Regular Expressions to detect biological sequences. A similar but generalized approach is given in Kessentini et al. [2013]. Besides the standard keyword-filler-model (introduced later in Section 7.2) for keyword search, they also proposed an enhanced model where only the prefix or suffix of the keyword is given. This model allows a set of feasible words containing the defined prefix or suffix.

Recently, Bideault et al. published a similar approach to ours in Bideault et al. [2015b]. They proposed an HMM - BLSTM hybrid model for word spotting exploiting Regular Expressions. Analogously to Kessentini et al. [2013], they build small HMM models in advance (e.g. for a keyword, for digits or letters) and combine them to a model capturing the Regular Expression. The authors then applied their model to keyword and "regex" spotting.

The results of the current chapter are published in Strauß et al. [2016]. To keep this chapter consistent with the previous, we adapt the notation to Weighted Automata.

6.2 Automaton of feasible label sequences

To decode any ConfMat according to a Regular Language $\mathcal{L}$, the corresponding Automaton has to be composed with $T_{\mathcal{F}}$:

Corollary 6.1. For any Regular Expression $\boldsymbol{r}$, let $A_{\boldsymbol{r}} = (Q, \mathbb{A}, \delta, q_0, F)$ be the Automaton (NFA without epsilon transitions) accepting $\mathcal{L}(\boldsymbol{r})$ where $Q := \{q_0, \dots, q_n\}$.

Let $\mathring{\mathrm{A}}_{\boldsymbol{r}} := \left(\mathring{Q}, \mathbb{A}', \mathring{\delta}, q_0^{\oslash}, \mathring{F}\right)$ be the *Extended Automaton* where $\mathring{Q} := \{q_i^{\ell} \mid i \in \{0, \dots, n\}, \ell \in \mathbb{A}'\}$, for any $a \in \mathbb{A}$, $a \neq \ell \in \mathbb{A}'$ and $i \in \{0, \dots, n\}$

$$\mathring{\delta}(q_i^{\ell}, \ell) := \{q_i^{\ell}\}, \tag{6.2.1}$$

$$\mathring{\delta}(q_i^{\ell}, a) := \{q_j^{a} \in \mathring{Q} \mid q_j \in \delta(q_i, a)\}, \tag{6.2.2}$$

$$\mathring{\delta}(q_i^{a}, \oslash) := \{q_i^{\oslash}\} \tag{6.2.3}$$

and $\mathring{F} := \{q_i^{\ell} \in \mathring{Q} \mid q_i \in F\}$. Then $\mathring{\mathrm{A}}_{\boldsymbol{r}}$ accepts $\boldsymbol{\pi} \in (\mathbb{A}')^*$ if and only if $\mathcal{F}(\boldsymbol{\pi}) \in \mathcal{L}(\boldsymbol{r})$.

Proof. The proposition is a direct consequence from $\mathring{\mathrm{A}}_{\boldsymbol{r}} = T_{\mathcal{F}} \circ A_{\boldsymbol{r}}$ with $q_i^{\ell} = (q_{\ell}, q_i) \in Q_{\mathcal{F}} \times Q$ with $T_{\mathcal{F}} = (Q_{\mathcal{F}}, \mathbb{A}', \mathbb{A}, \delta_{\mathcal{F}}, q_{\oslash}, Q_{\mathcal{F}})$. □

Interpretation: We split every state from Q into $|\mathbb{A}'|$ new states. Each of them is reached by reading a specific label if there is a connection at all. If there is connection from q_i to q_j in $A_{\boldsymbol{r}}$ via reading $a \in \mathbb{A}$, q_i^{ℓ} is connected to q_j^{a} in $\mathring{\mathrm{A}}_{\boldsymbol{r}}$ for any $\ell \neq a$.

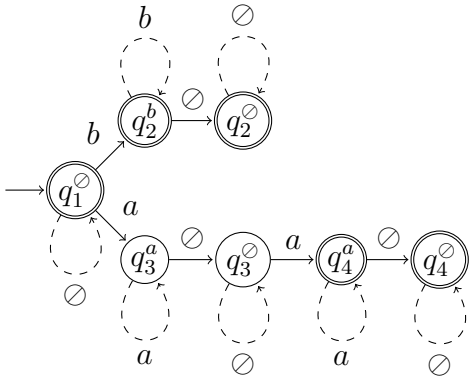


Figure 6.1: Automaton $\mathring{\mathrm{A}}_{\boldsymbol{r}}$ with $\boldsymbol{r}$ =`aaa*|b*` (compare Figure 2.3(b)). Unreachable states are not presented. If q_j is final, any $q_j^{\ell} \in \mathring{Q}$ is final.

Thus, one transition $q_j \in \delta(q_i, a)$ in $A_{\boldsymbol{r}}$ corresponds to $|\mathbb{A}'|$ arcs in $\mathring{\mathrm{A}}_{\boldsymbol{r}}$. Furthermore, any character state q_i^a $(a \in \mathbb{A})$ is connected to the corresponding NaC-state $q_i^{\oslash}$.

Obviously, some states may be not reachable and can be ignored in the calculation such that the precise number of meaningful states and connections strongly depends on the considered Regular Expression. Throughout this section, we assume that $\mathring{Q}$ contains only the reachable states of the composition. Figure 6.1 shows the necessary states and transitions for the Regular Expression `aaa*|b*` (compare Figure 2.3(b)).

Once $\mathring{\mathrm{A}}_{\boldsymbol{r}}$ is generated, Algorithm 3 is applied to $W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{r}})$ to calculate the maximum reward of any label sequence from $\mathcal{F}^{-1}(\boldsymbol{z})$.

6.3 Naïve Approach

In this section, we give two naïve alternatives to Algorithm 3.

6.3.1 A^* Search

The A^* Search is a general search strategy which explores the search space by considering those candidates first which are the most promising. Algorithm 6 describes a naïve A^* *Search* algorithm on Regular Expressions that returns the most likely label sequence. This algorithm guarantees an optimal solution but it can be time consuming because of the huge number of possible label sequences. To cut unlikely label sequences, we define an upper bound $\overline{\mathrm{P}}(\boldsymbol{\pi}|\boldsymbol{X}) := \mathrm{P}(\boldsymbol{\pi}\boldsymbol{\tau}|\boldsymbol{X})$ as estimation of the final probability of any label sequence with prefix $\boldsymbol{\pi}$. In our experiments, we used $\boldsymbol{\tau} := \boldsymbol{\beta}_{|\boldsymbol{\pi}|+1:T}$ such that $\overline{\mathrm{P}}(\boldsymbol{\pi}|\boldsymbol{X}) := \prod_{r=1}^{|\boldsymbol{\pi}|} y_{r,\pi_r} \prod_{s=|\boldsymbol{\pi}|+1}^{T} y_{s,\beta_s}$. Another heuristic which appears to work well in practice is to sort the prefix list $\mathbb{L}$ by $\frac{\overline{\mathrm{P}}(\boldsymbol{\pi}|\boldsymbol{X})}{t}$. The

precise decoding time for this algorithm depends heavily on the likelihood of the optimal label sequence $\boldsymbol{\pi}^*$. If $\mathrm{P}(\boldsymbol{\pi}^*|\boldsymbol{X}) \approx 1$, other candidates will be pruned soon and the running time will be very small. If $\mathrm{P}(\boldsymbol{\pi}^*|\boldsymbol{X}) \ll 1$ many label sequences remain in the list and have to be evaluated.

Algorithm 6: A^* Search

input : ConfMat $\boldsymbol{Y}$, Extended Automaton $\mathring{\mathrm{A}} = (\mathring{Q}, \mathbb{A}', \mathring{\delta}, \mathring{q}_0, \mathring{F})$
output: most likely feasible label sequence $\boldsymbol{\pi}^*$

```
for γ ∈ 𝔸′ do
    for q ∈ δ̊(q̊₀, γ) do
        Add (q, γ, 1) to 𝕃;                          /* initialize 𝕃 */
while 𝕃 not empty do
    (q, π, t) ← Item from 𝕃 with maximum P̄(π|X)/t;
    Remove (q, π, t) from 𝕃;
    if t < T then
        for γ ∈ 𝔸′ \ {π_t} do
            for q′ ∈ δ̊(q, γ) do
                Add (q′, πγ, t + 1) to 𝕃;
        Add (q, ππ_t, t + 1) to 𝕃;
    else
        if q ∈ F̊ and P(π | X) > P(π* | X) then
            π* ← π;
            Remove all (q′, π′, t′) ∈ 𝕃 with P̄(π′|X) < P(π|X);
```

6.3.2 Beam Search

Since the number of feasible label sequences grows exponentially in T in the worst case, there is a standard heuristic to reduce the search space called *Beam Search* which allows only n prefixes[1] at each position. In Graves and Jaitly [2014], for example, the authors introduced a Beam Search algorithm for efficient decoding in case of speech recognition. They maximize over the CTC-probability instead of the path probability there.

Algorithm 7 contains the pseudo code of a Beam Search variant adapted to our problem. Generally, Beam Search does not guarantee to find the optimal sequence. The given algorithm has the additional drawback that it does not even guarantee to find any feasible label sequence at all since the final list $\mathbb{L}$ could contain only those $(q, \boldsymbol{\pi}, T)$ with $q \notin \mathring{F}$.

[1] n is called the *Beam Width*.

Algorithm 7: Beam Search

input : ConfMat $\boldsymbol{Y}$, Extended Automaton $\mathring{\mathrm{A}} = (\mathring{Q}, \mathbb{A}', \mathring{\delta}, \mathring{q}_0, \mathring{F})$, Beam Width n
output: most likely feasible label sequence $\boldsymbol{\pi}^*$ which survived in $\mathbb{L}$

```
for γ ∈ 𝔸′ do
    for q ∈ δ̊(q̊_0, γ) do
        Add (q, γ, 1) to 𝕃;                          /* initialize 𝕃 */
for i ← 2 to T do
    𝕃̄ ← the n most likely items of 𝕃;
    𝕃 ← {};
    for (q, π, t) ∈ 𝕃̄ do
        for γ ∈ 𝔸′ \ {π_t} do
            for q′ ∈ Q : q′ ∈ δ̊(q, γ) do
                Add (q′, πγ, t + 1) to 𝕃;
        Add (q, ππ_t, t + 1) to 𝕃;
π* ← π from (q, π, t) ∈ 𝕃̄ with maximum P(π|X) and q ∈ F̊;
```

6.4 Efficient Decoding of Regular Expressions

Given a Regular Expression $\boldsymbol{r}$ and the corresponding Extended Automaton $\mathring{\mathrm{A}}_{\boldsymbol{r}} = (\mathring{Q}, \mathbb{A}', \mathring{\delta}, \mathring{q}_0, \mathring{F})$, we search for the most likely word $\boldsymbol{z}^*$ in $\mathcal{L}(\boldsymbol{r})$:

$$\boldsymbol{z}^* = \arg\max_{\boldsymbol{z} \in \mathcal{L}(\boldsymbol{r})} \max_{\boldsymbol{\pi} \in \mathcal{F}^{-1}(\boldsymbol{z})} \mathrm{P}(\boldsymbol{\pi}|\boldsymbol{X}).$$

Since even for simple Regular Expressions $\boldsymbol{r}$, $\mathcal{L}(\boldsymbol{r})$ could contain an infinite number of words whose likelihood cannot be calculated one after another. Thus, exploiting the graphical structure is not only efficient but necessary to find the most likely $\boldsymbol{z}^*$.

6.4.1 Preliminaries

We already mentioned that the maximum reward, i.e., the highest probability, can be calculated using Algorithm 3 in combination with $W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{r}})$. The Extended Automaton contains up to $|\mathbb{A}'|$ times more states and transitions compared to the original one and in fact many calculations can be saved. In Section 3, we mentioned that Algorithm 3 needs $\mathcal{O}(|\mathring{Q}|^2\,|\mathbb{A}|\,T)$ multiplications. Thus, the running time seems $\mathcal{O}(|Q|^2\,|\mathbb{A}|^3\,T)$ in the worst case since $|\mathring{Q}| = |Q||\mathbb{A}'|$ but this is not exact. The factor $|\mathbb{A}|$ in the estimation of Algorithm 3 results from the assumption that the Automaton could possibly move from state q_i to q_j by reading any character from $\mathbb{A}$. The Extended Automaton moves from q_i^ℓ to q_j^a only by reading character a such that

the complexity is rather $\mathcal{O}(|Q|^2\,|\mathbb{A}|^2\,T)$ in the worst case. Below, we analyze the computations and show which of them can be avoided leading to an algorithm that only needs $\mathcal{O}(|Q|^2\,T)$ operations. More precisely, the resulting algorithm considers only the three most likely labels per transition from any q_i to q_j and saves only three different likelihoods for distinct prefixes per original state. The gain depends highly on the actual Regular Expression.

To explain the new algorithm, we first need to introduce some notation on $W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{r}})$ using terms from $A_{\boldsymbol{r}}$:

Definition 6.2. $(\xi, \Xi, \gamma, \Gamma)$ Let $\gamma_t^k(i,j) \in \{\gamma \in \mathbb{A} \mid q_j \in \delta(q_i, \gamma)\}$ denote k-th likely characters at position t among any character which moves q_i to q_j in $A_{\boldsymbol{r}}$:

$$y_{t,\gamma_t^1(i,j)} \geq y_{t,\gamma_t^2(i,j)} \geq y_{t,\gamma_t^3(i,j)} \geq \dots$$

Let $\Gamma_t(i,j) := \{\gamma_t^1(i,j), \gamma_t^2(i,j), \gamma_t^3(i,j)\}$ denote the set of the three most likely characters per transition. If the transition only reads one or two such characters, Γ is accordingly smaller. Analogously, we sort the states q_i^ξ for fixed i according to their reward at time t:

$$\alpha_t\left(q_i^{\xi_t^1(i)}\right) \geq \alpha_t\left(q_i^{\xi_t^2(i)}\right) \geq \dots$$

Let $\Xi_t(i) := \{\xi_t^1(i), \xi_t^2(i)\} \cup \{\oslash\}$ denote the set containing the NaC and the characters $\xi_t^j(i)$ with the greatest reward $\alpha_t\left(q_i^{\xi_t^j(i)}\right)$ at position t. If there is only one or no transition in $A_{\boldsymbol{r}}$ which moves to q_i, $\Xi_t(i)$ is accordingly smaller[2] .

