A Large Vocabulary Online Handwriting Recognition System for Turkish

by

Esma Fatıma Bilgin Taşdemir

Submitted to the Graduate School of Engineering and
Natural Sciences
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Sabancı University
June, 2018

A LARGE VOCABULARY ONLINE HANDWRITING RECOGNITION SYSTEM FOR TURKISH

APPROVED BY:

Prof. Dr. Ayşe Berrin Yanıkoğlu.....

(Thesis Supervisor)

Asst. Prof. Dr. Kamer Kaya.....

Asst. Prof. Dr. Hüseyin Özkan.....

Assoc. Prof. Dr. Mehmet Göktürk.....

(Gebze Teknik Üniversitesi- Bilgisayar Mühendisliği Bölümü)

Prof. Dr. Tunga Güngör.....

(Boğaziçi Üniversitesi - Bilgisayar Mühendisliği Bölümü)

DATE OF APPROVAL: 01/06/2018

Acknowledgments

Foremost, I would like to thank my advisor Prof. Berrin Yanıkoğlu for the continuous support during my research and for providing me with the opportunity to complete my PhD thesis at Sabanci University.

I would like to express my sincere gratitude to my friends Atia Shafique and Damla Arifoğlu for giving me support, good advice and making my gradute years more joyful. I owe special thanks to my family, to my dad who always encourages his children in their academic journeys, to my mom who provides me the invaluable support whenever I need, to my sisters Rümeysa, Zehra and Betül whose substantive, enthusiastic conversations on both academic topics and everything else are always a source of inspiration and to my brother Ahmed who helps me to take a break by changing the subject to arts and crafts.

I wish to thank my husband, Benna, who has been a constant source of support and encouragement during the challenges of graduate school. Without his enduring help, this thesis would not have been possible. Finally, I thank to my son İbrahim, an endless source of love, for patiently waiting for his mommy to finish reading and writing so that he could spend happy moments with her.

This thesis was financially supported by Scientific and Research Council of. Turkey (TÜBİTAK) with the 2211-Graduate Scholarship Programme (2211-Yurt İçi Doktora Burs Programı). Also part of this work is supported by TÜBİTAK under the project number 113E062. I would like to express my sincere gratitude for the supports provided.

A Large Vocabulary Online Handwriting Recognition System for Turkish

Esma Fatıma Bilgin Taşdemir
Computer Science and Engineering
Ph.D. Thesis, 2018
Thesis Supervisor: A. Berrin Yanıkoğlu

Keywords: Hidden Markov Models, Online Handwriting Recognition, Delayed Strokes, Turkish Handwriting Recognition

Abstract

Handwriting recognition in general and online handwriting recognition in particular has been an active research area for several decades. Most of the research have been focused on English and recently on other scripts like Arabic and Chinese. There is a lack of research on recognition in Turkish text and this work primarily fills that gap with a state-of-the-art recognizer for the first time. It contains design and implementation details of a complete recognition system for recognition of Turkish isolated words. Based on the Hidden Markov Models, the system comprises pre-processing, feature extraction, optical modeling and language modeling modules. It considers the recognition of unconstrained handwriting with a limited vocabulary size first and then evolves to a large vocabulary system.

Turkish script has many similarities with other Latin scripts, like English, which makes it possible to adapt strategies that work for them. However, there are some other issues which are particular to Turkish that should be taken into consideration separately. Two of the challenging issues in recognition of Turkish text are determined as delayed strokes which introduce an extra source of variation in the sequence order of the handwritten input and high Out-of-Vocabulary (OOV) rate of Turkish when words are used as vocabulary units in the decoding process. This work examines the problems and alternative solutions at depth and proposes suitable solutions for Turkish script particularly.

In delayed stroke handling, first a clear definition of the delayed strokes is developed and then using that definition some alternative handling methods are evaluated extensively on the UNIPEN and Turkish datasets. The best results are obtained by removing all delayed strokes, with up to 2.13% and 2.03% points recognition accuracy increases, over the respective baselines of English and Turkish. The overall system performances are assessed as 86.1% with a 1,000-word lexicon and 83.0% with a 3,500-word lexicon on the UNIPEN dataset and 91.7% on the Turkish dataset.

Alternative decoding vocabularies are designed with grammatical sub-lexical units in order to solve the problem of high OOV rate. Additionally, statistical bi-gram and tri-gram language models are applied during the decoding process. The best performance, 67.9% is obtained by the large stem-ending vocabulary that is expanded with a bi-gram model on the Turkish dataset. This result is superior to the accuracy of the word-based vocabulary (63.8%) with the same coverage of 95% on the BOUN Web Corpus.

Türkçe İçin Geniş Dağarcıklı Çevrimiçi El Yazısı Tanıma Sistemi

Esma Fatıma Bilgin Taşdemir
Bilgisayar Bilimi ve Mühendisliği
Doktora Tezi, 2018
Tez Danışmanı: A. Berrin Yanıkoğlu

Anahtar Sözcükler: Saklı Markov Modelleri, Çevrimiçi El Yazısı Tanıma, Gecikmiş Vuruş, Türkçe El Yazısı Tanıma

Özet

El yazısı tanıma alanında yapılan pek çok çalışma İngilizce, Arapça ve Çince gibi dillerin yazılarını konu almaktadır. Türkçe için yapılmış sınırlı çalışmaların arasında çevrimiçi tanıma konusunda eksiklik vardır. Bu tez çalışmasıyla ilk kez olarak, en gelişmiş teknolojiyi içeren bir yalıtık ve kısıtsız şekilde yazılmış Türkçe kelime tanıma sistemi gerçekleştirilmiştir. Saklı Markov Modelleri kullanılan sistem önişleme, öznitelik çıkarma, optik modelleme ve dil modelleme birimlerinden oluşmaktadır. Sistem, orta ölçekli bir dağarcıkla tasarlanıp daha sonra büyük dağarcıkla çalışır hale getirilmiştir.

Türkçe yazının Latin alfabesi kullanan diğer yazı sistemleri ile olan benzerlikleri, literatürde kullanılan pek çok tekniği Türkçe için de kullanılabilir kılar. Ancak Türkçe'ye has bazı özellikler tanıma işlemini güçleştirmektedir. Bunlardan ikisi gecikmiş vuruşlar ve çok fazla sayıda olan dağarcık dışı kelimelerdir. Bu tezde her iki problem de ayrıntılı şekilde ele alınmış ve bazı çözümler üretilmiştir.

Gecikmiş vuruşlar için net bir tanım oluşturulmuş ve bu tanım kullanılarak bir dizi önişleme yöntemi arasından Türkçe'ye en uygunu bulunmuştur. İngilizce UNIPEN veri kümesi ve Türkçe verilerden oluşan diğer bir küme üzerinde yapılan testlerde en iyi sonuç, bu vuruşların silinmesi yöntemi ile elde edilmiştir. Bu şekilde yapılan önişleme ile İngilizce'de 1,000 kelimelik dağarcık için %2.23 artışla %86.1 tanıma başarısı gözlenirken Türkçe'de %2.03 artışla %91.7 tanıma oranı yakalanmıştır.