Recall Algorithm 3 for $\mathring{\mathrm{A}}_{\boldsymbol{r}}$: The likelihood of the most likely label sequence is

$$\max_{\boldsymbol{\pi} \in (\mathbb{A}')^T} \rho^{\max}(\boldsymbol{\pi}) = \max_{q_j^\ell \in \mathring{F}} \alpha_T\left(q_j^\ell\right).$$

Obviously, it is sufficient to know the most likely state $q_j^{\xi_j^1(T)}$ for any j at time T. Unfortunately, we potentially need more than the most likely state at time $T-1$ to determine $\xi_j^1(T)$:

$$\alpha_t(q_i^\gamma) = \max_{q_j^\xi \in \mathring{Q}} \alpha_{t-1}(q_j^\xi)\mathring{\lambda}(q_j^\xi, \gamma, q_i^\gamma, t)$$

According to Corollary 6.1 there is a direct transition between q_i^ℓ and q_j^ℓ reading any $\ell \in \mathbb{A}'$ if and only if $i = j$. Any other transition from q_i^ξ to q_j^γ reading γ is valid if $\gamma \neq \oslash$ and $q_j \in \delta(q_i, \gamma)$ or $\gamma = \oslash$ and $i = j$. Since $\mathring{\lambda}(q_i^\xi, \gamma, q_j^\gamma, t) = y_{t,\gamma}$ for any

[2]Especially, if there is no cycle in $A_{\boldsymbol{r}}$ which contains q_0, $\Xi_t(0) = \{\oslash\}$.

connection which is valid in $\mathring{\mathrm{A}}_{\boldsymbol{r}}$ and $\mathring{\lambda}(q_i^{\xi}, \gamma, q_j^{\gamma}, t) = 0$ otherwise,

$$\alpha_t(q_j^{\gamma}) = \max \left\{ \max_{\gamma:\, q_j \in \delta(q_i,\gamma)} \max_{\xi \in \mathbb{A}'} \alpha_{t-1}(q_i^{\xi}) y_{t,\gamma}(1 - \delta_{\xi,\gamma}),\ \alpha_{t-1}(q_j^{\gamma}) y_{t,\gamma} \right\} \tag{6.4.1}$$

for $\gamma \in \mathbb{A}$ and

$$\alpha_t(q_j^{\oslash}) = y_{t,\gamma} \max_{\xi \in \mathbb{A}'} \alpha_{t-1}(q_j^{\xi}). \tag{6.4.2}$$

Unfortunately, the Kronecker delta $\delta_{\xi,\gamma}$ prevents an individual maximization of α_{t-1} and $y_{t,\gamma}$ in Eq. (6.4.1). Recall that $\alpha_0(q_i^{\ell}) = 1$ if $\ell = NaC$ and $i = 0$ and $\alpha_0(q_i^{\ell}) = 0$ otherwise.

6.4.2 Pruning

In the following, we analyze the calculation of α and speed-up the process by avoiding unnecessary computations. The speed-up is based on two theorems which finally lead to a time complexity which is independent of the number of labels in $\mathbb{A}'$. The first theorem states that it is sufficient to know α of the two most likely states $q_i^{\xi_t^1(i)}$ and $q_i^{\xi_t^2(i)}$ and of the corresponding NaC-state $q_i^{\oslash}$ to calculate the contribution to $\alpha_t(q)$ of non-loop-transitions. Additionally, we only need the three most likely probabilities $y_{t,a}$ (i.e., $a \in \Gamma_t(i,j)$) per transition. The second theorem justifies that it is reasonable to do the same for the loop of q_i^{γ} under certain conditions. To proof the first theorem, we need the following Lemma:

Lemma 6.3. Let Ξ and Γ be finite sets which may have a non empty intersection. Consider two functions $f : \Xi \to \mathbb{R}_{\geq 0}$ and $g : \Gamma \to \mathbb{R}_{\geq 0}$ and denote the elements according to their value under f and g, respectively: $f(\xi^1) \geq f(\xi^2) \geq \ldots$ and $g(\gamma^1) \geq g(\gamma^2) \geq \ldots$. Then

$$\max_{\xi \in \Xi,\, \gamma \in \Gamma} f(\xi) g(\gamma)(1 - \delta_{\xi,\gamma}) = \max_{\xi \in \{\xi^1, \xi^2\}} \max_{\gamma \in \{\gamma^1, \gamma^2\}} f(\xi) g(\gamma)(1 - \delta_{\xi,\gamma}).$$

Again, δ symbolizes the Kronecker delta.

Proof. For sake of simplicity, we write $\mathrm{O} := \max_{\xi \in \Xi} \max_{\gamma \in \Gamma} f(\xi) g(\gamma)(1 - \delta_{\xi,\gamma})$. By case distinction:

1: $\xi^1 \neq \gamma^1$, i.e., $\delta_{\xi^1,\gamma^1} \neq 1$. Hence, the maximum is reached by combining the maximum values:

$$\mathrm{O} = f(\xi^1) g(\gamma^1)$$

2: $\xi^1 = \gamma^1$, i.e., $\delta_{\xi^1,\gamma^1} = 1$. Thus, there are two candidate combinations which

could maximize O:

$$\mathrm{O} = \max\left\{f(\xi^1)g(\gamma^2)\ ,\ f(\xi^2)g(\gamma^1)\right\}$$

This completes the proof. □

Corollary 6.4. Let $\gamma^* := \arg\max_{\gamma\in\Gamma}\max_{\xi\in\Xi} f(\xi)g(\gamma)(1-\delta_{\xi,\gamma})$. Then

$$\max_{\gamma\in\Gamma\setminus\{\gamma^*\}}\max_{\xi\in\Xi} f(\xi)g(\gamma)(1-\delta_{\xi,\gamma}) = \max_{\gamma\in\{\gamma^1,\gamma^2,\gamma^3\}\setminus\{\gamma^*\}}\max_{\xi\in\{\xi^1,\xi^2\}} f(\xi)g(\gamma)(1-\delta_{\xi,\gamma})$$

results from Lemma 6.3.

Theorem 6.5. Let $W_{\boldsymbol{X}}(\mathring{\mathrm{A}}_{\boldsymbol{r}})$ be the WA corresponding to a Regular Expression $\boldsymbol{r}$. For any state index j and any time step t

$$\begin{aligned}\alpha_t^1(j) &:= \alpha_t\left(q_j^{\xi_t^1(j)}\right)\\ &= \max\left\{\max_{q_i\in\mathrm{pre}(q_j)}\max_{\xi\in\Xi_{t-1}(i)}\max_{\gamma\in\Gamma_t(i,j)}\alpha_{t-1}(q_i^\xi)y_{t,\gamma}(1-\delta_{\xi,\gamma}),\ \max_{\gamma\in\mathbb{A}}\alpha_{t-1}(q_j^\gamma)y_{t,\gamma}\right\}\end{aligned}$$

Analogously

$$\begin{aligned}\alpha_t^2(j) &:= \alpha_t\left(q_j^{\xi_t^2(j)}\right)\\ &= \max\left\{\max_{q_i\in\mathrm{pre}(q_j)}\max_{\xi\in\Xi_{t-1}(i),\gamma\in\Gamma_t(i,j)\setminus\{\xi_t^1(j)\}}\alpha_{t-1}(q_i^\xi)y_{t,\gamma}(1-\delta_{\xi,\gamma}),\right.\\ &\qquad\left.\max_{\gamma\in\mathbb{A}\setminus\{\xi_t^1(j)\}}\alpha_{t-1}(q_j^\gamma)y_{t,\gamma}\right\}\\ \alpha_t^\varnothing(j) &:= \alpha_t(q_j^\varnothing)\\ &= y_{t,\varnothing}\max_{\xi\in\Xi_{t-1}(j)}\alpha_{t-1}(q_j^\xi)\end{aligned}$$

Interpretation: Since $\Xi_{t-1}(i)$ contains the two most likely labels $\xi_t^1(i), \xi_t^2(i)$ and the NaC, the two greatest states values $\alpha_t^{1/2}(j)$ can in most cases be calculated using only the previous $\alpha_{t-1}^{1/2}(i)$ and those of the NaC. Only the arcs continuing the previous label (dashed in Figure 6.1) need more than the two most likely alphas.

Proof. The claim for $\alpha_t^\varnothing(j)$ is a direct consequence from Eq. (6.4.2) and the definition of $\Xi_{t-1}(i)$.

For fixed i and j, Lemma 6.3 applies here with $f(\xi) := \alpha_{t-1}(q_j^\xi)$ and $g(\gamma) := y_{t,\gamma}$ such that

$$\max_{\xi\in\mathbb{A}'}\max_{\gamma:q_j\in\delta(q_i,\gamma)}\alpha_{t-1}(q_j^\xi)y_{t,\gamma}(1-\delta_{\xi,\gamma}) = \max_{\xi\in\Xi_{t-1}(i)}\max_{\gamma\in\Gamma_t(i,j)}\alpha_{t-1}(q_j^\xi)y_{t,\gamma}(1-\delta_{\xi,\gamma}).$$

$$\alpha_t^1(j) := \max_{\gamma \in \mathbb{A}} \alpha_t\left(q_j^\gamma\right)$$

with Eq. (6.4.1)

$$\begin{aligned}
&= \max\left\{ \max_{\gamma\in\mathbb{A}} \max_{q_i : q_j \in \delta(q_i,\gamma)} \max_{\xi\in\mathbb{A}'} \alpha_{t-1}(q_i^\xi) y_{t,\gamma}(1-\delta_{\xi,\gamma}),\ \max_{\gamma\in\mathbb{A}} \alpha_{t-1}(q_j^\gamma) y_{t,\gamma} \right\} \\
&= \max\left\{ \max_{q_i\in \mathrm{pre}(q_j)} \max_{\xi\in\Xi_{t-1}(i),\gamma\in\Gamma_t(i,j)} \alpha_{t-1}(q_i^\xi) y_{t,\gamma}(1-\delta_{\xi,\gamma}),\ \max_{\gamma\in\mathbb{A}} \alpha_{t-1}(q_j^\gamma) y_{t,\gamma} \right\}
\end{aligned}$$

Analogously,

$$\max_{\xi\in\mathbb{A}'} \max_{\{\gamma : q_j \in \delta(q_i,\gamma)\}\setminus\{\xi_t^1(j)\}} \alpha_{t-1}(q_j^\xi) y_{t,\gamma}(1-\delta_{\xi,\gamma}) := \max_{\xi\in\Xi_{t-1}(i),\gamma\in\Gamma_t(i,j)\setminus\{\xi_t^1(j)\}} \alpha_{t-1}(q_j^\xi) y_{t,\gamma}(1-\delta_{\xi,\gamma})$$

results from Corollary 6.4 which, finally, yields the proposition. □

Consider a transition from q to $\hat{q}$ of the original Automaton $A_{\boldsymbol{r}}$ with $q \neq \hat{q}$. The Extended Automaton creates multiple new (sub)states from q and $\hat{q}$ depending on the number of labels which lead to q and $\hat{q}$, respectively. Theorem 6.5 states that we only need to consider three substates from q and the three most likely characters which move q to $\hat{q}$ per time step. If there is an analogous proposition for loop transitions from q to q, there is no need to consider all substates of the Extended Automaton.

Unfortunately, pruning any transition except those starting in $q_i^{\xi_t^1}$, $q_i^{\xi_t^2}$ and $q_i^{\oslash}$ also for loop transitions prunes valid paths. The next theorem states that this should not happen too often in practice. Therefore, let $\tilde{\alpha}$ be the approximation of α which also uses only the two most likely character states plus the NaC-state and the 3 most likely labels for reading consecutive identical labels. Let $\tilde{\xi}_t(i)$ and $\tilde{\Xi}_t(i)$ denote the values which correspond to $\xi_t(i)$ and $\Xi_t(i)$, respectively, by substituting α by $\tilde{\alpha}$. Let $\Gamma_t(j)$ contain the two most likely characters $\gamma_t^k(i,j)$ for any i,k at position t. Then, we additionally restrict loops to read only characters from $\Gamma_t(j)$.

$$\begin{aligned}
\tilde{\alpha}_t^1(j) :&= \tilde{\alpha}_t\left(q_j^{\tilde{\xi}_t^1(j)}\right) \\
&= \max\Bigg\{ \max_{q_i\in\mathrm{pre}(q_j)} \max_{\tilde{\xi}\in\tilde{\Xi}_{t-1}(i)} \max_{\gamma\in\Gamma_t(i,j)} \tilde{\alpha}_{t-1}(q_i^{\tilde{\xi}}) y_{t,\gamma}(1-\delta_{\tilde{\xi},\gamma}), \\
&\qquad\qquad \max_{\gamma\in\{\tilde{\xi}_{t-1}^1(j),\tilde{\xi}_{t-1}^2(j)\}\cap\Gamma_t(j)} \tilde{\alpha}_{t-1}(q_j^\gamma) y_{t,\gamma} \Bigg\}
\end{aligned} \tag{6.4.3}$$

Analogously

$$\begin{aligned}\tilde{\alpha}_t^2(j) :&= \tilde{\alpha}_t\left(q_j^{\tilde{\xi}_t^2(j)}\right)\\ &= \max\Bigg\{\max_{q_i\in\mathrm{pre}(q_j)}\max_{\tilde{\xi}\in\tilde{\Xi}_{t-1}(i)}\max_{\gamma\in\Gamma_t(i,j)\setminus\{\tilde{\xi}_t^1(j)\}}\tilde{\alpha}_{t-1}(q_i^{\tilde{\xi}})y_{t,\gamma}(1-\delta_{\tilde{\xi},\gamma}),\\ &\qquad\max_{\substack{\gamma\in\{\tilde{\xi}_{t-1}^1(j),\tilde{\xi}_{t-1}^2(j)\}\cap\Gamma_t(j)\\ \gamma\neq\tilde{\xi}_t^1(j)}}\tilde{\alpha}_{t-1}(q_j^{\gamma})y_{t,\gamma}\Bigg\}\end{aligned} \tag{6.4.4}$$

$$\begin{aligned}\tilde{\alpha}_t^{\oslash}(j) :&= \tilde{\alpha}_t(q_j^{\oslash})\\ &= y_{t,\oslash}\max_{\tilde{\xi}\in\tilde{\Xi}_{t-1}(j)}\tilde{\alpha}_{t-1}(q_j^{\tilde{\xi}})\end{aligned} \tag{6.4.5}$$

Theorem 6.6. Let $\boldsymbol{\pi}^* = \boldsymbol{\pi}^*(\boldsymbol{r})$ be the most likely feasible label sequence with respect to the Regular Expression $\boldsymbol{r}$. Assume the following conditions:

(a) $\forall t : (\pi_t^*, \ldots, \pi_{t+n-1}^*) = a^n \in \mathbb{A}^n \Rightarrow n \leq 2$ (i.e., $\boldsymbol{\pi}^*$ contains at most 2 consecutive identical labels from $\mathbb{A}$)
(b) $\forall t : |\{a \in \mathbb{A} : y_{t,a} \geq y_{t,\oslash}\}| < 3$ (the NaC is one of the three most likely labels at each position)

Then, $\max_{q_i\in F}\tilde{\alpha}_T^1(i) = \max_{q_i\in F}\alpha_T^1(i) = \mathrm{P}(\boldsymbol{\pi}^*|\boldsymbol{X})$.

Interpretation: The approximation $\max_{q_i\in F}\tilde{\alpha}_T^1(i)$ yields the same probability as $\max_{q_i\in F}\alpha_T^1(i)$ under the above assumptions. Since $\tilde{\alpha}_t^1(j)$, $\tilde{\alpha}_t^2(j)$ and $\tilde{\alpha}_t^{\oslash}(i)$ only depend on $\tilde{\alpha}_{t-1}^1(i)$, $\tilde{\alpha}_{t-1}^2(i)$ and $\tilde{\alpha}_{t-1}^{\oslash}(i)$ of predecessor states q_i of q_j, the calculation of the other sub-states q_i^l with $l \notin \{\xi_t^1(i), \xi_t^2(i), \oslash\}$ can be omitted.

Proof. Note, that the approximation is exact for states reached by transitions reading only one or two labels since in this case only transitions with $\lambda(q,\gamma,q',t) = 0$ are pruned.

Let $\boldsymbol{\pi}^*$ be the most likely feasible label sequence with respect to the Regular Expression $\boldsymbol{r}$. Further, let $q_{i_1}^{\pi_1^*}, \ldots, q_{i_T}^{\pi_T^*}$ be the accepting state sequence. That means,

$$\forall t: \quad \alpha_t\left(q_{i_t}^{\pi_t^*}\right) = \alpha_{t-1}\left(q_{i_{t-1}}^{\pi_{t-1}^*}\right) y_{t,\pi_t^*}.$$

For a specific $t > 1$, it remains to show that $\tilde{\alpha}_t\left(q_{i_t}^{\pi_t^*}\right) = \alpha_t\left(q_{i_t}^{\pi_t^*}\right)$ for $\pi_t^* = \pi_{t-1}^* \in \mathbb{A}$ (and thus $\pi_{t-2}^* \neq \pi_t^* \neq \pi_{t+1}^*$ due to the Assumption (a)). Instead of $\pi_t^* \in \mathbb{A}$, Eqs. (6.4.3) and (6.4.4) restrict π_t^* to be an element of:

(1) $\pi_t^* = \pi_{t-1}^* \in \{\tilde{\xi}_{t-1}^1(i_{t-1}), \tilde{\xi}_{t-1}^2(i_{t-1})\}$ and
(2) $\pi_t^* \in \Gamma_t(i_t)$.

Thus, if (1) and (2) hold, $\tilde{\alpha}_t\left(q_{i_t}^{\pi_t^*}\right) = \alpha_t\left(q_{i_t}^{\pi_t^*}\right)$ assumed that $\tilde{\alpha}_{t-1}\left(q_{i_{t-1}}^{\pi_{t-1}^*}\right) = \alpha_{t-1}\left(q_{i_{t-1}}^{\pi_{t-1}^*}\right)$.

If $\pi_t^* \notin \Gamma_t(i_t)$, the substitution of π_t^* by $\oslash$ would yield a feasible ($\pi_t^* \neq \pi_{t+1}^*$, Assumption (a)), more likely (Assumption (b)) label sequence which is a contradiction to the definition of $\boldsymbol{\pi}^*$. For the same reason $\pi_{t-1}^* \in \Gamma_t(i_{t-1})$.

If $\pi_t^* = \pi_{t-1}^* \notin \{\tilde{\xi}_{t-1}^1(i_{t-1}), \tilde{\xi}_{t-1}^2(i_{t-1})\}$, then there are $q_{i_{t-1}}^{a_1}$ and $q_{i_{t-1}}^{a_2}$ s.t. $\tilde{\alpha}_{t-1}(q_{i_{t-1}}^{a_1}) \geq \tilde{\alpha}_{t-1}(q_{i_{t-1}}^{a_2}) \geq \tilde{\alpha}_{t-1}\left(q_{i_{t-1}}^{\pi_{t-1}^*}\right)$. Since $\pi_{t-1}^* \in \Gamma_t(i_{t-1})$, $a_i \notin \Gamma_{t-1}(i_{t-1})$ for some $i \in \{1, 2\}$.

Now, $\tilde{\alpha}_{t-1}(q_{i_{t-1}}^{a_i})$ can result from appending a new label or from continuing the previews: If $\tilde{\alpha}_{t-1}(q_{i_{t-1}}^{a_i}) = \tilde{\alpha}_{t-2}(q_k^\ell) y_{t-1,a_i}$ with $q_{i_{t-1}}^{a_i} \neq q_k^\ell$,

$$\tilde{\alpha}_{t-2}(q_k^\ell) y_{t-1,\oslash} \geq \tilde{\alpha}_{t-1}(q_{i_{t-1}}^{a_i}) \geq \tilde{\alpha}_{t-1}\left(q_{i_{t-1}}^{\pi_{t-1}^*}\right)$$

since $y_{t-1,\oslash} > y_{t-1,a_i}$. That means, there is a prefix which can be feasibly continued by $\boldsymbol{\pi}_{t:T}^*$ which is more likely than $\boldsymbol{\pi}_{1:t-1}^*$. This is a contradiction to the definition of $\boldsymbol{\pi}^*$.

But $\tilde{\alpha}_{t-1}(q_{i_{t-1}}^{a_i}) = \tilde{\alpha}_{t-2}(q_{i_{t-1}}^{a_i}) y_{t-1,a_i}$ contradicts the definition of $\tilde{\alpha}$ since $a_i \notin \Gamma_{t-1}(i_{t-1})$. Hence, $\pi_{t-1}^* \in \{\tilde{\xi}_{t-1}^1(i_{t-1}), \tilde{\xi}_{t-1}^2(i_{t-1})\}$. □

Remark 6.7. $\tilde{\alpha}_t(q)$ differs from $\alpha_t(q)$ only for states q with incoming transitions which read more than 2 characters. We call these states and transitions *critical*.

The conditions of Theorem 6.6 are not unlikely to occur in Recurrent Neural Networks trained with CTC. The NaC is always very probable (Assumption (b)) and the likelihoods of other labels are often very spiky (Assumption (a)), i.e., one rarely observes more than two consecutive identical labels in the best path except for the NaC. (In Bluche et al. [2015], they call this *the dominance of blank predictions*.)

Remark 6.8. Due to Theorem 6.5 and 6.6, the description of the proposed algorithm is easier with the Automaton $A_{\boldsymbol{r}}$ directly from Thompson's Construction instead of describing it based on the Extended Automaton. We only need to compute three values for each of the states of $A_{\boldsymbol{r}}$. Furthermore, the three most likely labels per transition from q_i to q_j and per position can be preselected in advance which further reduces the access time if several transitions read the same set of labels. Algorithm 8 calculates the most likely path through the Automaton efficiently using the above results.

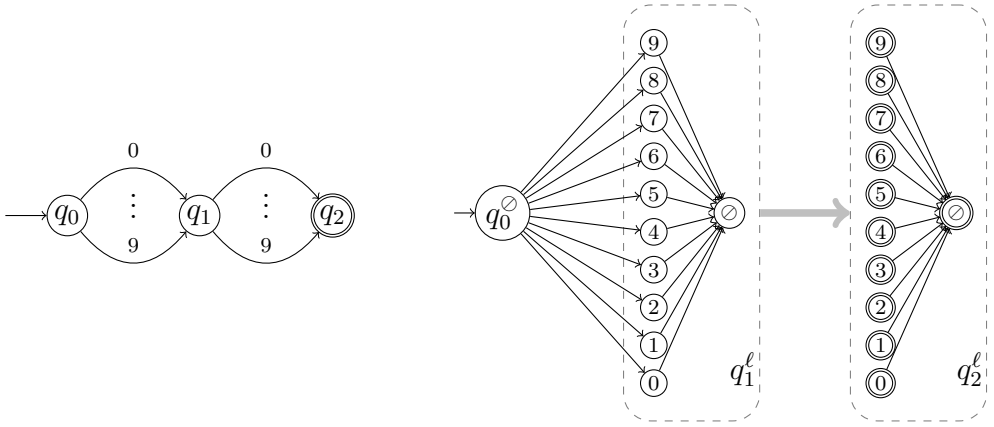
Example 6.9. For $\boldsymbol{r} := [0-9]\{2\}$, the resulting Automata are given in Figure 6.2. Algorithm 3 needs to compute and save $\alpha_t(q_t^\ell)$ for each of the 23 states q_t^ℓ of $\mathring{\mathrm{A}}_{\boldsymbol{r}}$ per position t. Calculating the $\tilde{\alpha}$ from Eq. (6.4.3) - (6.4.5), we need to compute only 7 values (in state q_0 only $\alpha_t^\oslash(0)$ is calculated). The precise gain depends on the Regular Expression.

If for a specific Regular Expression there remain only q_i^l for any i in Q since the other sub-states are not reachable, the number of multiplications will be equal.

Algorithm 8: RegExDecoder

input : $A_r = (Q, \mathbb{A}, \delta, q_0, F)$
output: $\max_{q \in F} \max\{\alpha_T^1(q), \alpha_T^{\oslash}(q)\}$

for $t \leq T$, $q_i \in Q$, $\theta \in \{1, 2, \oslash\}$ **do**
  $\alpha_t^\theta(i) \leftarrow 0$;
  $\xi_t^\theta(i) \leftarrow \varepsilon$;
$\alpha_0^1(0) \leftarrow 1$;
for $t = 1$ **to** T **do**
  for $q_j \in Q$ **do**
    for $q_i \in \mathrm{pre}(q_j)$ **do**
      `/* Appending new chars of the trans. from` q_i `to` q_j `*/`
      for $a \in \Gamma_t(i, j)$ **do**
        $\eta \leftarrow y_{t,a} \max\{\alpha_{t-1}^\theta(j) \mid \theta \in \{1, 2, \oslash\}, \xi_{t-1}^\theta(i) \neq a\}$;
        if $\alpha_t^1(j) < \eta$ **then**
          if $a \neq \xi_t^1(j)$ **then**
            $\alpha_t^2(j) \leftarrow \alpha_t^1(j)$;
            $\xi_t^2(j) \leftarrow \xi_t^1(j)$;
          $\alpha_t^1(j) \leftarrow \eta$;
          $\eta_t^1(j) \leftarrow a$;
        else if $\alpha_t^2(j) < \eta$ *and* $a \neq \xi_t^1(j)$ **then**
          $\alpha_t^2(j) = \eta$;
          $\xi_t^2(j) = a$;
      `/* Continuing previous labels of` q_j `*/`
      for $a \in \Gamma_t(j)$ **do**
        if $a = \xi_{t-1}^\theta(j)$ $(\theta \in \{1, 2\})$ **then**
          $\eta \leftarrow y_{t,a} \alpha_{t-1}^\theta(j)$;
          if $a \neq \xi_t^1(j)$ **then**
            $\alpha_t^2(j) \leftarrow \alpha_t^1(j)$;
            $\xi_t^2(j) \leftarrow \xi_t^1(j)$;
          $\alpha_t^1(j) \leftarrow \eta$;
          $\eta_t^1(j) \leftarrow a$;
        else if $\alpha_t^2(j) < \eta$ *and* $a \neq \xi_t^1(j)$ **then**
          $\alpha_t^2(j) = \eta$;
          $\xi_t^2(j) = a$;
    $\alpha_t^{\oslash}(j) \leftarrow \max\{\alpha_{t-1}^1(j), \alpha_{t-1}^{\oslash}(j)\}\, y_{t,\oslash}$;



(a) $A_{\boldsymbol{r}}$ for $r :=$ `[0-9]{2}`.

(b) Corresponding $\mathring{\mathrm{A}}_{\boldsymbol{r}}$. Bold arc represents any connection from q_1^ℓ to q_2^a for $\ell \neq a$ ($\ell \in \mathbb{A}'$, $a \in \mathbb{A}$).

Figure 6.2: Automaton accepting $\mathcal{L}($`[0-9]{2}`$)$ (left) and Automaton accepting any label sequence collapsing to an element from $\mathcal{L}($`[0-9]{2}`$)$ (right).

Capturing Groups

As already mentioned, information about a part of the Regular Expression can be crucial. In the case of keyword spotting for example, the likelihood of the keyword determines whether or not the current spot is accepted. But also the likelihood of labels next to the keyword are important to decide whether or not the spotted word is only a part of a larger word. To associate parts of the Regular Expression with parts of the Automaton, we take advantage of the notion of Capturing Groups:

A *Capturing Group* g of a Regular Expression $\boldsymbol{r}$ is a connected part within a pair of parentheses. Thus, the group is related to those transitions of the Automaton $A_{\boldsymbol{r}}$ which directly result from the subexpression g. Hence, only if the most likely path associated with $\boldsymbol{r}$ makes use of any transition related to g, g captures some part of the current ConfMat $\boldsymbol{Y}$. Then, the corresponding captured label sequence is the part of the most likely label sequence $\boldsymbol{\pi}^*(\boldsymbol{r})$ read by the subautomaton related to g. In a straightforward way, one calculates the probability or the bounds (start and end position) of g according to $\boldsymbol{\pi}^*(\boldsymbol{r})$.[3]

6.5 Experiments

The aim of this section is to show that the decoding works properly and fast. We show that Algorithm 8 works correctly in practical applications and analyze situations when the heuristic of Theorem 6.6 fails by comparing our decoding result to

[3]Since the NaC is not part of the Regular Expression, one may decide whether or not the likelihood calculation and the optimal label sequence include the starting and trailing NaC-labels.

Table 6.1: Statistics over the text recognition experiment: "size" denotes the number of words of the vocabulary, "# total" (transitions) denotes the number of transitions in Automaton and "# critical" (transitions) denotes the number of transitions which read more than 2 labels. "greatest deviation - absolute" denotes the difference between the exact negative logarithmic probability and the result of the RegEx-Decoder. The "greatest deviation - relative" is the deviation divided by the exact absolute logarithmic probability.

vocabulary	size	transitions		greatest deviation	
		# total	# critical	absolute	relative
HTRtS	9273	12398	12	9.95E-14	2.1E-12
general English	21698	25997	32	9.95E-14	2.1E-12

the exact most likely label sequence. Further applications of the RegEx-Decoder can be found in Strauß et al. [2014] and Leifert et al. [2014a]. The following results are part of Strauß et al. [2016].

We did all time statistics on a laptop with Intel i7-4940MX 3.10GHz CPU, 32GB RAM and SSD.

Text Recognition

First, we show that our approximation is reasonable for practical applications such as the HTRtS competition from the ICFHR2014 (see Sánchez et al. [2014]). The data consists of 400 handwritten pages. We train on 350 pages and validate on 50 pages. The validation set is also used to evaluate the decoding. Each page consists of several lines of text including words, punctuations, numbers and symbols. The Neural Network used in Strauß et al. [2014] generates the ConfMats. We compare the most likely word of a vocabulary obtained by the RegEx-Decoder[4] with the result of the Vocabulary Automaton from Section 5.3.1. For this purpose, the RegEx-Decoder is used to split the ConfMats into regions of words and regions containing spaces, numbers etc. The evaluation is done on the resulting 4657 submatrices representing the word regions. These matrices correspond (more or less) to ConfMats of subimages of single words. We use two vocabularies: one containing 9273 words (generated from HTRtS data) and one containing 21698 words (a modern, general vocabulary build from two million English sentences from `http://corpora.uni-leipzig.de/`).