Tanıma sisteminin çözümleme aşamasında kelime-altı birimler kullanılarak dağarcık dışı kelimelerin tanıma başarısına olan olumsuz etkisinin giderilmesi sağlanmıştır. Ayrıca, N-gram istatistiksel dil modelleri de kullanılmıştır. Geniş dağarcıklı tanıma için gövde-ekler şeklinde kelime-altı birimlerin kullanılması ile elde edilen %67.9 tanıma başarısı, kelimelerin kullanılması ile elde edilen başarıdan (%63.8) daha fazla olarak ölçülmüştür.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction to Online Handwriting Recognition

Online handwriting recognition (OHWR) is the task of interpreting handwritten input, at character, word, or line level. The handwriting is captured by a digitizer equipment, such as a tablet or pen-enabled smart phones, and represented in the form of a time series of pen tip coordinates that are captured. Other types of data such as pen pressure, pen up/down status, velocity, azimuth and altitude are collected by many modern input devices as well.

The main component in a OHWR system is a recognizer which receives the input signal, runs an algorithm according to the technique it is designed with and outputs digital text as the result with a probability. A language model can be employed in order to improve the results of the recognition module. Typically, a lexicon containing the vocabulary for a given task is used to restrict the word hypothesis to be one of the allowed alternatives.

The size of the recognition vocabulary has a direct effect on recognition performance. Recognition speed and performance drops as the number of alternative words get larger. On the other hand, a recognizer can only recognize words contained in the vocabulary, so, not included words translates into errors. Vocabularies of size over 5,000 are usually classified as 'large' while a typical modern large vocabulary recognition task has a lexicon of 20,000 or more words.

By the advancement of technology, OHWR is found more and more in everyday life. However, despite on-going research for several decades and significant improvements brought on with the use of deep learning techniques recently, recognition systems are far from being perfect in case of unconstrained handwriting recognition with a large vocabulary for general purpose. For this reason, systems sometimes constrain the writing style to be hand-print only (e.g. as in forms) in order to simplify the task of the recognizer. In addition to writing style limitations, some applications may require input written in a limited area like a box or over a baseline.

1.1 Issues in Online Handwriting Recognition

Handwriting recognition requires accurate and robust models to accommodate the variability in time and feature space. The main sources of variations within handwriting styles are natural inter-writer (how different people write different characters and words) and intra-writer variances (how one's handwriting vary from time to time), as well as technical specifications (e.g. sampling rate) of digitizers that record the input signal.

Natural Variations

Handwriting is a complex activity that requires fine motor skills. Each individual develops a style of her own and even then his/her writings show variations from one sample to the other. The shapes of the characters, slant and skew of the writing, conformity to a base line, writing speed and pen pressure and order of strokes are all sources of variations affecting the style.

Writer-dependent systems are trained and tested with writings of the same writer(s). If a system is designed to be writer-dependent, its parameters are tuned to recognize hand-writings from a single writer or a few writers hence, the amount of variations drops dramatically. A writer-independent system is capable of recognizing handwriting from users whose writing are not seen by the system during training. Much larger training data, effective normalization before recognition and more complex recognizer architecture are needed for learning invariant and generalized characteristics of handwriting in writer-independent systems.

Writing styles can be grouped as hand-printed, cursive and a mixture of these two. Recognition of fully cursive style is the most difficult because of missing pen-up points which marks the boundaries of characters in hand-printed style.

In addition to the variations general to all writing systems, some scripts may be more challenging with number of characters, similarity between characters, allographs, ligatures and writing order of the strokes in a character. A simple example of the latter one is differences in writing order of dots of 'i' letters and horizontal bars of 't' letters in Latin-based alphabets, which is referred as 'delayed strokes'. Turkish script has more of such characters in addition to 'i' and 't' which introduces more variation in writing order.

Vocabulary Size

Recognition vocabulary is another factor to determine performance of an OWHR. Most of the time, words that can be recognized are limited with a lexicon of arbitrary size. When the system assumes that the input will be one of the closed set of words (e.g. a list of words that contain all possible numbers on a bank check), it is called a closed-vocabulary system. An open-vocabulary system on the other hand is capable of recognizing words outside of the lexicon (e.g. character-based systems are example for open-vocabulary systems). Vocabulary size has direct impact on the system design. A closed-vocabulary task with a small vocabulary size can model these words directly in recognition phase. However, as

the vocabulary size increases, modeling of letters instead of words is preferred because of the computational complexity of modeling individual words.

Although open-vocabulary tasks are deemed to be harder than small-to-medium sized closed-vocabulary tasks, closed-vocabulary approach can be equally challenging with large vocabulary sizes. In this regard, Turkish language introduces additional difficulty due to its level of productivity in word formation.

Turkish is an agglutinative language where new words are formed by adding suffixes to the end of root words. There are grammatical rules governing which suffixes may follow which others, and in what order, but the number of possible words that may be generated by adding suffixes is practically infinite. As such, a finite-size vocabulary for Turkish would miss a significant percentage of Turkish words, causing a high Out-of-Vocabulary (OOV) rate. This makes vocabulary-based text recognition approaches unsuitable for Turkish, or other agglutinative languages. As a solution to OOV problem, sub-word-based vocabularies have been suggested recently.

Imperfections in Input Signal

Imperfections of a digitizer in capturing strokes of the writer may lead to missing data points which in turn causes to gaps within strokes. On the other hand, the digitizer can record extra points which makes the signal noisy. Both the missing data and noisy data problems should be handled before proceeding further in the recognition process. Inexperienced writers may have difficulties in using the stylus and interface of the application. Some writers may even fail to properly record their writings once they create them. Mislabeled data due to user fault constitute a serious problem for handwriting datasets.

Evaluation Metrics

There are several evaluation metrics for OWHR. Word accuracy is the percentage of words recognized correctly as outputting the same label with the reference string. Word Error Rate (WER) is a popular metric which is initially used in speech recognition domain. WER provides a solution where comparison of sequences with different lengths is possible. It is based on the Levenshtein distance [1]. It is calculated with a dynamic programming algorithm efficiently. WER is calculated as the minimum number of edits with substitution, insertion and deletion of words from the reference string to the output, normalized by the number of reference words:

$$WER = \frac{\#Substitutions + \#Insertions + \#Deletions}{\#References} \tag{1.1}$$

Similar to WER, a Character Error Rate (CER) is calculated by using characters instead of words for counting number of edits.

1.2 Online Handwriting Recognition Systems Overview

Handwriting recognition systems are typically designed as a large pipeline process where the output of a previous module is input to the next one.

In the pre-processing step, the raw data from the digitizer is cleaned from noisy and spurious data and missing data points are recovered. If the handwriting is in multiple lines, line segmentation is applied beforehand. Next, several normalization procedures are applied to reduce variations in writing order, size, slant, skew and writing speed. Finally, relevant and discriminant features are extracted before the recognition phase. Most of the time, features are heuristic and handcrafted, designed for capturing the important characteristics of handwriting based on human knowledge. With the recent *Deep Learning* technique, features can be learned from the training data itself, but it is a usual practice to use handcrafted features along with self-learned features in online handwriting domain [2].