Table 6.1 shows the deviation of the negative logarithmic likelihood of the RegEx-Decoder and the exact decoding. Clearly, the deviation is negligible. There is an

[4]The used Automaton is a DAFSA of Section 2.2.

intersection of both vocabularies which includes especially the most frequent words. Thus, it is not surprising that both vocabularies show the same deviation since both extrema (the greatest absolute and relative deviation) appear for the same words ("General" and "of"). Since the number of critical transitions is very small, we expected a small divergence due to our approximation. In fact there is *no additional confusion of words* caused by our approximation. Thus, the experiment shows that the approximation of Theorem 6.6 is reasonable in practical applications with few critical transitions.

We evaluated the impact of the decoder empirically on the HTRtS test set. We decreased the word error rate by 3 percentage points compared to the best path decoding (i.e., $\mathcal{F}(\boldsymbol{\beta})$) of the entire line (from 50.89% to 48.06%) just by defining an appropriate Regular Expression for the expected line structure without any vocabulary. Including a vocabulary, we further decreased the WER to 33.90%.

Number Recognition

The next experiment involves artificially generated writings and investigates the correctness of Alg. 8 in case of a relatively large number of critical transitions. By Remark 6.7, we know that errors only appear for states reached by transitions which read more than two labels. We enforce this condition by searching only for digits. Thus, any transition is critical since these transitions read more than two labels. To enforce further continuation errors[5], we vary the number of digits actually depicted in the image while the search pattern remains 3 to 5 digits (i.e., the Regular Expression is `[0-9]{3,5}`). If the number of digits actually displayed in the image is greater than 5, the decoder has to suppress valid digits which also promotes errors since the optimal label sequence provides a label for each row of the ConfMat.

We vary the number of digits from 4 to 9. For each number of digits, we generate 10,000 synthetic writings. The digits are narrowly written to enforce further confusions (see Figure 6.3). The resulting images work as input to four Neural Networks with different number recognition expertise. We will compare the decoding results over the ConfMats generated by these NNs. The RegEx-Decoder searches in these ConfMats for the most likely number with 3 to 5 digits. The resulting number and probability is compared with the result from a traditional Vocabulary Automaton as in Section 5.3.1 using a vocabulary of all numbers with 3 to 5 digits. Any difference in the resulting optimal label sequence is regarded as an error.

Table 6.2 shows the number of errors per network and digits in the image. The more the algorithm is forced to suppress digits the more errors occur. For 4 and 5 digits there is no force to suppress any written digit since the corresponding Automaton is allowed to accept the ground truth. The errors are negligible in this case. From

[5]Remember that one of the conditions for errors is that the optimal label sequence $\boldsymbol{\pi}$ contains more multiple consecutive identical letters.

Figure 6.3: Artificial writings of two numbers for the number recognition task.

Table 6.2: Number recognition task: Number of differences in the most likely label sequences of the RegEx-Decoder and the exact decoding for different NNs and different number of digits in the image but a constant Regular Expression of `[0-9]{3,5}`.

	4	5	6	7	8	9
net1	0	0	1	12	9	27
net2	0	0	24	27	40	40
net3	0	0	6	3	4	4
net4	0	1	4	4	7	7

6 to 9 digits, the number of errors increases.

Even if there is a relatively high number of critical transitions, there will only be little error if the Regular Expression fits to the image content. If the Regular Expression does not fit to the number of digits in the image, there will be a moderate risk of generating additional confusion errors due to our approximation. However, even under exact decoding the best feasible label sequence then has a very low probability which will be further underestimated by the approximation. Hence, the approximation will likely not be harmful here since the decoding result can either be rejected immediately or it is unlikely to be of any significance in downstream processing steps.

Figure 6.4 shows the required decoding time for the images as describe above applied to different decoding algorithms and for different numbers of digits in the image. The RegEx-Decoder needs between 0.19 ms and 0.28 ms per ConfMat on average depending on the number of digits. The conventional Vocabulary Automaton needs at least 4.68 ms per ConfMat since it has to calculate the probabilities of more or less all numbers with the specific number of digits under consideration. It naturally reuses already calculated probabilities whenever the beginnings are the same. Additionally, we applied the limit procedure of Section 5.3.2. Even with this speed-up mechanism the RegEx-decoder is more than 22 times faster. The running time for the A^* Search is growing more than exponentially in the number of digits since the likelihood of $\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X}) \ll 1$ for $n > 5$ digits. For 4 and 5 digits it is likely that the best path $\boldsymbol{\beta}$ is already the decoding result such that the bound is tight and prunes any path.

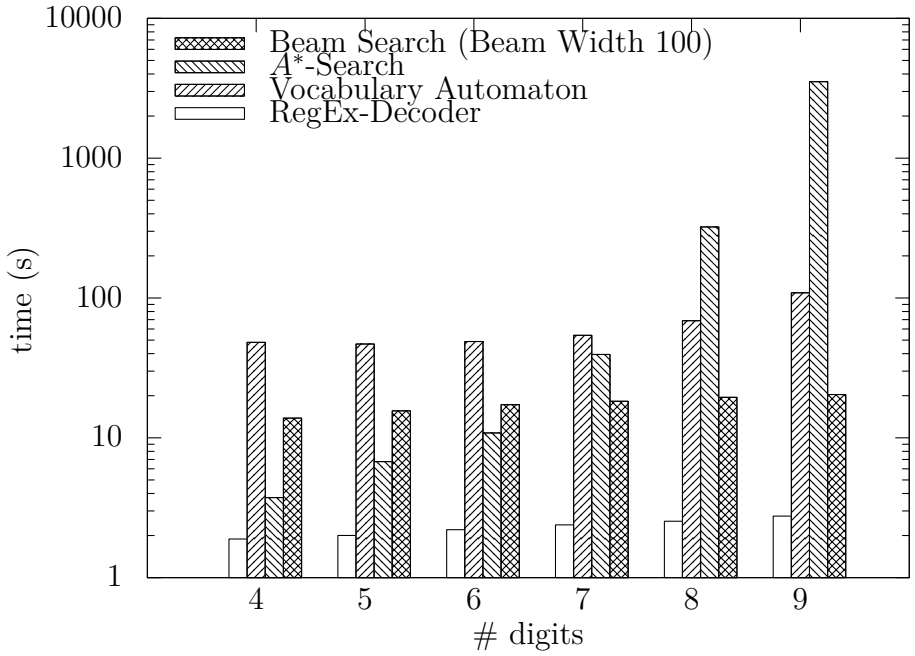


Figure 6.4: Decoding times of different decoding algorithms for the number recognition task (10,000 ConfMats) averaged over all four NNs.

The Beam Search with Beam Width 100 needs almost seven times more time for the calculation than the RegEx-Decoder. A point of criticism might be that we use no independent implementation to compare the time complexity and we may not have implemented the Beam Search algorithm optimally. Let us count the multiplications: Beam Search with Beam Width 100 calculates for each of the 100 prefixes at each time step 11 new prefixes (one for each digit plus one adding the NaC) and thus 1100 multiplications per time step in total in the worst case. The RegEx-Decoder needs for each of the 5 transitions at most 6 multiplications to update α^1 and α^2 plus one additional for $\alpha^{\oslash}$.[6]Thus, we have 35 multiplications in total. Therefore, our theoretical analysis rather indicates that the RegEx-Decoder is implemented suboptimally since Beam Search needs 31 times more multiplications but the computation time is only seven times higher. Although Beam Search with Beam Width 100 is much slower, it yields significantly more errors (round about 40 errors on average if the ground truth are 4 or 5 digits). To get a comparable performance for the experiments with 4 and 5 digits, we needed a Beam Width of at least 1000 in our experiments[7].

6.6 Combined Vocabulary and Regular Expression Constraints

In Section 6.4, we describe a method to calculate the most likely label sequence constrained to Regular Expressions. This method is only efficient if the number of transitions in the corresponding Automaton is not too large. If the Regular Expression is very complex since it, e.g., contains a vocabulary, the number of transitions is huge such that a complete search of the search space is intractable even for the heuristic proposed in Theorem 6.6. To prune the search space, we suggest a combination of Beam Search (introduced in Section 6.3.2) and the method from Section 6.4. As a result, we will be able to incorporate large vocabularies[8] into the decoding.

The algorithm of Section 6 guarantees (under the assumptions of Theorem 6.6) the calculation of the two most likely prefixes reaching any state which do not end on the same character. For a large number of states, many of these will never contribute to the result or the contribution is limited to a few time steps. Hence, it is an intuitive approach to reduce the number of states calculated at a certain time step to stay efficient.

Instead of expanding each state at each time step, we only keep the most promising states and omit all the others. Algorithm 9 is the natural extension of Algorithm 8

[6]Compared to Strauß et al. [2016], we revised the description of the decoder which led to a decrease of the required multiplications.

[7]The running time increases from round about 15 sec to 115 sec for all 10,000 ConfMats.

[8]Large in terms of several thousands of words.

to Beam Search.

If the Beam Width n is sufficiently great, Algorithm 9 computes the same paths as Alg. 8 and thus yields the same results e.g. for the Number Recognition experiment of Section 6.5.

6.7 Conclusion

In this chapter, we considered Regular Expressions decoding constraints and effective decoding methods. We show how to exploit Finite State Automata to find the most likely feasible label sequence of a Regular Language. A further analysis of the decoding procedure yields a pruning strategy of the search space such that it also works fast for complex Regular Expressions and many ConfMats. We also propose an approximation which is shown to be exact under conditions which are commonly satisfied for CTC-trained NNs. This theoretical result was confirmed by experiments. As a main result, we showed that the decoder is applicable in practical scenarios. Even if the approximation fails to produce exact results, it is likely that the ground truth does not fit to the Regular Expression. This results in a low probability decoding result further underestimated by our approximation which should not be harmful in most applications.

The proposed speed-ups only work for the path probability $\mathrm{P}(\boldsymbol{\pi}|\boldsymbol{X})$ (instead of the CTC probability). If the decoder should return the exact probability, all paths contribute to the result and, thus, cannot be skipped. Hence, speed-ups seem to be hard. Additionally, we have to take care about distinct paths through the Automaton accepting the same label sequence. An *Unambiguous FSA* or even a DFA is required to ensure that the Automaton accepts any label sequence only once. We already discussed the disadvantages of DFAs in Chapter 2.

There are plenty of applications for the proposed algorithm. The method can be applied e.g. to keyword spotting (compare Section 7.2) but also patterns of other information retrieval tasks can be described conveniently.

Algorithm 9: RegExDecoderBeamSearch

input : ConfMat $\boldsymbol{Y}$, $A_r = (Q, \mathbb{A}, \delta, q_0, F)$, Beam Width n
output: $\max_{q \in F} \max\{\alpha_T^1(q), \alpha_T^{\oslash}(q)\}$

for $t \leq T$, $q_i \in Q$, $\theta \in \{1, 2, \oslash\}$ **do**
  $\alpha_t^\theta(i) \leftarrow 0$;
  $\xi_t^\theta(i) \leftarrow \varepsilon$;
$\alpha_0^1(0) \leftarrow 1$;
$\mathbb{L} \leftarrow \{q_0\}$;
for $t = 1$ **to** T **do**
  $\overline{\mathbb{L}} \leftarrow n$ items of $\mathbb{L}$ with greatest value α_t^1;
  $\mathbb{L} \leftarrow \{\}$;
  for $q_j \in Q$ **do**
    for $q_i \in \mathrm{pre}(q_j) \cap \overline{\mathbb{L}}$ **do**
      /* Appending new chars of the trans. from q_i to q_j */
      for $a \in \Gamma_t(i, j)$ **do**
        $\eta \leftarrow y_{t,a} \max\{\alpha_{t-1}^\theta(j) \mid \theta \in \{1, 2, \oslash\}, \xi_{t-1}^\theta(i) \neq a\}$;
        if $\alpha_t^1(j) < \eta$ **then**
          if $a \neq \xi_t^1(j)$ **then**
            $\alpha_t^2(j) \leftarrow \alpha_t^1(j)$;
            $\xi_t^2(j) \leftarrow \xi_t^1(j)$;
          $\alpha_t^1(j) \leftarrow \eta$;
          $\eta_t^1(j) \leftarrow a$;
        else if $\alpha_t^2(j) < \eta$ *and* $a \neq \xi_t^1(j)$ **then**
          $\alpha_t^2(j) = \eta$;
          $\xi_t^2(j) = a$;
    /* Continuing previous labels of q_j */
    for $a \in \Gamma_t(j)$ **do**
      if $a = \xi_{t-1}^\theta(j)$ $(\theta \in \{1, 2\})$ **then**
        $\eta \leftarrow y_{t,a} \alpha_{t-1}^\theta(j)$;
        if $a \neq \xi_t^1(j)$ **then**
          $\alpha_t^2(j) \leftarrow \alpha_t^1(j)$;
          $\xi_t^2(j) \leftarrow \xi_t^1(j)$;
        $\alpha_t^1(j) \leftarrow \eta$;
        $\eta_t^1(j) \leftarrow a$;
      else if $\alpha_t^2(j) < \eta$ *and* $a \neq \xi_t^1(j)$ **then**
        $\alpha_t^2(j) = \eta$;
        $\xi_t^2(j) = a$;
    $\alpha_t^{\oslash}(j) \leftarrow \max\{\alpha_{t-1}^1(j), \alpha_{t-1}^{\oslash}(j)\}\, y_{t,\oslash}$;

7 Applications

In this chapter, we apply the proposed techniques to handwritten texts. Since the words in natural language texts typically follow specific statistics, we first investigate how to integrate Language Models into the decoding process. Afterwards, we propose a keyword spotting method and also show that even full text recognition can be processed using Regular Expressions.

7.1 Incorporating a Language Model

To our experience, the NN's ability to learn dependencies of characters depends highly on the training data: If many variations appear in the trained character sequences, the ConfMat will depend only weakly on character transitions. Thus, the NN will not be able to predict reliable prior word probabilities. It rather predicts the character sequence as accurately as possible. To also incorporate prior word probabilities, we borrow some basic ideas from domain adaptation (see, e.g., [Jiang, 2008, Section 6.2.1]): Let $\mathrm{P}_{\mathcal{S}}(Y \mid X)$ and $\mathrm{P}_{\mathcal{T}}(Y \mid X)$ be two conditional probability distributions of the random variables X and Y. Domain adaptation then tries to estimate the unknown target distribution $\mathrm{P}_{\mathcal{T}}$ from the known source distribution $\mathrm{P}_{\mathcal{S}}$. In our case, $\mathrm{P}_{\mathcal{S}}$ is the Neural Network probability of a character sequence $\boldsymbol{z}$ given the image $\boldsymbol{X}$ and $\mathrm{P}_{\mathcal{T}}$ is the corresponding "true" probability. For the next theorem, we attach the parameter $\boldsymbol{W}$ again to distinguish between the probability given by the NN and those underlying the data.