Once the handwriting is represented as feature vectors, it can be used for training a recognizer. Data used for training (i.e the train set) is learned to tune parameters of the system according to the technique the recognizer is based on. Next, the recognizer is tested on hold-out data to evaluate its performance on unseen data. A language model can be employed in this phase to integrate language knowledge to obtain better recognition results. Sometimes a final step of post-processing can be used for further improvement of the output. The whole system can be seen in Figure 1.1

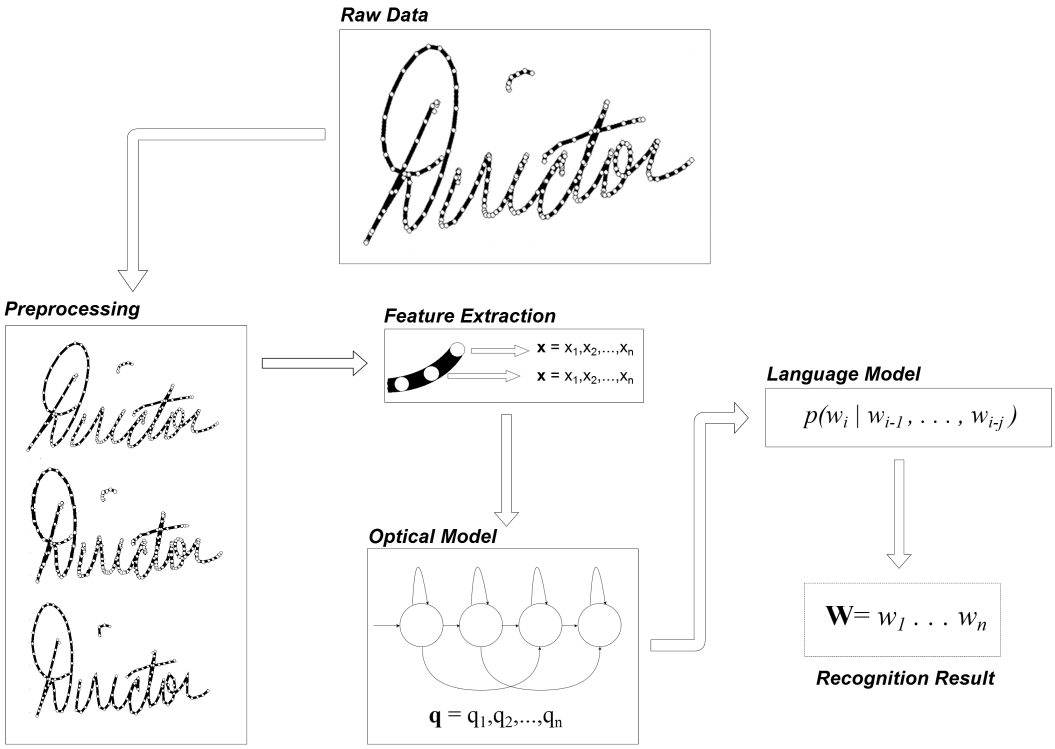

Figure 1.1: The recognition process as a pipeline

1.3 Literature Review

There have been many studies since early 1990s on online handwriting recognition problem (see [3] for a survey). While much of this research is focused on recognition of Latin-based alphabets, especially English, handwriting recognition in other scripts has also been gaining attention in recent years [4, 5, 6]. Initially limited to the recognition of isolated characters and digits, state-of-the-art research now aims at recognizing unconstrained word and sentence recognition.

Different techniques and approaches are used in recognition systems. One of the oldest and still popular techniques in handwriting recognition are Hidden Markov Models (HMM) [7, 8, 9, 10] and Artificial Neural Networks (ANN or NN) [11, 12, 13, 14]. More recently, deep learning techniques using recurrent neural networks have also been applied with very good results [15, 16, 17]. As each approach has its own strength, hybrid systems combining deep learning or other neural network approaches with hidden Markov models are proposed to combine the benefits of both approaches.

In the rest of this section, a comprehensive online handwriting recognition literature survey covering articles published after year 2000 is presented. Recognition systems are reviewed in subsections based on the recognition technique. Studies on Turkish handwriting recognition are covered in a separate subsection.

1.3.1 Hidden Markov Models Based Systems

Hidden Markov Models (HMM) form the earliest and the most widely used approaches in online handwriting recognition problem. HMMs can be used for modeling strokes, characters or words and can be used by themselves or in conjunction with other methods in hybrid systems [16, 18].

In many systems the parameters of an HMM are typically optimized by the Maximum Likelihood (ML) approach. However there are some shortcomings of this approach, such as its sensitivity to the form of the model and the fact that it is linked to the error rate of the system indirectly. To alleviate these shortcomings, Biem proposes to use the Minimum Classification Error (MCE) criterion along with an allograph-based HMM [8]. The MCE-based training is a discriminative training method where the aim is to find a set of parameters that minimize the empirical recognition error rate. The recognition system is trained with 52K samples from more than 100 writers for a 5K-word lexicon without any pruning and 10K-word lexicon with pruning. Compared to a ML-based baseline, 9.75% and 6.19% relative word error rate reductions are obtained for the 5K lexicon and 10K-lexicons respectively.

Most of the online handwriting recognition systems obtain the input data via a digitizer like a Personal Digital Assistant (PDA) and Tablet PC which captures information about the pen tip position, velocity, or acceleration as the user writes on the input surface. To bring handwriting recognition to the classrooms, Liwicki and Bunke use online handwritten text acquired by capturing the trajectory of a pen writing on a whiteboard [9]. Although the data is in on-line format, they transform it to offline form in order to remove writing order variations. An HMM offline handwriting recognizer which is designed for

the offline IAM database is then used for recognition, achieving a word recognition rate of 64.3% on a small dataset with 1000 words. A bigram language model derived from the LOB corpus is employed in the recognizer. In more recent work, the authors achieve better results by increasing the training set size [10]. Using again a bigram language model, the recognition rate is raised to 68.6% on the IAM-On database.

1.3.2 HMM-NN Hybrid Recognizers

HMM systems are capable of modeling dynamic time sequences of variable lengths, which makes them appropriate for the handwriting recognition task. However, they cannot make use of the context such as inter-symbol dependencies (e.g. how the letter 'e' appears after the letter 't'). In contrast, artificial neural networks in their most common form (multi-layer perceptrons) are able to capture the contextual information while lacking the ability to handle time varying sequences and their statistical variations. Hybrid approaches combine the strengths of both methods: in HMM-NN-based systems, NNs are used for frame/character classification, while HMMs are used for modeling the whole sequence.

Garcia-Salicetti et al. takes an original approach to HMM-NN combination and propose a hybrid system of handwritten word recognition based on HMMs with integrated NNs that are related to letter HMMs [14]. HMMs and NNs are simultaneously initialized and trained. By embedding NNs into the letter HMMs, their system remains within the HMM framework while extending the HMM with contextual information. They use Maximum Mutual Information (MMI) criterion for training the NNs. The system works in a writer-dependent framework with vocabulary sizes of 1,000, 10,600, 20,200. Word recognition rates for the dictionaries are reported as 96.93%, 91.13% and 88.67% respectively.

In the HMM-based system proposed by Schenk and Rigoll, a NN is used as part of feature extraction stage of an online handwriting recognition system [13]. After the feature extraction, a standard HMM is applied. On a 1,500-word database, they achieve 95.9% recognition rate using multi-layer perceptron networks and 95.2% recognition rate using a recurrent neural network architecture. In both cases a lexicon of 2,000 words is used for final word recognition.