Theorem 7.1. Let $\mathcal{X} \subset \mathbb{A}^*$ with $|\mathcal{X}| < \infty$ and let $\boldsymbol{X}$ be the input image. Assume two discrete conditional probability distributions $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z} \mid \boldsymbol{X})$ and $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{X}; \boldsymbol{W})$. Assume further $\mathrm{P}_{\mathcal{S}}(\boldsymbol{X} \mid \boldsymbol{z}; \boldsymbol{W}) = \mathrm{P}_{\mathcal{T}}(\boldsymbol{X} \mid \boldsymbol{z})$. Then for any $\boldsymbol{z} \in \mathcal{X}$

$$\mathrm{P}_{\mathcal{T}}(\boldsymbol{z} \mid \boldsymbol{X}) = \frac{1}{N} \quad \frac{\mathrm{P}_{\mathcal{T}}(\boldsymbol{z})}{\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{W})} \mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{X}; \boldsymbol{W}) \tag{7.1.1}$$

where

$$N = \sum_{\overline{\boldsymbol{z}} \in \mathcal{X}} \frac{\mathrm{P}_{\mathcal{T}}(\overline{\boldsymbol{z}})}{\mathrm{P}_{\mathcal{S}}(\overline{\boldsymbol{z}} \mid \boldsymbol{W})} \mathrm{P}_{\mathcal{S}}(\overline{\boldsymbol{z}} \mid \boldsymbol{X}; \boldsymbol{W}). \tag{7.1.2}$$

The proof is a direct consequence of Bayes' law for $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z} \mid \boldsymbol{X})$ and $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{X}; \boldsymbol{W})$.

For a given Neural Network, $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{X}; \boldsymbol{W})$ is given by CTC. Theorem 7.1 allows us to correct $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{X}; \boldsymbol{W})$ by incorporating a Language Model $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z})$ such that we obtain a better estimate for the "true distribution" $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z} \mid \boldsymbol{X})$. Since $\boldsymbol{z}$ is an abstract feasible character sequence which possibly could contain several words or a whole sentence, $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z})$ could be estimated by a higher order Language Model.

It remains to be shown whether or not the assumption $\mathrm{P}_{\mathcal{S}}(\boldsymbol{X} \mid \boldsymbol{z}; \boldsymbol{W}) = \mathrm{P}_{\mathcal{T}}(\boldsymbol{X} \mid \boldsymbol{z})$ is correct. Typically, NN-HMM hybrid models assume that the observations are modeled reliably. Many of these recognition systems work well in practice which may serve as an indication that the assumption is reasonable.

Fortunately, we are not restricted to any word statistics from the training set which typically is too small. Rather any Language Model capturing the *expected* statistics can be applied. In principle, the above theorem allows to train and test on different data, even data from different languages.

We have to estimate the source prior probability $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{W})$ learned by the Neural Network. Since the conditional distribution $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{X}; \boldsymbol{W})$ typically tends to have only weak impact on inter-character dependencies or even ignores them, it is reasonable to assume that $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{W})$ is a character model of low order or even constant. We will experimentally evaluate this in the context of keyword spotting.

Connection to HMMs Language Models have been integrated into HMM-Decoders for a long time with great success. Hybrid RNN-HMM models, where the RNN is trained in a discriminative manner, calculate the emission probability of an observation $\boldsymbol{x}$ under the HMM state s by

$$\mathrm{p}(\boldsymbol{x} \mid s) = \frac{\mathrm{P}(s \mid \boldsymbol{x})\,\mathrm{p}(\boldsymbol{x})}{\mathrm{P}(s)}$$

where $\mathrm{P}(s \mid \boldsymbol{x})$ is given by the NN (see e.g. Doetsch et al. [2014]). Consider an HMM which models the CTC topology, i.e., the support of the HMM which accepts (only) $\boldsymbol{z}$ is $\mathring{\mathrm{A}}_{\boldsymbol{z}}$. Let $\boldsymbol{x}_t$ be the input $\boldsymbol{X}$ at position t. Then the most likely state sequence $\boldsymbol{s}^*_{1:T}$ conditioned on $\boldsymbol{X}$ is decoded by

$$\begin{aligned}
\boldsymbol{s}^*_{1:T} &= \arg\max_{\boldsymbol{s}_{1:T}} \mathrm{P}(\boldsymbol{s}_{1:T} \mid \boldsymbol{X}) \\
&= \arg\max_{\boldsymbol{s}_{1:T}} \frac{\mathrm{p}(\boldsymbol{X} \mid \boldsymbol{s}_{1:T})\,\mathrm{P}(\boldsymbol{s}_{1:T})}{\mathrm{p}(\boldsymbol{X})} \\
&= \arg\max_{\boldsymbol{s}_{1:T}} \prod_{t=1}^{T} \frac{\mathrm{P}(s_t \mid \boldsymbol{x}_t)\,\mathrm{p}(\boldsymbol{x}_t)\,\mathrm{P}(\boldsymbol{s}_{1:T})}{\mathrm{p}(\boldsymbol{X})\,\mathrm{P}(s_t)} && (7.1.3) \\
&= \arg\max_{\boldsymbol{s}_{1:T}} \prod_{t=1}^{T} \frac{\mathrm{P}(s_t \mid \boldsymbol{x}_t)\,\mathrm{P}(\boldsymbol{s}_{1:T})}{\mathrm{P}(s_t)} && (7.1.4)
\end{aligned}$$

where $\mathrm{P}(s_t)$ corresponds to the state prior and $\mathrm{P}(\boldsymbol{s}_{1:T})$ is given by the transition probabilities including word (lexical) and LM (syntactical) probabilities as described in Section 4.2.4. If we interpret $\prod_t \mathrm{P}(s_t)$ as prior probability learned by the NN, Eq. (7.1.1) and Eq. (7.1.4) yield the same state sequence and, hence, they share a common probabilistic background.

If $\mathrm{P}(\boldsymbol{z}) = c$ for some c and for each $\boldsymbol{z} \in \bigcup_{n=1}^{N} \mathcal{V}^n$ for some sufficiently great N, we obtain Eq. (4.4.3) (up to a constant) which is the fundamental equation of the CTC-Token-Passing Algorithm.

7.2 Keyword Spotting

The term keyword spotting is ambiguously used for detecting words in both spoken and written language. We exclusively focus on identifying regions in a collection of handwritten texts which contain a specific given keyword. The keyword may either be given as a string (query-as-string) or as an isolated example "snippet" (query-as-example). Besides reading the textual information, another standard approach to keyword spotting uses image processing methods and computes the similarity on handcrafted features (see Pratikakis et al. [2014]). Typically, these approaches are *training free* and thus they are the preferred approach if there is no or only few training data available (Rath and Manmatha [2007]). These approaches are limited to the query-by-example scenarios. If there are transcriptions available to train a model to provide $\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$, the *training based* systems outperform the training free systems (see e.g. Vidal et al. [2015], Puigcerver et al. [2015a]). Many of the training-free approaches require a previous segmentation into single words and suffer from segmentation errors. Training based approaches based on HMMs or RNNs usually do not need a predefined segmentation. We focus on the query-by-string method into single words. The proposed procedure can be adapted to query-by-example conditions by simply reading the input example and proceed as if the string was given.

7.2.1 Previous Work

HMMs HMMs are a state-of-the-art approach to keyword spotting. The preferred procedure is to model the keyword as a concatenation of the corresponding character HMMs. Filler models cover the uninteresting parts of the text line. This model consists of a complete digraph where the nodes correspond to the labels. Between the filler and the keyword there is a space state to guarantee that the keyword is not part of any larger word. Figure 7.1 shows a scheme of this HMM model. Let $\boldsymbol{z}$

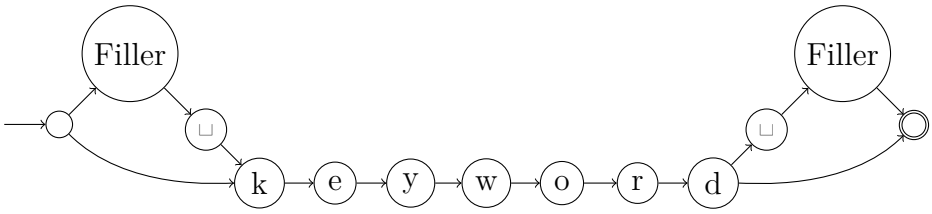


Figure 7.1: Standard HMM keyword model as concatenation of character HMMs

be the keyword. The standard score for $\boldsymbol{z}$ as subsequence contained in $\boldsymbol{X}$ is

$$\frac{\max_{\boldsymbol{w}\in\mathcal{L}(.*\boldsymbol{z}.*)} \ln \mathrm{p}(\boldsymbol{w}, \boldsymbol{X}) - \max_{\hat{\boldsymbol{w}}\in\mathcal{L}(.*)} \ln \mathrm{p}(\hat{\boldsymbol{w}}, \boldsymbol{X})}{|\boldsymbol{z}|} \tag{7.2.1}$$

where $.*$ is the Regular Expression notation of the filler. The denominator normalizes keywords of different length by the same number of states in the keyword. In Puigcerver et al. [2015b] they derive this score from the probabilistic point of view.

Bideault et al. [2015a] compared HMM hybrid models with MLP, RNN, Conditional Random Fields or BLSTM posterior estimators concluding that the combination of BLSTMs and HMMs works best. They also reported very good results of a plain CTC-BLSTM without an HMM stage.

RNNs Frinken et al. [2012] suggested a system based on BLSTM-NNs for keyword spotting. They proposed a simplified version of the Token Passing Algorithm from Graves et al. [2009] which calculates the probability of $.*_{\text{␣}}\boldsymbol{z}_{\text{␣}}.*$ as described in Section 5.2. This leads to problems if the text line starts or ends with the keyword. To circumvent this problem, the authors "add sequence elements to the beginning and the end of each text line that represents extra white space." They compared the proposed system to a training free Dynamic Time Warping approach and an HMM system from Fischer et al. [2010] and finally outperformed the reference systems by an impressive margin.

A straight approach was proposed in Fernández et al. [2007]. The Neural Network had one output neuron for each keyword plus an additional one for the not-a-keyword label. It was trained by CTC and performed well in the context of speech recognition. The resulting likelihoods served directly as confidence measures. This is a clear advantage over previous approaches since the problem of uncertain keyword matches is shifted to the Neural Network and optimized implicitly during the training procedure. The major drawback is that, in contrast to other training based approaches, this method is limited to keywords which are trained to detect. Also the detection of infrequent words might be a problem.

7.2.2 Regular Expression Approach

Similar to Frinken et al. [2012], we assume a Neural Network trained with CTC. For an exemplary architecture of the RNN and details of the corresponding preprocessing, we refer to Strauß et al. [2016].

In the following, $\boldsymbol{z}$ denotes a single keyword. Since the text line probably contains more than one word, uninteresting parts of the text line also have to be covered: An arbitrary character in Regular Expression notation is a dot `.` and it is arbitrarily often repeated by the Kleene Star `*`. Thus, `.*`$\boldsymbol{z}$`.*` will match any occurrence of $\boldsymbol{z}$ in the line, also these where $\boldsymbol{z}$ is only part a of longer word. To circumvent the part-of-word problem, we enforce a separation of the "filler model" `.*` and the keyword by separation characters such as "`-`", "`(`" or "`␣`". Putting these characters in brackets, like `[-(␣)]`, means that each of them can be read interchangeably. Hence, the resulting Regular Expression for a single keyword is:

$$\boldsymbol{r} := \big(\texttt{.*}\big[\texttt{␣(-}\big]\big)\texttt{?}\boldsymbol{z}\big(\big[\texttt{␣)-}\big]\texttt{.*}\big)\texttt{?}$$

This procedure can be extended to n-occurrences of the keyword:

$$\boldsymbol{r} := \big(\texttt{.*[␣(-]}\big)\texttt{?}\boldsymbol{z}\big(\texttt{[␣(-]}\big(\texttt{.*[␣(-]}\big)\texttt{?}\boldsymbol{z}\big)\texttt{\{n-1\}}\big(\texttt{[␣)-].*}\big)\texttt{?}$$

The separators at the beginning or end of a line are optional while separators between two keywords are mandatory. The path probability of the keyword is given by

$$\mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}) = \max_{\substack{\boldsymbol{\pi}\in\mathcal{F}^{-1}(\boldsymbol{z}) \\ |\boldsymbol{\pi}|=e-s+1}} \prod_{t=s}^{e} y_{t,\pi_{t-s+1}},$$

where s and e are the start and end position of the submatrix within the ConfMat corresponding to the *Capturing Group* of $\boldsymbol{z}$ as described in Section 6.4. The Capturing Group directly yields the subsequence $\boldsymbol{\pi}^*_{s:e}$ corresponding to $\boldsymbol{z}$ (i.e., $\mathcal{F}(\boldsymbol{\pi}^*_{s:e}) = \boldsymbol{z}$) and the corresponding probability $\mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$. We already mentioned in Section 6.4 that $\boldsymbol{\pi}^*_{s:e}$ (or rather s and e) are not well defined since NaCs may or may not be included. To avoid ambiguity, we use the longest subsequence which includes all NaCs. Alternatively, one may use

$$\mathrm{P}_{s:e}(\boldsymbol{z} \mid \boldsymbol{X}) := \sum_{\substack{\boldsymbol{\pi}\in\mathcal{F}^{-1}(\boldsymbol{z}) \\ |\boldsymbol{\pi}|=e-s+1}} \prod_{t=s}^{e} y_{t,\pi_{t-s+1}}.$$

In contrast to the HMM score of Eq. (7.2.1), we do not normalize by the maximum likelihood of any character sequence. Since $y_{t,l} < 1$ for any t, l, the path probability typically decreases if $e - s$ increases. Thus, we prefer a score relative to the number of positions rather than relative to the number of characters of the considered word $|\boldsymbol{z}|$. We obtain $\mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z})|\boldsymbol{X})^{\frac{1}{e-s+1}}$.

The separation characters should ensure that the match is not merely a part of a larger word. Unfortunately, there is always a small likelihood of such separation characters even if the match is a smaller part of another word. To detect these cases, the match is accepted only if the likelihood of each Capturing Group corresponding to a separation character exceeds a certain threshold. We consider the likelihoods of the separators and of the keyword separately to better identify whether the match is only part of a larger word. A low likelihood of the separator immediately leads to a rejection while a combined keyword-separator probability cannot differentiate a badly written match from the part-of-larger-word case.

7.2.3 Integrating Language Models

Due to Theorem 7.1, the keyword likelihood $\mathrm{P}_{s:e}(\boldsymbol{z} \mid \boldsymbol{X})$ can be rescored by a Language Model according to Eq. (7.1.1). Since we deal with single words $\boldsymbol{z}$ in the context of keyword spotting, the Language Model $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z})$ will be a word unigram.