In [12], Marukatat et al. use NNs to predict the emission probabilities in a hybrid system. They train the system with 30K words by 256 writers from UNIPEN dataset. With a 2,000-word lexicon, they report 80.1% and 77.9% word recognition rates for multi-writer and writer-independent (omni-writer) recognition tasks respectively.

Another work where this hybrid approach is used is [19]. Gauthier et al. propose classifier combination with an online HMM-NN and an offline HMM system. The offline recognizer uses the offline version of the online text. Using 40K words written by 256 writers from UNIPEN dataset for training, the combined classifier is tested on *parts of* a 1K sample set written by the *same* training set writers. They report a 87% recognition rate with a 1,500-word lexicon, which decreases to 79% when the lexicon size is increased to 10,000 words.

1.3.3 HMM- MSTDNN Hybrid Recognizers

Another type of hybrid systems used for online handwriting recognition task is a combination of a multi-state time delay NN (MSTDNN) with an HMM. Time delay NNs (TDNN) are time invariant NNs which can recognize a pattern regardless of its position in time. MSTDNN is an extension of TDNN with the dynamic time warping algorithm, aimed to integrate recognition and segmentation into a single architecture.

Jaeger et al. use MSTDNNs in conjunction with HMMs [20]. A tree representation of the dictionary employs HMMs that represent individual characters. Using an efficient search algorithm over the tree representation of the dictionary, they achieve a 96% word recognition rate with a 5,000-word dictionary. The recognition rates for 20,000 and a 50,000-word dictionaries are 93.4% and 91.2% respectively, using a combination of handwriting databases of Carnegie Mellon University, University of Karlsruhe and MIT.

Caillaut and Gaudin use the MSTDNN for computation of character likelihoods and observation probabilities in the HMM recognizer [18]. Searching for the best training scheme for the hybrid system, they find a new generic criterion combining Maximum Likelihood (ML) and Maximum Mutual Information (MMI) criteria with a global optimization approach defined at the word level. They achieve a 92.78% word recognition rate on IRONOFF word database.

1.3.4 BLSTM-based recognizers

Bidirectional Long Short Term Memory (BLSTM) architecture is another approach used for online handwriting recognition. BLSTMs consist of multiple recurrent neural networks (RNNs) for capturing long-term dependencies in both past and future context during recognition.

The earliest work proposing an RNN with BLSTM architecture for unconstrained handwritten word recognition is Liwicki et al. [15]. They use the Connectionist Temporal Classification (CTC) objective function in the recognizer, in order to correct errors made at character level and recognize the word. They achieve 74% word accuracy on the IAM-OnDB which contains forms of handwritten English text acquired on a whiteboard. The dictionary size is 20,000 consisting of the most frequent words from LOB, Brown and Wellington corpora.

In another study, Liwicki and Bunke apply feature selection to improve a BLSTM recognizer's performance [22]. The best combination of features are searched via sequential forward and sequential backward searches over a set of 25 features. They report 75.2% word recognition rate with a BLSTM recognizer on IAM-OnDB dataset using 17 selected features. Comparatively, an HMM classifier achieves 73.8% recognition rate with 16 selected features.

Graves et al. improve performance of a similar BLSTM system by integrating a bigram language model at connectionist temporal classification layer to achieve a recognition rate of 79.7% for a 20,000-word open lexicon [17]. A better result of 85.3% accuracy is achieved with a limited closed dictionary of size 5,597, without a language model. The BLSTM recognizer is compared to a state-of-the-art HMM recognizer and shown to out-

Table 1.1: Previous results on word recognition accuracies obtained on public databases.

Author	Method	DB	Test DB Size	Lexicon Size	Writer dependency	Acc. (%)
Jaeger et al. [20]	HMM-MSTDNN	CMU-UKA-MIT	4,105	20,000	I	93.4
				50,000	I	91.2
Caillaut et al. [18]	HMM-MSTDNN	IRONOFF	10,448	-	I	92.7
Gauthier et al. [19]	HMM-NN	UNIPEN	1,000	1,500	I	87.0
This work	HMM		7,000	1,000	I	86.1
This work	HMM		7,000	3,500	I	83.0
Marukatat et al. [12]	HMM-NN		?	5,000	I	77.0
Marukatat et al. [12]	HMM-NN		?	5,000	D	75.0
Liwicki et al. [21]	HMM	IAM-ON	6,204	2,337	I	70.8
Liwicki et al. [15]	BLSTM		?	20,000	I	74.0
Liwicki et al. [10]	HMM (offline features)		1,240	11,050	I	68.6
Liwicki et al. [22]	BLSTM		?	20,000	I	75.2
Graves et al. [16]	HMM and BLSTM		3,859 lines	20,000	I	79.6
Graves et al. [17]	BLSTM		3,859 lines	20,000	I	79.7
Salicetti et al. [14]	HMM-NN	Proprietary	1,500	1,000	D	96.9
				10,600	D	91.1
				20,200	D	88.6
Vural et al. [23]	HMM (Turkish)	Proprietary	200	1,000	I	91.4
This work	HMM (Turkish)	Proprietary	804	1,956	I	91.7

perform it under all test conditions. The relative error reduction is reported to be as much as 40% in some cases.

A summary of some of the most recent results reported on public databases or those that are done on Turkish is given in Table 1.1. Unfortunately, most of these results discussed in this section are not directly comparable, since different researchers have used different datasets or different subsets of a dataset or different settings (multi or omni-writer problem). Nonetheless, it is intended to give an idea about the progress in the field.

1.3.5 Handwriting Recognition for Turkish

There is little research about recognition of handwritten Turkish text. A number of studies cover offline Turkish character recognition with some constraints like the style or the case [24, 25, 26].

In offline handwritten Turkish text recognition, Yanikoglu and Kholmatov use the HMM letter models previously developed for English, by mapping the Turkish characters to the closest English character (the input of the word güneş is recognized as gunes). They report 56% top-10 word recognition rate using an 17,000-word lexicon obtained from a newspaper corpus. Resembling the work in [27], [28] use character-based word recognition method for offline lowercase mixed-style handwritten Turkish words and achieve 84% recognition rate by using a dictionary of size 2,500.

In online handwriting recognition, Vural et al. obtain 94% word recognition rate using a lexicon of 1,000-word lexicon and report that about 35% of the errors are due to delayed strokes [23].

1.4 Thesis Overview

With the ever increasing use of computers and digital appliances like tablet PCs or smartphones, input modalities evolve towards the more natural interactions like touch, speech, sketching and handwriting. Handwriting recognition in general and online handwriting recognition in particular is an active research area which have a direct impact on everyday technology. Products employing handwriting recognition functionality are widely in use. Educational applications which have been employing handwriting recognition ability for some time, are likely to benefit from any improvement in that research area. Nevertheless, there is a lack of research on recognition of online handwritten text in Turkish and this work primarily aims to fill that gap with a state-of-the-art recognizer for the first time.

This thesis focuses on building an online handwriting recognizer with Hidden Markov Models for recognition of Turkish isolated words. It starts with development of a recognition engine with comparatively smaller vocabulary which is intended to be a part of an educational software for tablet PCs as described in [29]. Then, the system evolves to a large vocabulary recognition system by integration of state-of-the-art techniques of language modeling. The thesis provides solutions to the problems particular to recognition of Turkish language and script.

Chapter 2 presents theoretical background for Hidden Markov Models and their use for handwriting recognition. In Chapter 3, details of language modeling for handwriting recognition is explained. Two main problems in recognition of Turkish language script, delayed strokes and high rate of OOV are discussed in Chapter 4 and some solutions are proposed in Chapter 5. Datasets, text corpora and software resources are presented in Chapter 6. Chapter 7 describes the proposed recognition system as a baseline while Chapter 8 includes experimental results regarding each solution for attempted problems and the overall system performance. And finally Chapter 9 draws conclusions and suggests directions for future work.

Chapter 2

Hidden Markov Models for Online Handwriting Recognition

Hidden Markov Models (HMMs), which are initially applied to the automatic speech recognition (ASR) problem [30], have been a popular method for handwriting recognition as well. Capability of HMMs for time alignment and for the maximum likelihood formulation of the parameter estimation make them a suitable technique for modeling sequences in recognition problems in domains like speech, computational biology and online handwriting recognition. Starting from early 1990s ([31, 32]), HMMs became a preferable approach for script recognition; historically first for offline modality and then for online form ([33, 3]). In this section, first, a brief introduction will be made to HMMs theory. Next, use of HMMs for online handwriting recognition problem will be explained.

2.1 Hidden Markov Models

HMMs can be classified as discrete and continuous based on their observation densities. If the observations come from a categorical distribution, HMMs are said to be discrete. With continuous HMMs, observations are generated from a Gaussian distribution. For the sake of simplicity, the theory will be explained mostly with discrete HMMs. Later, application of the technique to online handwriting will cover continuous type of HMMs.

2.1.1 Definition

A Hidden Markov Model (HMM) is a statistical model in which a system is modeled with a set of unobservable/hidden states and probabilistic transitions between them. In HMMs, the sequence of states that the system goes through are unknown but states emit symbols from a predefined alphabet with a probability model which are observable themselves.

Discrete HMM formalism is a kind of discrete state Markov process with additional property of probabilistic outputs. A discrete state Markov process can be in one of n distinct discrete states, $\{S_1, S_2, ..., S_N\}$ at any given time. It satisfies the first order Markov

condition such that its future behavior depends only on its present state which brings out the *state-independence assumption*.

According to the first order Markov assumption, the probability of being in a state *Q* at time *n* only depends on the observation at time $n-1$:

$$P(Q_n = S_i|Q_{n-1} = S_j, Q_{n-2} = S_a, ..., Q_0 = S_b) = P(Q_n = S_i|Q_{n-1} = S_j), \forall i, j, a, b, n. \tag{2.1}$$

The process can be in any one of its discrete states initially at time t_0. Initial state probabilities of a discrete state Markov process with N states are shown as $\Pi = \{\pi_i\}$ where

$$\pi_i = P(Q_0 = S_i), 1 \leq i \leq N. \tag{2.2}$$

The process can make a transition from $state_i$ to $state_j$ with a probability a_{ij} at a discrete time. State transition probabilities are shown as $A = \{a_{ij}\}$ where

$$P(Q_n = S_j|Q_{n-1} = S_i), 1 \leq i \leq N \text{ and } \sum_j a_{ij} = 1, \forall i. \tag{2.3}$$

Π and *A* together define a discrete state Markov process. An extension with probabilities of observation symbol emission at visited states is required for definition of an HMM.

In HMM formalism, each state can emit an observation with a probability. Since observations are probabilistic, an observation sequence does not have a corresponding deterministic state sequence. In general, there are many possible state sequences which generate an observation sequence which makes the actual state sequence "hidden".

If a model has *N* distinct states , *M* distinct observation symbols in each state and O_n is the observation at time *n* and v_k is an event for which the observation symbol is *k* then the state observation probabilities $B = \{b_{iv_k}\}$ are

$$b_{iv_k} = P(O_n = v_k|Q_n = S_i), 1 \leq i \leq N \text{ and } 1 \leq k \leq M. \tag{2.4}$$

It is assumed that the output observation at a given time is dependent only on the current state, it is independent of previous observations and states which can be stated as

$$P(O_n = v_k|O_{n-1} = v_a, O_{n-2} = v_b, \ldots, O_0 = v_c, Q_n = S_k) = P(O_n = v_k|Q_n = S_k), \forall k, a, b, c, n. \tag{2.5}$$

With the extension of state observation probabilities over discrete state Markov process, a HMM λ is formally defined by the following elements:

- A set of states $Q = S_1, ..., S_N$.
- A transition model, defined by the probabilities $P(Q_t = S_i|Q_{t-1} = S_j)$, for all S_i , $S_j \in Q$ where $P(Q_t = S_i)$ denotes the probability of the process being in state S_i at time *t*.
- A probability distribution over initial states $P(Q_1 = S), \forall S \in Q$.
- An emission model, defined by the probabilities $P(X|S)$, where $S \in Q$, and *X* is an observation from the alphabet. In case of discrete HMMs alphabet is a finite set and it is set of real numbers $\mathbb{R}^D$ with D dimension of observations in continuous HMMs.

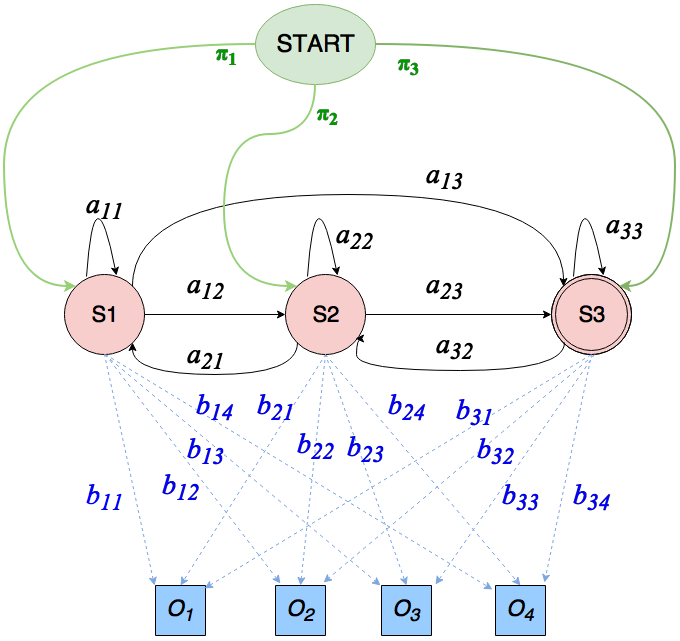

Figure 2.1: A HMM with three states and 4-symbol observation alphabet. Observation probability notation is simplified to b_{ij} as the probability of emitting *jth* symbol at *ith* state

2.1.2 Three Basic Problems of HMMs

HMMs are characterized by three fundamental problems:

1. The Evaluation Problem: Given an HMM, $\lambda = (A, B, \Pi)$ and an observation sequence $O = \{O_1 O_2 \dots O_T\}$, what is $P(O|\lambda)$, the probability of the observation sequence *O* ?
2. The Decoding Problem: Given an observation sequence $O = \{O_1 O_2 \dots O_T\}$, and an HMM $\lambda = (A, B, \Pi)$ what is the optimal state sequence $Q = \{Q_1 Q_2 \dots Q_T\}$ which maximizes $P(Q, O|\lambda)$?