Typically, the vocabulary represents only a small fraction of all words of the considered language. Thus, it is impossible to sum the probabilities of all these feasible words. Hence, any computed normalization constant N must be an approximation. Words not contained in the vocabulary are called *out-of-vocabulary words (OOV words)*. A heuristic which worked well in our experiments is to include the collapsed best path $\mathcal{F}(\boldsymbol{\beta}_{s:e})$ temporarily as an OOV word into the vocabulary if it is not already contained. The prior probability of the newly added word is a previously determined constant which is equal for any OOV word.

7.2.4 Experimental Validation

The following experiment was done for the ImageCLEF 2016 Handwritten Scanned Document Retrieval Task (Villegas et al. [2016]).

Data Set and Task

The data set comprises handwritten texts from Jeremy Bentham who was an important English philosopher of the 18th century. The texts were digitalized and transcribed by the Transcribe Bentham project (see Moyle et al. [2011]).

The data set is divided into a training and development set of 363 and 433 pages including transcripts and a test set of 200 pages without transcription. For the development set, an additional list of 510 queries and the corresponding ground truth results (including geometrical information) are provided for purpose of testing & tuning the system.

The task is to search in the development and test pages for queries from a new list (of 1000 queries) including multi-word queries, OOV words and hyphenated matches. The whole query has to match in a segment of 6 consecutive lines in the correct order. The segments overlap such that one single query match appears in up to 6 segments. The goal is to return the corresponding segments for any query.

Setup

A system overview including preprocessing methods and network architecture can be found in Strauß et al. [2016]. We focus here on the decoding strategies reported there.

In contrast to $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z})$ which is reasonably a word unigram, the prior source probability $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z})$ is unclear. We estimate the character transition probability learned by the NN in three different ways:

- $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z}) = \mathrm{P}_{\mathcal{T}}(\boldsymbol{z})$: Target and source prior probabilities cancel each other such that the likelihoods $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z} \mid \boldsymbol{X})$ and $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z} \mid \boldsymbol{X})$ are equal up to the normalization. We refer to this prior scheme as `abs`.
- $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z}) \propto 1$: The prior source probability is assumed to be constant such that only information of the target domain is used.[1] We refer to this prior scheme as `prior`.
- $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z}) \propto \left(\prod_{i=1}^{|\boldsymbol{z}|} \mathrm{P}(z_i)\right)^c$: $\mathrm{P}(z_i)$ is the prior character probability of z_i and $0 < c \leq 1$.[2] We refer to this prior scheme as `da`.

In the last both schemes, `prior` and `da`, the prior probabilities are estimated up to a constant factor $1/N'$ which is basically the reciprocal of the sum of the estimates of $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z})$ over the finite set of all words $\boldsymbol{z}$. The prior normalization constant N' can be merged with the normalization constant N from Eq. (7.1.2).

To sort a vocabulary according to the probability on a specific submatrix of a given ConfMat, there is no need to calculate the normalization $1/N$. The normalization is only crucial for comparing the probabilities of two matches corresponding to different submatrices and, thus, it directly determines the position in the list of the results of all queries over the whole document. This position directly influences the evaluation measure and therefore the performance of the system. As a rule of thumb: the smaller the position of a certain match the greater the impact on the performance.

The normalization is computational expensive since it requires to sum over all feasible strings including word sequences and parts of words if the segmentation fails. In our experiments, we approximate the sum by considering only the 10 (out of

[1] This corresponds to the Language Model integration used in Graves et al. [2008] and Graves and Jaitly [2014] (see Eq. (4.4.3)).

[2] In the submitted system, this character priors are estimated on the training set and $c = 0.5$.

12740) most likely vocabulary matches and, optionally, the additional OOV word as described above. We ignore therefore many feasible combinations in the calculation of the normalization constant N and assume a perfect segmentation. To analyze the impact of the normalization, we submitted results with (`normed`) and without (`unnormed`) normalization.

Typically, the word likelihoods are calculated using the path probability $\mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$. To investigate the impact of using the path probability as an approximation of $\mathrm{P}_{s:e}(\boldsymbol{z} \mid \boldsymbol{X})$, we submitted comparable systems for both likelihood estimates. To distinguish them, we denote the schemes as `path` or `ctc`, respectively.

The performance is measured in Average Precision (AP) and Normalized Discounted Cumulative Gain (NDCG). Additionally, both of them are applied in two ways:

(a) globally, i.e., to the whole set of results (`gAP` & `gNDCG`),
(b) individually to each query and average these individual measures (`mAP` & `mNDCG`).

Thus, in total, there are 4 different benchmark measures. A description of these performance measures is given in the appendix.

Results

This section reports the results which, again, are reported in Strauß et al. [2016]. The experiments are designed to compare the different decoding components described above:

- source likelihood (`path` and `ctc`)
- source prior probability (`abs`, `prior` and `da`)
- normalization (`normed` and `unnormed`)

We were restricted to submit only 10 systems. Since we usually use the path probability instead of the CTC probability, we skipped two decoding schemes which use the CTC posterior and are not normalized at the same time.

For additional statistics on subsets of the results or on box level results we refer to Villegas et al. [2016] were a detailed comparison of all submitted systems is available.

Normalization The Tables 7.1 and 7.2 show the impact of the normalization on the four different measures. Normalizing the probabilities typically improves the recognition score except for one configuration: If the source prior is equal to the target prior (i.e., `abs`). Then the normalization can be counterproductive if the data is different to the trained data. The NN is trained to optimize the unnormalized CTC probability. So it is not surprising that the system works well if the data fits to the training data and we only use the source posterior probability without normalization as score. Even for the development set – where the data seems to fit

Table 7.1: Results on the development set on segment level for the MDRNN trained only on the training set. `bl` denotes the baseline system.

	source prior	abs		prior		da		bl
	source posterior	path	ctc	path	ctc	path	ctc	
gAP	normed	94.81	94.89	95.36	**95.42**	94.99	95.04	74.2
	unnormed	94.77		91.73	91.87	92.58		
mAP	normed	89.71	**89.90**	89.58	89.76	89.63	89.82	49.9
	unnormed	89.42		88.59	88.89	89.13		
gNDCG	normed	96.72	96.78	96.78	**96.83**	96.73	96.77	80.1
	unnormed	96.69		96.34	96.41	96.46		
mNDCG	normed	90.77	**90.97**	90.66	90.85	90.70	90.89	51.7
	unnormed	90.61		89.96	90.25	90.36		

Table 7.2: Results on the test set on segment level for the MDRNN trained only on the training set. `bl` denotes the baseline system.

	source prior	abs		prior		da		bl
	source posterior	path	ctc	path	ctc	path	ctc	
gAP	normed	33.89	33.98	43.20	43.72	46.74	**47.13**	14.4
	unnormed	42.21		36.17	36.30	39.65		
mAP	normed	38.62	39.46	39.10	**40.03**	39.19	39.89	8.1
	unnormed	38.18		36.50	37.13	37.91		
gNDCG	normed	59.39	60.03	61.40	62.14	61.98	**62.70**	27.5
	unnormed	61.14		59.70	60.37	60.62		
mNDCG	normed	40.57	41.40	40.96	**41.86**	41.05	41.73	9.4
	unnormed	40.22		38.85	39.51	40.04		

the training data well – the gain from the normalization is not significant. Thus, the normalization can be omitted using the `abs` decoding scheme.

If source and target prior differ, the scores of different matches at distinct positions are not comparable anymore without normalization. Therefore, the normalization increases the recognition rate by around 7 `gAP` points for `prior` and `da` schemes at the test set (Table 7.2).

Path vs. CTC Probability The NN is trained to optimize the CTC probability. Thus, it is not surprising that the CTC probability is typically slightly better (less than 0.6 gAP points on the test set, see Table 7.2) than the path probability.

Finally, the gap between path and CTC probability is small for all experiments. Thus, the path probability is a good approximation for the CTC probability.

Priors The evaluation is less clear than the one above. The results do not only depend on different experimental setups (i.e., different tables) but they also highly depend on the measure. Considering the development set (Table 7.1) and `gAP`, the `prior` scheme works slightly better (less than 0.6 gAP points) than `abs` and `da`. The mAP measure puts more weight on infrequent words. Thus, the `abs` decoding scheme works better than the `prior` decoding scheme which naturally favors frequent words.

Compared to the development set, the results on the test set gain more from canceling the NN's source prior probability. Especially, if the gAP value is considered, the `da` scheme yields better results (greater than 3 gAP points) than the others. Measuring the mAP, the `prior` decoding scheme is slightly better (less than 0.2 mAP points compared to `da`).

Considering only OOV words, the `da` scheme yields the lowest error rates independent of the precise measure. This may be an indication that the `da` scheme improves the probability although there will be better estimates of the precise prior probability learned by the NN.

The Tables 7.1 and 7.2 also contain the results of the baseline system. Since baseline systems are typically not optimized, the comparison should not be overstated. Unfortunately, there was no other participant who beat the baseline.

7.3 Handwritten Text Recognition

We also applied the proposed methods to full text transcription.

7.3.1 Previous Work

HMMs: As already stated in Section 4.2, a typical HMM-HTR engine divides different levels of abstraction. Many authors distinguish the character, word and Language Model level (see e.g. Gómez [2010]). The Language Model allows in principle any combination of feasible character sequences. The set of character sequences contains not only vocables[3], it also includes punctuations, numbers etc. Since the Language Model does not consider spaces, it is often modeled as optional character appended to each "word" model. This might lead to problems as reported in Toselli and Vidal [2015]. The authors there suggest two kinds of white spaces to model e.g. a concatenation of vocables and punctuation marks more realistically.

Furthermore, it is difficult to incorporate OOV words into decoding basically because these words are not provided by the Language Model. Different authors try to tackle this problem by Language Models on subword units or even on the character level (see e.g. Kozielski et al. [2013b]).

RNNs: Graves et al. [2009] proposes the CTC-Token-Passing Algorithm (see Section 4.4.3). Unfortunately, they do not report how to integrate the space. In Graves and Jaitly [2014] the authors extend this algorithm to higher-order Language Models by a combination with Beam Search. To keep always the most promising candidates they propose a clever strategy to convert the Language Model probability into a telescoping series to add some piece of the Language Model probability at each time step instead of adding the whole prior probability at the beginning or the end of a word.

7.3.2 Segmentation

We assume a structure of the text lines which can be described by Regular Expressions: Several templates describe the atoms of natural language such as words, dates, amounts of currency, punctuations etc. This structure can be derived directly from the training set or set up manually. Character sequences which are not fitting the expected structure will not be decoded.

The Regular Expression describing all atomic parts is used to "parse" the ConfMat. Due to capturing groups, the resulting parts are automatically labeled as regions of words, numbers, dates etc. which can be used for further analysis.

At this level of abstraction, a word region is defined as a concatenation of alphabetic letters which yields many infeasible words. To further constrain the decoding, a search for the most likely vocables within the vocabulary is accomplished leading to a *Confusion Network*, i.e., a Weighted Automaton $(Q, \mathcal{V}, \lambda, q_0, F)$ with only one

[3] That means, words from a dictionary rather than arbitrary sequences from $\mathbb{A}^*$.

final state where any path from the initial to the final state visits any other state. Thus, this WA is a simple *word graphs* of *word latticies* of candidate word sequences. Although the Confusion Network relies very much on the segmentation it has been shown in several articles that is a good compromise between accuracy (of a complete word graph) and speed (Hakkani-Tür et al. [2006], Bertoldi et al. [2007]). An exemplary Confusion Network is given in Figure 7.2.

A typical CTC trained NN overestimates the NaC-likelihood. Thus, we modify the path probability for the NaC-loop. Instead of the $\lambda(q_\oslash, \oslash, q_\oslash, t) = y_{t,\oslash}$, we define $\lambda(q_\oslash, \oslash, q_\oslash, t) = C y_{t,\oslash}$ where C is a constant[4]. Now, the sum over the label sequences from $(\mathbb{A}')^T$ will not yield 1. Hence, we normalize the maximum reward by $\sum_{\boldsymbol{\pi} \in (\mathbb{A}')^T} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ where $\mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ penalizes the NaC-loop as well. This normalization can be efficiently calculated using the underlying Automaton of $T_{\mathcal{F}}$, generate the WA according to Lemma 4.22 with the additional NaC-loop penalty and calculate the total reward of Algorithm 4.

7.3.3 Integration of Language Models

Analogously to the case of keyword spotting, Language Models can be incorporated using Theorem 7.1. In contrast to keyword spotting where only single words are decoded, the decoding process has to transcribe whole lines of text. Thus, $\boldsymbol{z} := \boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ is a sequence of vocables, numbers, punctuations etc. (also spaces) covered by the Confusion Network and $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z})$ can profit from higher order Language Models. For sake of simplicity, we set $\mathrm{P}_{\mathcal{S}}(\boldsymbol{z}) \propto 1$ in this section. Then

$$\mathrm{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X}) \propto \mathrm{P}_{\mathcal{T}}(\boldsymbol{z}_1)\, \mathrm{P}_{s_1:e_1}(\boldsymbol{\pi}^*(\boldsymbol{z}_1) \mid \boldsymbol{X}) \prod_{i=2}^{n} \mathrm{P}_{\mathcal{T}}(\boldsymbol{z}_i \mid \boldsymbol{z}_1, \ldots \boldsymbol{z}_{i-1})\, \mathrm{P}_{s_i:e_i}(\boldsymbol{\pi}^*(\boldsymbol{z}_i) \mid \boldsymbol{X})$$

where n is the number of matching Capturing Groups and (s_i, e_i) are the boundaries of the ith Capturing Group. Alternatively, the analogous equation can be derived for $\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$ instead of $\mathrm{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$. Again, $\mathrm{P}_{\mathcal{T}}(\boldsymbol{z}_i \mid \boldsymbol{z}_1, \ldots \boldsymbol{z}_{i-1})$ will be approximated by an n-gram Language Model.

7.3.4 Experimental Validation

Data-Set

We demonstrate the practicability of the proposed method on a subset of the data set of Section 7.2.4 published as ICDAR2015 Competition HTRtS (Sánchez et al. [2015]). The training set contains of 433 pages for which a text-to-line alignment is available. Additionally, the organizers provided 313 pages with a transcription at

[4] In the following experiment, $C = 0.4$.

text block level without text-to-line level alignment. The testset contains 50 pages (where the transcription was unknown to the participants).

Setup

Again, we do not report the NN's architecture and the preprocessing methods and refer to Leifert et al. [2016]. The decoding results of two NNs were submitted: One NN trained only on the 433 pages with text to line alignment (Batch 1) and another NN trained on all the whole training data (Batch 1 & 2). Here we focus on three decoding strategies of different complexity to analyze the impact of each strategy:

`Dec-BP`: Best Path decoding The most simple decoding procedure is to calculate the best path $\boldsymbol{\beta} = \arg\max_{\boldsymbol{\pi} \in (\mathbb{A}')^*} \mathrm{P}(\boldsymbol{\pi} \mid \boldsymbol{X})$ without any restrictions. The resulting character sequence is $\mathcal{F}(\boldsymbol{\beta})$. This is called *Best Path Decoding* in Graves [2012].