3. The Training Problem: Given an observation sequence $O = \{O_1 O_2 \dots O_T\}$, what is the optimal model λ which maximizes $P(O|\lambda)$?

The Forward Algorithm; A Solution to The Evaluation Problem

In HMM systems, each hidden state produces only a single observation but the actual state sequence $Q = \{Q_1 Q_2 \dots Q_T\}$ is unknown for a given observation sequence $O = \{O_1 O_2 \dots O_T\}$. In order to compute probability of a particular observation, $P(O|\lambda)$, given the model parameter λ, probability of observation of *O* is summed up over all possible *Q*:

$$P(O|\lambda) = \sum_{allQ} P(O, Q|\lambda) = \sum_{allQ} P(O|Q, \lambda) P(Q|\lambda). \tag{2.6}$$

$P(Q|\lambda)$ can be calculated from probability of being in each state in *Q* by using the state independence assumption 2.1 :

$$P(Q|\lambda) = \pi_{Q_1} a_{Q_2 Q_1} a_{Q_3 Q_2} \dots a_{Q_t Q_{t-1}}. \tag{2.7}$$

Similarly, by the output independence assumption 2.5, $P(O|Q, \lambda)$ is calculated as:

$$P(O|Q, \lambda) = b_{Q_1 O_1} b_{Q_2 O_2} \dots b_{Q_T O_T}. \tag{2.8}$$

Rewriting 2.6 using 2.7 and 2.8, the probability of an observation sequence given a model is obtained by summing over all state sequences:

$$P(O|\lambda) = \sum_{allQ} (b_{Q_1 O_1} b_{Q_2 O_2} \dots b_{Q_T O_T}) \cdot \left(\pi_{Q_1} a_{Q_2 Q_1} a_{Q_3 Q_2} \dots a_{Q_t Q_{t-1}}\right). \tag{2.9}$$

It is infeasible to sum over all possible state sequences, especially when the number of possible states *N*, or the length of the observation sequence *T* increases. Instead of the naive method which has a complexity $O(2TN^T)$, a dynamic programming approach can be employed with much efficiency. The forward algorithm provides an efficient solution by a recursive computation of so-called forward variables with a complexity of $O(N^2 T)$.

A forward variable, $\alpha_t(j)$, keeps the probability of being in state *j* after seeing the first *t* observations, at time *t*, given the model λ.

$$\alpha_t(j) = P(O_1 O_2 \dots O_t, Q_t = S_j|\lambda). \tag{2.10}$$

With the dynamic programming approach, each forward variable is computed from the forward variable value of its previous time step. Following is the recursive definition for forward variables:

$$\alpha_t(j) = \begin{cases} \pi_j b_{jo_1} & t = 1, 1 \leq j \leq N \\ \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} bj(o_t); & 1 \leq j \leq N, 2 \leq t \leq T \end{cases} \tag{2.11}$$

After calculation of forward variables for all states at time T, probability of a sequence of length *T* is computed by summation of $\alpha_T(j)$ values:

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_T(j). \tag{2.12}$$

The Viterbi Algortihm: A Solution to the Decoding Problem

Given an observation sequence $O = \{O_1 O_2 \dots O_T\}$ and a model λ, the most probable sequence of states that have generated *O* , $Q = \{Q_1 Q_2 \dots Q_T\}$, can be computed naively by running the forward algorithm for all possible hidden state sequences. However, it is again infeasible for many real tasks due to exponentially large number of state sequences. The Viterbi algorithm, which is based on dynamic programming paradigm provides an efficient solution to the decoding problem.

Much like the forward variables of forward algorithm, the Viterbi algorithm keeps calculations at each time step as intermediate values and builds up on them to reach a final calculation. An intermediate value $v_t(j)$ represents the maximum probability of being in state *j* after seeing the first *t* observations by passing through the most probable state sequence $Q_1 Q_2 \dots Q_{t-1}$ at time *t*, given the model λ.

$$v_t(j) = \max_{Q_1 Q_2 \dots Q_{t-1}} P(Q_1, Q_2, \dots, Q_{t-1}, O_1, O_2, \dots, O_{t-1}, Q_t = j|\lambda). \tag{2.13}$$

The recursive definition of $v_t(j)$ values make use of the calculations from the previous time steps:

$$v_t(j) = \begin{cases} \pi_j b_{jo_1} & t = 1, 1 \leq j \leq N \\ \max\limits_{1 \leq j \leq N} v_{t-1}(i) a_{ij} bj(o_t), & 1 \leq j \leq N, 2 \leq t \leq T \end{cases} \tag{2.14}$$

At the end of process, $\max\limits_{1 \leq j \leq N} v_T$ value gives the maximum probability *P** of seeing the given observation sequence by passing through the most optimal state sequence *Q**. Keeping track of hidden states that led to each state by means of back-pointers allows to recover the optimal state sequence with backtracking from the end state to the beginning.

The Baum-Welch Algorithm; A Solution to Training Problem

Given an observation sequence, $O = O_1 O_2 \dots O_t$, computation of initial, transition and observation probabilities that maximize the probability of the observation sequence, $P(O|\lambda)$ is the training problem of HMMs.

The Baum-Welch algorithm [34, 35], which is also known as the forward-backward algorithm is the standard method of training HMMs. The algorithm itself is a version of Expectation-Maximization (EM) algorithm where starting off with the initial estimates for the transition and observation probabilities, it iteratively computes expected state occupancy count and expected state transition counts and then re-estimates the probabilities from the calculated values.

Maximum likelihood estimation of the probability a_{ij} of a particular transition between states *i* and *j* can be calculated by counting the number of times the transition takes place divided by the total number of any transitions from state *i*:

$$a_{ij} = \frac{\text{number of transitions from } Q_i \text{ to } Q_j}{\sum_{Q \in \mathbf{Q}} \text{number of transitions from } Q_i \text{ to } Q} \tag{2.15}$$

These counts cannot be computed directly in an HMM because the states are hidden and the sequence of states that generate a given input is unknown. The forward variable and a similar backward variable are used to compute estimations for these counts in The Baum-Welch algorithm.

A backward variable, $\beta_t(i)$, represents the probability of seeing the partial observation sequence $O_{t+1}O_{t+2}\ldots O_T$ at time *t* and state *i* in a given HMM λ:

$$\beta_t(i) = P(O_{t+1}O_{t+2}\ldots O_T | Q_t = S_i, \lambda). \tag{2.16}$$

Computation of backward variables is done recursively:

$$\beta_t(i) = \begin{cases} 1 & t = T, 1 \leq i \leq N \\ \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} b_j(o_{t+1}); & 1 \leq i \leq N, 2 \leq t \leq T \end{cases} \tag{2.17}$$

Probability of a given observation sequence $P(O|\lambda)$ can be defined in terms of both forward (2.11) and backward variables as :

$$P(O|\lambda) = \sum_{j=1}^{N} \alpha_T(j) = \sum_{i=1}^{N} \beta_1(i)\pi_i. \tag{2.18}$$

An estimation for a_{ij} can be expressed as :

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from } Q_i \text{ to } Q_j}{\sum_{Q \in \mathbf{Q}} \text{expected number of transitions from } Q_i \text{ to } Q} \tag{2.19}$$

Expected number of transitions from Q_i to Q_j can be expressed as the sum of probabilities of making a transition from state *i* to state *j* at time *t* for all time steps in the

observation sequence. $\xi_t(i,j)$ is defined as the joint probability of being in state *i* at time *t* and in state *j* at time $t+1$, given an observation sequence *O* and a model λ:

$$\xi_t(i,j) = P(Q_t = S_i, Q_{t+1} = Sj|O,\lambda). \tag{2.20}$$

$$\xi_t(i,j) = \frac{P(Q_t = S_i, Q_{t+1} = Sj, O|\lambda)}{P(O|\lambda)} \tag{2.21}$$

The numerator term of 2.21 can be expressed with forward and backward variables and transition probability of corresponding states:

$$P(Q_t = S_i, Q_{t+1} = Sj, O|\lambda) = \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j). \tag{2.22}$$

Finally, $\xi_t(i,j)$ is defined as:

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}. \tag{2.23}$$

Using definition in 2.23, estimated transition probability $\hat{a}_{ij}$ in 2.19 can be expressed as:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{j=1}^{N}\xi_t(i,j)}. \tag{2.24}$$

Estimation of observation probability of symbol v_k at state *j*, $\hat{b}_j(v_k)$ can be explained as:

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \tag{2.25}$$

Probability of being in state *j* at time *t*, $\gamma_t(j)$ with a given observation sequence *O* and a model λ is defined as:

$$\gamma_t(j) = P(Q_t = S_j|O,\lambda) = \frac{P(Q_t = S_j, O|\lambda)}{P(O|\lambda)} = \frac{\alpha_t(j)\beta_t(j)}{P(O|\lambda)}. \tag{2.26}$$

The expected number of times the model is in state *j* and symbol v_k is observed is calculated by summing $\gamma_t(j)$ up for all time steps *t* in which the observed symbol is v_k. Following is the rewriting of equation 2.25 with $\gamma_t(j)$ notation:

$$\hat{b}_j(v_k) = \frac{\sum_{t=1\cap O_t=v_k}^{T\cap O_t=v_k}\xi_t(j)}{\sum_{t=1}^{T}}. \tag{2.27}$$

The forward-backward algorithm calculates iteratively first ξ and γ from current transition and observation probabilities, *A* and *B*, of the model λ and then use the resulting values to make estimations $\hat{a}$ and $\hat{b}$ to recompute *A* and *B*.

2.1.3 HMMs for Online Handwriting Recognition

Online handwriting is represented with states in a HMM system. According to the chosen recognition unit, a set of states and transitions between them can represent words, characters or sub-characters. Characters are the natural unit for many scripts.

When the recognition unit is chosen as the characters, each training word or sentence is represented by concatenating a sequence of character models together to form a one large model. In Figure 2.2, words $\{$*ONE, TWO, THREE* ... *TEN*$\}$ are represented with concatenation of character HMMs and each character is modeled by three states.

Tasks of handwriting recognition process can be expressed as special cases of the three problems of HMMs. For example, given a set of character models λ and an observation sequence *O*, character recognition is evaluating each of the model λ for *O* to find the model that have the maximum probability, $P(O|\lambda)$. During the training phase, for each training sample, a composite model is effectively synthesized by concatenating the phoneme models given by the transcription of that sample. The parameters of the character models are then re-estimated using the Baum-Welch algorithm.

On the other hand, word recognition with character-based HMMs is the decoding problem; given a set of character models λ, the optimal sequence of states Q* corresponding to the observation sequence is found using the Viterbi algorithm. Using a lexicon of possible words in the form of a word network, the most probable match is searched for a given test sample. Again, the words in the network are represented with concatenated character models as shown in Figure 2.2. Sentence recognition is similar to word recognition with the addition of a special model for *space* character.

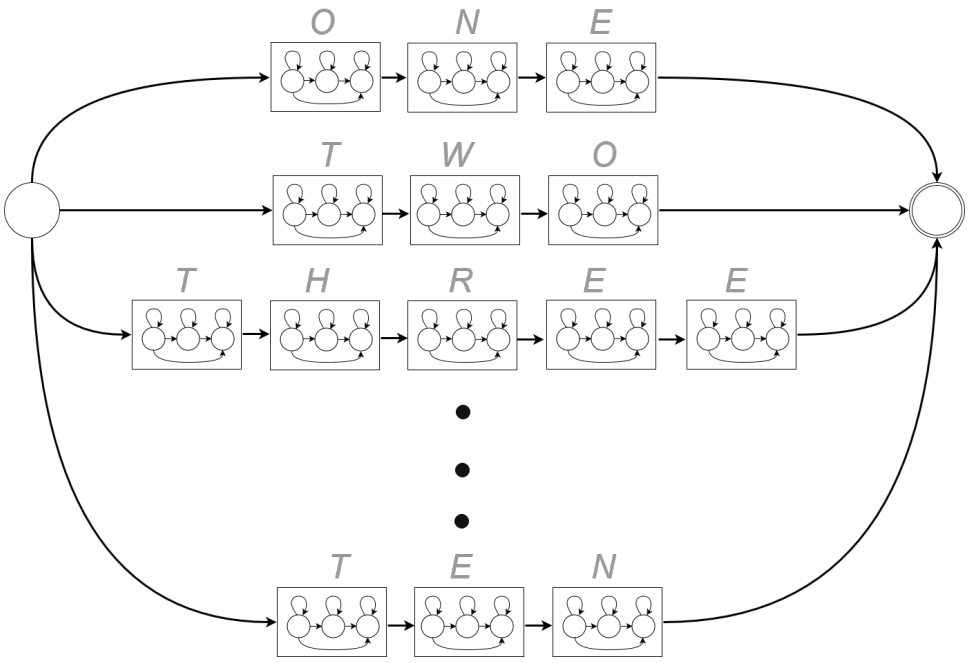

Figure 2.2: Word HMM as concatenation of character HMMs

HMM Configuration

As a graphical model, the design of topology is an important part of configuration in HMMs. The term topology defines valid transitions between states within the context of HMMs. Figure 2.3 shows some examples of possible topology designs. Ergodic or fully-connected topology where transitions between any two states is allowed is generally not suitable for handwriting recognition since handwriting data is chronologically organized. The models where transitions are limited to potentially relevant ones and irrelevant ones are suppressed by hard-coded transition probability of 0 are better in describing handwriting. Linear and Bakis type are the most common HMM topologies in handwriting recognition. A transition can represent 1) a progress in time from Q_i to Q_j where $j > i$, 2) a self transition to the same state to match variable duration of a segment, 3) a skip one or more (usually two) states forward to match optional or missing parts.