`Dec-RE`: Regular Expression decoding The Regular Expression for this decoding scheme only models the atomic components such as amounts of money, dates, punctuation, words[5] etc. in a very flexible way. Especially, there is no vocabulary incorporated in this step. Thus, there will be many word errors.

The precise Regular Expression is composed of 5 parts, which are too technical, and is not presented here in detail. We only demonstrate exemplary simplified Capturing Groups:

- the preword character group: `[("]`
- the word group: `[A-Za-z][a-z]*`
- the word alternatives: `[0-9.,]+£?` (currency, general number)
- the post-word character group: `('s)?[)".;:]?`
- the separating character group: `[␣&-]`

Their combination forms the actual Regular Expression:

$$\boldsymbol{r} := \Big(\ \texttt{[("]?}\ \ \Big(\texttt{[0-9.,]+£?}\ \Big|\ \texttt{[A-Za-z][a-z]*}\ \Big)\ \texttt{('s)?[)".,:]?}\ \ \texttt{[␣&-]}\ \Big)\texttt{+}$$

The Algorithm 8 from Section 6.4 searches for the most likely label sequence collapsing to an element from the Regular Language defined by the above Regular Expression.

`Dec-LM`: Language Model decoding Based on the regions derived by the previous decoding step, we obtain the Confusion Network by creating a transition for each of the groups of the Regular Expression. For each transition corresponding to a

[5] Words in the sense of vocables now and further.

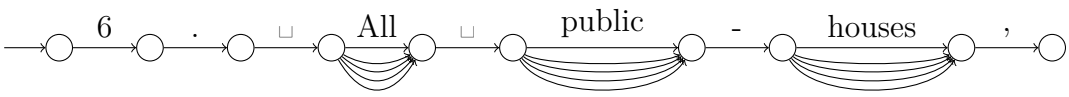


Figure 7.2: Schematic Confusion Network for the textline "*6. All public-houses,*". Multiple transitions symbolize alternative word transitions.

Table 7.3: Results of HTRtS-15 competion. *Batch 1* refers to results generated by the NN trained only on the 433 completly transcribed pages. *Batch 1 & 2* referes to results generated by the NN which was trained using any of the 746 pages.

	Batch 1		Batch 1 & 2	
	WER	CER	WER	CER
`Dec-BP`	50.35	19.82	51.16	19.96
`Dec-RE`	48.37	18.87	48.33	18.83
`Dec-LM`	33.89	15.17	33.77	15.10
A2IA			31.6	14.7
`ROVER-5`			30.2	15.5

word, we add alternative transitions between the same states for each word of the vocabulary. We also include the "raw" decoding result of the word regions fitting best to the Regular Expression. The likelihood for any word $\boldsymbol{z}$ is $\mathrm{P}_{s:e}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$ where s & e are the start and end indices of the word region. This leads to the Confusion Network where only transitions that belong to words have alternatives. Figure 7.2 shows a schematic Confusion Network used in HTR.

We simply approximate the prior probability of a word by a unigram Language Model which is derived from the word frequency of the training data. The vocabulary contains 12246 words. The prior probability of the "raw" decoding result is a constant which is considerably smaller than the smallest prior of any word from the vocabulary and which is obtained empirically.

Results

Table 7.3 shows the error rates of the HTRtS competition. Every fifth character of the best path is either wrong or missing which leads to high error rates (≈ 50 WER and ≈ 20 CER). Already the application of Regular Expressions yields some improvement although there is no stringent constraint associated with it. The inclusion of the vocabulary and of the unigram clearly yield a significant improvement.

Table 7.3 shows two additional results: We finally submitted a combination of 5 NNs

whose results are combined by ROVER (Fiscus [1997]) which additionally improved the error rates. We refer to it as `ROVER-5`. A2iA also submitted a system based on recurrent LSTM NN which incorporates a 3-gram. Their results are also included in the table.

The results show that the segmentation of the text line is reliable enough to achieve error rates which are comparable to the state-of-the-art approaches. It further shows that many confusions appear within word regions and can be corrected by vocabulary matching. Nevertheless, a higher-order Language Model will probably improve the recognition. Also a Language Model integration into the decoder could improve the results by fixing segmentation errors. The drawback of such an integration is that it typically increases the running time dramatically.

7.4 Conclusion

In this section, we investigated more practical issues. First, we showed how to integrate a Language Model into the decoding process. There are strong connections to established decoding methods such as HMMs or CTC-Token-Passing algorithm.

We introduced a decoding scheme for keyword spotting. The application of Regular Expressions makes the decoding very convenient. We also analyze the impact of the Language Model integration and conclude that it drops the error rate significantly. The calculation of the normalization constant is only a naïve approximation. More precise calculations remain open for future research.

We also propose a simple and fast method for full text recognition. This approach makes use of Regular Expressions and a (simple unigram) Language Model but also of the single-word decoding methods introduced in Section 5. We expect a higher order Language Model to further decrease the error rate which merits further investigations. Besides the speed of the proposed method, another advantage is that the feasible structure can be defined in beforehand which avoids for example missing or unintended spaces. The proposed method relies on a previous segmentation. To circumvent this problem, it is necessary to integrate the vocabulary into the Regular Expression such that segmentation into words and the word decoding is done in one step. To keep things tractable, the complex search space has to be pruned, e.g., by the method proposed in Section 6.6. First tests did not lead to an improvement. One possible reason is that we need to integrate scaling factors analogously to the Grammar Scale Factor and the Word Insertion Penalty for HMMs which we did not test yet.

8 Conclusion & Open Problems

We consider the output $\boldsymbol{Y} \in [0,1]^{T \times n}$ of a complex Neural Network trained by Connectionist Temporal Classification (CTC) for handwritten text recognition. For any given writing $\boldsymbol{X}$, the output $\boldsymbol{Y}$ yields a probability per character and per position. The search for the most likely feasible character sequence is called *decoding*. CTC is a discriminative training scheme, i.e., it maximizes the conditional likelihood $\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$ given a writing $\boldsymbol{X}$ of the corresponding character sequence $\boldsymbol{z}$. In contrast, many state-of-the-art approaches combine such Neural Networks with Hidden Markov Models (HMM) to accomplish the decoding. HMMs are generative models, that means, they internally use the probability $\mathrm{P}(\boldsymbol{X} \mid \boldsymbol{z})$ of the observation $\boldsymbol{X}$ given the character sequence $\boldsymbol{z}$. To obtain this probability, they usually use Bayes' theorem. The approach of this thesis is a discriminative decoding directly using the output of the Neural Network which works analogously to the training.

The feasible character sequences (e.g., single-words, sequences of words or Regular Languages) can be described by Finite State Automata. Different Automata or rather Finite State Transducers which model different layers of abstraction can be composed to form the entire system. Since the decoding is obviously an optimization problem, we formulate it as optimization problem on Weighted Automata. These optimization problems are solved by Dynamic Programming. Different Automata model different constraints. We review known results and reformulate them according to the Weighted Automata notation. We show how to create such Automata for certain kinds of constraints, we integrate Language Models, prune the search space by canceling branches which will not contribute to the result.

If the constraints are given by Regular Expressions the corresponding Automaton will be complex and has to save $k+1$ values in each state if the state is reached by reading k different characters. One major result of this thesis is an efficient algorithm that can handle such constraints. An investigation of the paths of the Automaton and their contribution to the final result leads to two theorems which state that we only have to save at most three values per state independent of the precise value of k which reduces the computational costs from $\mathcal{O}(n^2 T|Q|^2)$ in the worst case to $\mathcal{O}(T|Q|^2)$ if Q is the set of states of the Automaton.

We test the proposed methods in real world applications at the HTRtS data set. Especially in the case of keyword spotting, the feasible character sequences can be formulated conveniently by Regular Expressions. Additionally, we propose an approach for full text recognition although obvious issues remain for future research.

The results of both practical applications originate from two competitions where we successfully submitted a common software system of the research group including the proposed methods and algorithms.

Several aspects of this thesis remain for future research: Especially, the effective integration of Language Models into the decoder has to be investigated such that higher order Language Models could be used. This is a very promising point since previous articles reported a significant drop of the error rates after incorporating Language Models. Recently, Neural Language Models have been reported to outperform n-grams. Thus, a comparison of both approaches applied to HTR could be very interesting.

A more reliable normalization constant in case of keyword spotting which is still tractable will surely improve the recognition results.

The analysis of the conceptional difference between discriminative and generative decoding remains as an open problem. We already showed that the integration of a Language Model leads to a similar decoding process assumed that the HMM states have the same interpretation as the CTC states. Is there a discriminative equivalent for a greater number of HMM states and what is its interpretation? Could this modified topology improve the recognition capacity of a Neural Network?

Appendix

1 Evaluation Measures

In the following, we introduce the Average Precision (AP) and the Normalized Discounted Cumulative Gain (NDCG) as it was used in Villegas et al. [2016].

Assume the result list is sorted according to the score. Let $\mathrm{TP}(i)$ (true positive) be 1 if match i is correct and 0 else. Then the *Precision* of the first k elements of the result list is defined as

$$\mathrm{prec}(k) := \frac{\sum_{i=1}^{k} \mathrm{TP}(i)}{k}.$$

Let $N > 0$ be the number of matches and $M > 0$ be the number of keywords in the ground truth (targets), then

$$\mathrm{AP} := \frac{1}{M} \sum_{k=1}^{N} \mathrm{prec}(k)\, \mathrm{TP}(k).$$

The competition organizers referred to the AP over the whole query list as gAP and to the averaged AP over the individual queries as mAP. Since the latter case also comprises situations where M or N are 0, they defined $\mathrm{AP} = 1$ if $M = N = 0$ and $\mathrm{AP} = 0$ if $M = 0 \neq N$ or $N = 0 \neq M$.

The mNDCG and gNDCG are defined analogously where NDCG is

$$\mathrm{NDCG} := \begin{cases} \frac{1}{Z} \sum_{k=1}^{N} \frac{2^{\mathrm{TP}(k)}-1}{\log_2(k+1)} & \text{if } N > 0 \cap M > 0 \\ 1 & \text{if } N = M = 0 \\ 0 & \text{else} \end{cases}$$

where $Z := \sum_{k=1}^{N} \frac{1}{\log_2(k+1)}$ is a normalization constant to keep the values between 0 an 1.

Obviously, the individual measures (mAP & mNDCG) put equal weight on each keyword query which favors infrequent queries compared to the global ones (gAP & gNDCG). Figure 1 shows the relation between NDCG and AP for a single query. NDCG favors a few results with high rank and there is a big plateau with no significant decrease of impact. For AP, there a significant decrease of impact in k until

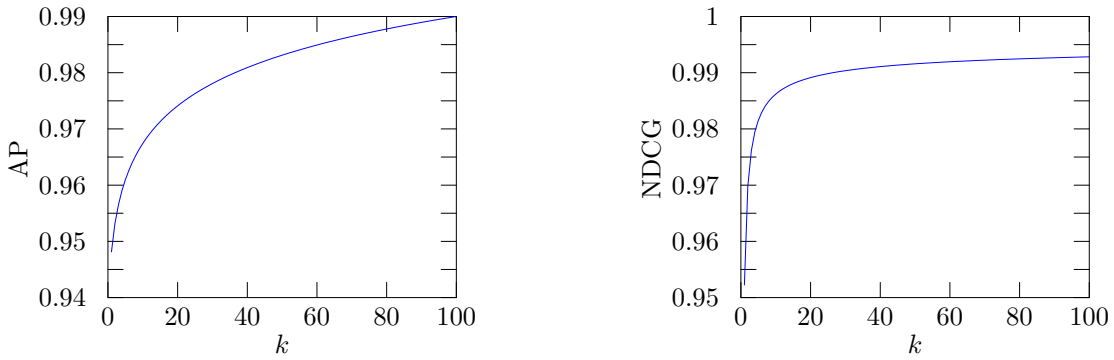


Figure 1: AP and NDCG, respectively for $N = M = 100$ and only one false positive at position k

the end of the result list.

Acknowledgement

First of all, I would like to thank my supervisor Prof. Dr. Roger Labahn. Doubtless this thesis would not have been written without your commitment and valuable support.

A special thanks to my colleagues Gundram Leifert and Tobias Grüning for fruitful discussions. It is a pleasure to work with you guys. Thanks also to Max Weidemann for canceling severeal typos. I am grateful to the people from Planet, especially Welf Wustlich, Jesper Kleinjohann, Ulrich Bernert, Jan Starzynski and Klaus Geltmeier, for motivating the research by practical needs. Thanks to Udo Siewert for proof-reading and very detailed suggestions in the last years. Thanks to anyone working at the Institute of Mathematics for friendliness and collegiality.

Last but not least I have to mention the two most important people in my life: Anne and Levi. Thanks for all your love.

List of Figures

List of Tables

List of Algorithms

Acronyms

CER character error rate . . . 25
ConfMat Confidence Matrix . . . 2
DAFSA Deterministic Acyclic (Finite State) Automaton . . . 12
DFA Deterministic Finite Automaton . . . 8
FST Finite State Transducer . . . 17
HMM Hidden Markov Model . . . 29
HTR handwritten text recognition . . . 1
LeakyLP LeakyLP Cell . . . 41
LM Language Model . . . 27
LSTM Long Short Term Memory Cell . . . 39
MLP Multi-layer Perceptron . . . 36
NaC not-a-character or blank . . . 47
NFA Nondeterministic Finite Automaton . . . 8
NFPA Probabilistic Automaton without final probabilities and only one initial state . . . 16
NN (Artificial) Neural Network . . . 36
OOV Out-of-vocabulary . . . 98
PA Probabilistic Automaton . . . 15
RNN Recurrent Neural Network . . . 38
WA Weighted Automaton . . . 14
WER word error rate . . . 26

Symbols

$\oslash$	not-a-character or blank (NaC) 47
$\mathring{\mathrm{A}}_{\boldsymbol{z}}$	Automaton accepting $\mathcal{F}^{-1}(\boldsymbol{z})$ 49
$\mathbb{A}'$	Extended alphabet 47
$\boldsymbol{\beta}$	best path 49
ε	empty word 7
$\mathcal{F}$	Collapse function 49
$\mathcal{L}$	Regular Language 7
$\mathrm{P}(\boldsymbol{\pi}^*(\boldsymbol{z}) \mid \boldsymbol{X})$	path probability 53
$\mathrm{P}(\boldsymbol{z} \mid \boldsymbol{X})$	CTC (word) probability 53
$\boldsymbol{o}_{1:n}$	Compact notation of a sequence $o_1, \ldots, o_n$ 5
$\mathcal{S}^*$	Kleene closure on $\mathcal{S}$ 5
T	ConfMat/output positions 2
$T_{\mathcal{F}}$	FST which defines the relation $\{(\boldsymbol{\pi}, \mathcal{F}(\boldsymbol{\pi})) \mid \boldsymbol{\pi} \in (\mathbb{A}')^*\}$ 49
$\mathcal{V}$	vocabulary of feasibel words 27
$W_{\boldsymbol{X}}(A)$	Weighted Automaton with support A and weights resulting from Network's output of input $\boldsymbol{X}$ 48
$\boldsymbol{Y}$	ConfMat (Output of a NN with Softmax activation given some input $\boldsymbol{X}$) 4
$\boldsymbol{z}'$	Extension of word $\boldsymbol{z}$ pumped up by NaCs. 49
$\propto$	Proportional 5

Bibliography

L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

R. E. Bellman and S. E. Dreyfus. *Applied dynamic programming*. Princeton University Press, 1962.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, 3(Feb):1137–1155, 2003.

N. Bertoldi, R. Zens, and M. Federico. Speech translation by confusion network decoding. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–1297. IEEE, 2007.

G. Bideault, L. Mioulet, C. Chatelain, and T. Paquet. Benchmarking discriminative approaches for word spotting in handwritten documents. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 201–205. IEEE, 2015a.

G. Bideault, L. Mioulet, C. Chatelain, and T. Paquet. Spotting Handwritten Words and REGEX using a two stage BLSTM-HMM architecture. In *SPIE/IS&T Electronic Imaging*, pages 94020G–94020G. International Society for Optics and Photonics, 2015b.

T. Bluche. *Deep Neural Networks for Large Vocabulary Handwritten Text Recognition*. PhD thesis, Université Paris Sud-Paris XI, 2015.

T. Bluche, H. Ney, and C. Kermorvant. Feature extraction with convolutional neural networks for handwritten word recognition. In *2013 12th International Conference on Document Analysis and Recognition*, pages 285–289. IEEE, 2013.

T. Bluche, J. Louradour, M. Knibbe, B. Moysset, M. F. Benzeghiba, and C. Kermorvant. The A2iA Arabic Handwritten Text Recognition System at the Open HaRT2013 Evaluation. In *11th IAPR International Workshop on Document Analysis Systems (DAS), 2014*, pages 161–165. IEEE, 2014.

T. Bluche, H. Ney, J. Louradour, and C. Kermorvant. Framewise and CTC Training of Neural Networks for Handwriting Recognition. In *International Conference on Document Analysis and Recognition – ICDAR 2015*, pages 81–85, 2015.

J. L. Castro, C. J. Mantas, and J. Benıtez. Neural networks with a continuous squashing function in the output are universal approximators. *Neural Networks*, 13(6):561–563, 2000.

S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 310–318. Association for Computational Linguistics, 1996.

K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

J. Daciuk, S. Mihov, B. W. Watson, and R. E. Watson. Incremental construction of minimal acyclic finite-state automata. *Computational linguistics*, 26(1):3–16, 2000.

W. De Mulder, S. Bethard, and M.-F. Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Computer Speech & Language*, 30(1):61–98, 2015.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (Methodological)*, pages 1–38, 1977.

P. Doetsch, M. Kozielski, and H. Ney. Fast and robust training of recurrent neural networks for offline handwriting recognition. In *14th International Conference on Frontiers in Handwriting Recognition (ICFHR), 2014*, pages 279–284. IEEE, 2014.

P. Dupont, F. Denis, and Y. Esposito. Links between probabilistic automata and hidden Markov models: probability distributions, learning models and induction algorithms. *Pattern Recognition*, 38(9):1349–1371, 2005.

S. Fernández, A. Graves, and J. Schmidhuber. An application of recurrent neural networks to discriminative keyword spotting. In *International Conference on Artificial Neural Networks*, pages 220–229. Springer, 2007.

A. Fischer, A. Keller, V. Frinken, and H. Bunke. HMM-based word spotting in handwritten documents using subword models. In *20th International Conference on Pattern Recognition (ICPR), 2010*, pages 3416–3419. IEEE, 2010.

J. G. Fiscus. A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER). In *1997 IEEE Workshop on Automatic Speech Recognition and Understanding, 1997. Proceedings.*, pages 347–354. IEEE, 1997.

G. D. Forney Jr. The Viterbi Algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

V. Frinken, A. Fischer, and H. Bunke. A Novel Word Spotting Algorithm Using Bidirectional Long Short-Term Memory Neural Networks. In F. Schwenker and N. Gayar, editors, *Artificial Neural Networks in Pattern Recognition*, volume 5998 of *Lecture Notes in Computer Science*, pages 185–196. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12158-6. doi: 10.1007/978-3-642-12159-3_17. URL `http://dx.doi.org/10.1007/978-3-642-12159-3_17`.

V. Frinken, A. Fischer, R. Manmatha, and H. Bunke. A novel word spotting method based on recurrent neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(2):211–224, 2012.

V. R. Gómez. *Multimodal Interactive Transcription of Handwritten Text Images*. PhD thesis, Universitat Politècnica de València, 2010.

A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385 of *Studies in Computational Intelligence*. Springer, 2012. ISBN 978-3-642-24796-5.

A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014.

A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009.

A. Graves, S. Fernández, F. J. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine learning*, pages 369–376, 2006.

A. Graves, S. Fernández, and J. Schmidhuber. Multi-dimensional recurrent neural networks. In *Proceedings of the 2007 International Conference on Artificial Neural Networks*, Porto, Portugal, 2007.

A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernández. Unconstrained on-line handwriting recognition with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 577–584, 2008.

A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

E. Grosicki and H. E. Abed. ICDAR 2009 Handwriting Recognition Competition. In *2009 10th International Conference on Document Analysis and Recognition*, pages 1398–1402. IEEE Computer Society, 2009.

D. Hakkani-Tür, F. Béchet, G. Riccardi, and G. Tur. Beyond ASR 1-best: Using word confusion networks in spoken language understanding. *Computer Speech & Language*, 20(4):495–514, 2006.

D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural computation*, 9(8):1735–1780, 1997.

J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Pearson Education India, second edition edition, 2001.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

H. Jaeger. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach. Technical Report GMD Report 159, German National Research Center for Information Technology, 2002.

J. Jiang. *Domain adaptation in natural language processing*. Phd thesis, University of Illinois at Urbana-Champaign, 2008. URL `https://www.ideals.illinois.edu/bitstream/handle/2142/11465/Domain%20Adaptation%20in%20Natural%20Language%20Processing.pdf?sequence=2&isAllowed=y`.

T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

Y. Kessentini, C. Chatelain, and T. Paquet. Word Spotting and Regular Expression Detection in Handwritten Documents. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 516–520. IEEE, 2013.

W. Khreich, E. Granger, A. Miri, and R. Sabourin. A survey of techniques for incremental learning of HMM parameters. *Information Sciences*, 197:105–130, 2012.

S. C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In C. Shannon and J. McCarthy, editors, *Automata Studies, Annals of Math. Studies 34*, page 3–42. Princeton University Press, New Jersey, 1956.

M. Kozielski, P. Doetsch, and H. Ney. Improvements in RWTH's system for off-line handwriting recognition. In *2013 12th International Conference on Document Analysis and Recognition*, pages 935–939. IEEE, 2013a.

M. Kozielski, D. Rybach, S. Hahn, R. Schlüter, and H. Ney. Open vocabulary handwriting recognition using combined word-level and character-level language models. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8257–8261. IEEE, 2013b.

A. Krogh et al. An introduction to hidden Markov models for biological sequences. *New Comprehensive Biochemistry*, 32:45–63, 1998.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

G. Leifert, T. Strauß, R. Labahn, and W. Wustlich. A new Approach for Cells in Multidimensional Recurrent Neural Networks. Technical report, University of Rostock, 2013.

G. Leifert, T. Grüning, T. Strauß, and R. Labahn. CITlab ARGUS for historical data tables: Description of CITlab's system for the ANWRESH-2014 Word Recognition task. *arXiv preprint arXiv:1412.6012*, Apr. 2014a.

G. Leifert, T. Strauß, T. Grüning, and R. Labahn. Cells in Multidimensional Recurrent Neural Networks. *arXiv preprint arXiv:1412.2620*, 2014b. submitted to Journal of Machine Learning Research.

G. Leifert, T. Strauß, T. Grüning, and R. Labahn. CITlab ARGUS for historical handwritten documents: Description of CITlab's system for the HTRtS 2015 task : Handwritten text recognition on the transcriptorium dataset. *arXiv preprint arXiv:1605.08412*, 2016.

C. D. Manning and H. Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 188–191. IEEE, 1971.

S. Mihov. Direct building of minimal automaton for given list. *Annuaire de l'Université de Sofia "St. Kl. Ohridski"*, 91(1):212–225, 1998.

M. Minsky and S. Papert. *Perceptrons*. MIT press, 1969.

A. Mnih and K. Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in Neural Information Processing Systems*, pages 2265–2273, 2013.

A. Mnih and Y. W. Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

M. Mohri. Weighted automata algorithms. In *Handbook of weighted automata*, pages 213–254. Springer, 2009.

M. Mohri, F. Pereira, and M. Riley. Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pages 559–584. Springer, 2008.

M. Moyle, J. Tonra, and V. Wallace. Manuscript transcription by crowdsourcing: Transcribe Bentham. *Liber Quarterly*, 20(3-4), 2011.

B. Moysset, T. Bluche, M. Knibbe, M. F. Benzeghiba, R. Messina, J. Louradour, and C. Kermorvant. The A2iA multi-lingual text recognition system at the second Maurdor evaluation. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 297–302. IEEE, 2014.

M. Pechwitz, S. S. Maddouri, V. Märgner, N. Ellouze, and H. Amiri. IFN/ENIT-database of handwritten arabic words. In *7th Colloque International Francophone sur l'Ecrit et le Document (CIFED 2002)*, Hammamet, Tunis, 2002.

T. Plötz and G. A. Fink. Markov models for offline handwriting recognition: a survey. *International Journal on Document Analysis and Recognition (IJDAR)*, 12(4):269–298, 2009.

D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, et al. The Kaldi speech recognition toolkit. In *IEEE 2011 workshop on automatic speech recognition and understanding*, number EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

I. Pratikakis, K. Zagoris, B. Gatos, G. Louloudis, and N. Stamatopoulos. ICFHR 2014 competition on handwritten keyword spotting (H-KWS 2014). In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 814–819. IEEE, 2014.

J. Puigcerver, A. H. Toselli, and E. Vidal. ICDAR2015 Competition on Keyword Spotting for Handwritten Documents. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 1176–1180. IEEE, 2015a.

J. Puigcerver, A. H. Toselli, and E. Vidal. Probabilistic interpretation and improvements to the HMM-filler for handwritten keyword spotting. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 731–735. IEEE, 2015b.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

T. M. Rath and R. Manmatha. Word spotting for historical documents. *International Journal of Document Analysis and Recognition (IJDAR)*, 9(2-4):139–152, 2007.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

S. Ruder. An overview of gradient descent optimization algorithms. *ArXiv e-prints*, Sept. 2016.

S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

D. Rybach, C. Gollan, G. Heigold, B. Hoffmeister, J. Lööf, R. Schlüter, and H. Ney. The RWTH aachen university open source speech recognition system. In *Interspeech*, pages 2111–2114, 2009.

H. Sak, A. Senior, K. Rao, O. Irsoy, A. Graves, F. Beaufays, and J. Schalkwyk. Learning acoustic frame labeling for speech recognition with recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 4280–4284. IEEE, 2015.

J. A. Sánchez, V. Romero, A. H. Toselli, and E. Vidal. ICFHR2014 Competition on Handwritten Text Recognition on tranScriptorium Datasets (HTRtS). In *Proceedings of the International Conference on Frontiers in Handwriting Recognition – ICFHR 2014*, Aug. 2014.

J. A. Sánchez, A. H. Toselli, V. Romero, and E. Vidal. ICDAR2015 Competition HTRtS: Handwritten Text Recognition on the tranScriptorium Dataset. In *Proceedings of the International Conference on Document Analysis and Recognition – ICDAR 2015*, 2015.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

M. Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

A. Stevenson, editor. *Oxford Dictionary of English*. Oxford University Press, 2010.

T. Strauß, R. Labahn, G. Leifert, and W. Wustlich. An approach for output decoding of neural networks. Technical Report 2012/3, Universität Rostock, Dec. 2012. Available: `http://ftp.math.uni-rostock.de/pub/preprint/2012/pre12_03.pdf`.

T. Strauß, R. Labahn, G. Leifert, and W. Wustlich. An extended approach for output decoding of neural networks. Technical Report 2013/2, Universität Rostock, Nov. 2013.

T. Strauß, T. Grüning, G. Leifert, and R. Labahn. CITlab ARGUS for historical handwritten documents: Description of CITlab's system for the HTRtS 2014 Handwritten Text Recognition task. *arXiv preprint arXiv:1412.3949*, Apr. 2014.

T. Strauß, T. Grüning, G. Leifert, and R. Labahn. CITlab ARGUS for Keyword Search in Historical Handwritten Documents: Description of CITlab's System for the ImageCLEF 2016 Handwritten Scanned Document Retrieval Task. In *CLEF2016 Working Notes*, CEUR Workshop Proceedings, Évora, Portugal, September 5-8 2016. CEUR-WS.org <http://ceur-ws.org>.

T. Strauß, G. Leifert, T. Grüning, and R. Labahn. Regular expressions for decoding of neural network outputs. *Neural Networks*, pages 1–11, 2016. doi: 10.1016/j.neunet.2016.03.003.

K. Thompson. Programming Techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

A. H. Toselli and E. Vidal. Handwritten Text Recognition Results on the Bentham Collection with Improved Classical N-Gram-HMM methods. In *Proceedings of the 3rd International Workshop on Historical Document Imaging and Processing*, pages 15–22. ACM, 2015.

E. Vidal, F. Thollard, C. De La Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic Finite-State Machines-Part II. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1026–1039, 2005.

E. Vidal, A. H. Toselli, and J. Puigcerver. High performance query-by-example keyword spotting using query-by-string techniques. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 741–745. IEEE, 2015.

M. Villegas, J. Puigcerver, A. H. Toselli, J.-A. Sánchez, and E. Vidal. Overview of the ImageCLEF 2016 Handwritten Scanned Document Retrieval Task. In *CLEF2016 Working Notes*, CEUR Workshop Proceedings, Évora, Portugal, September 5-8 2016. CEUR-WS.org <http://ceur-ws.org>.

L. Vincent. Google Book Search: Document Understanding on a Massive Scale. In *ICDAR*, pages 819–823, 2007.

P. Vincent, A. de Brébisson, and X. Bouthillier. Efficient exact gradient update for training deep networks with very large sparse targets. In *Advances in Neural Information Processing Systems*, pages 1108–1116, 2015.

L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

M. Zimmermann and H. Bunke. Optimizing the integration of a statistical language model in HMM based offline handwritten text recognition. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 2, pages 541–544. IEEE, 2004.