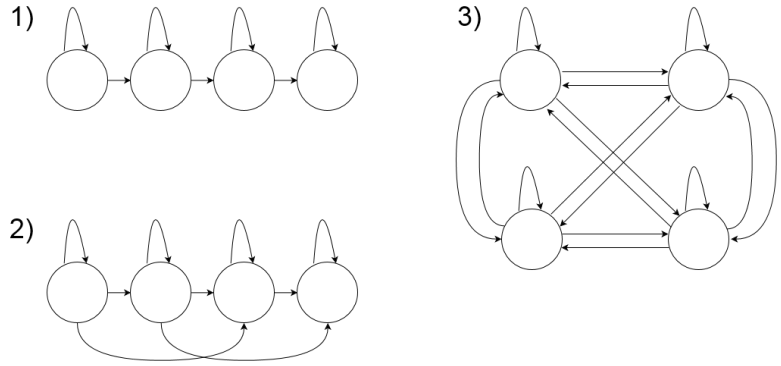

Figure 2.3: HMM topologies : 1) Linear 2) Bakis-type 3) Ergodic

The number of states per model can be the same for all HMMs or can be designed according to the complexity of models. For example, in case of character models, number of states can be adjusted to acknowledge the different length of characters. Word models are formed by concatenating character models end-to-end.

For most of the pattern recognition applications, output are real-values coming from distributions over $\Re$ which do not have parametric descriptions. Mixture of densities are used for approximating output probabilities in such system. Gaussian mixture models, which are very common in HMMs, can be defined as:

$$p(x|s) = \sum_{m=1}^{M} c_{sm} N(x; \mu_{sm}, \Sigma_{sm})$$

where $N(.; \mu, \Sigma)$ is a multivariate Gaussian distribution with μ mean and Σ covariance matrix. The subscript notation *"sm"* indicates m-th mixture component in GMM of state *s*. The number of Gaussian mixtures is a crucial part of the HMM configuration. It is usually decided using an iterative approach as incrementation by splitting during training of models ([36]).

Although it is very common to choose HMM configurations based on experiments and heuristics, there are some other methods to adapt number of states, topology and number of mixtures to model structure automatically [37, 38, 39, 36]. The effects of individual configuration components are also reported in some studies [40, 41]. While the number of states per model is an important part of HMM configuration, the topology is found to have a stronger influence in improving the modeling capability of HMMs.

Chapter 3

Language Modeling

Language modeling is an essential part of all modern recognition systems. It is used for improving the recognition results by imposing some constraints on the decoding procedure and the output of the system.

During the HMM decoding process, the most probable path is searched through a probabilistically scored time/state lattice (see 2.1.2). A list of words comprising the lexicon is used for limiting the search of probable paths to valid words designated for that system. The size of the lexicon is typically thousands of words, for *large vocabulary* recognition systems. A task related to the language modeling is to generate an appropriate recognition lexicon for the particular task, which is covered in Chapter 4.

It is not sufficient to constrain the output, but the order of the elements in the output should be correct as well. For example, not every sequence of valid words make a grammatical sentence. It is necessary to have a means of selecting the correct ordering of lexicon items. Another main motivation of language modeling is to choose the most likely observation sequence by likelihood estimation.

There are different methods of modeling sequence probabilities in a language. Statistical language modeling is a well-founded and popular approach in speech and handwriting recognition [42]. Statistical language models that are based on N-gram statistics have been the dominant approach in language modeling because of their simplicity and low computational complexity. Other methods include Maximum Entropy Language Model (ME LM) [43] and Neural Network based language models [44]. In this work, N-gram models are utilized for language modeling.

3.1 N-gram Language Modeling

An N-gram is an N-token sequence of lexicon elements such as letters, syllables and words. An N-gram model indicates the conditional probability of observing a word given the history of the previous $N-1$ words [45]. The chain rule of probability provides a way for the calculation of members of a joint distribution of some random variables $x_1 \ldots x_n$, using conditional probabilities:

$$P(x_1 \dots x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1x_2) \dots P(x_n|x_1 \dots x_{n-1}) = \prod_{k=1}^{n} P(x_k|x_1 \dots x_{k-1}). \tag{3.1}$$

Using the chain rule, a relationship can be established between the joint probability of a sequence of words and the conditional probability of a word given the previous ones. The probability of a sequence of words, $(w_1, w_2, ..., w_N)$ can be estimated by multiplying the conditional probabilities of the words on their history of previous words. as :

$$\begin{aligned} P(w_1 \dots w_n) &= P(w_1)P(w_2|w_1)P(w_3|w_1w_2) \dots P(w_n|w_1 \dots w_{n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_1 \dots w_{k-1}). \end{aligned} \tag{3.2}$$

Computation of exact probability of a word, given a long sequence of preceding words $P(w_n|w_1 \dots w_{n-1})$, is neither feasible nor always possible for larger values of N. Instead of computing the probability of a word given its entire history, an approximation is possible by limiting the history to a few words. For example, a *bi-gram model (2-gram)* uses a history of one word while a *tri-gram model (3-gram)* covers the previous two words in its history. In general, a *N-gram* model approximates the conditional probability of the next word by using previous $N - 1$ words. As the value of N increases, N-gram models are more successful in modeling the training data since they use longer context, at the cost of increased complexity.

With bi-gram models, the probability of a word given all the words previous to it $P(w_n|w_1 \dots w_{n-1})$, is approximated by using only the the conditional probability of $P(w_n|w_{n-1})$ words:

$$P(w_n|w_1 \dots w_{n-1}) \approx P(w_n|w_{n-1}). \tag{3.3}$$

The probability of a complete word sequence is:

$$P(w_1 \dots w_{n-1}) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}). \tag{3.4}$$

The general equation of N-gram approximation for the conditional probability of the next word of a given sequence is:

$$P(w_n|w_1 \dots w_{n-1}) \approx P(w_n|w_{n-N+1} \dots w_{n-1}). \tag{3.5}$$

3.1.1 Estimation of N-gram model parameters

Parameters of an N-gram model can be estimated with the Maximum Likelihood Estimation (MLE). Using bi-gram modeling, the probability estimation of a given a sequence of words $(w_{n-1}w_n)$ is calculated as the count of the sequence normalized by sum of counts for all word sequences starting with w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum C(w_{n-1}w)}. \tag{3.6}$$

which can be further simplified to :

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}. \tag{3.7}$$

In general, based on normalization of observation frequencies, i.e. *relative frequencies*, MLE parameter for the N-gram probabilities can be written as:

$$P(w_n|w_{n-N+1}\dots w_{n-1}) = \frac{C(w_{n-N+1}\dots w_{n-1}w_n)}{C(w_{n-N+1}\dots w_{n-1})}. \tag{3.8}$$

Parameters that are estimated by the MLE, as its name suggests, maximizes the likelihood of the training set T, given a model M, $P(T|M)$.

3.1.2 Interpolation and Backoff

Interpolation and backoff are two mechanisms to estimate the higher-order N-grams that suffer from data sparsity, from lower-order probabilities. The backoff model uses the lower-order counts only if a higher-order count is zero. Interpolation, on the other hand, estimates the probability from all of the N-grams, such that counts of all N-gram orders are mixed to do a weighted interpolation. The weights are learned from a held-out dataset.

3.1.3 Smoothing

Since the maximum likelihood estimation of the model parameters is based on assigning probabilities from counts in a limited training set, N-gram sequences that are missing from the training set are set to zero probability. However, any valid sequence should have a non-zero probability.

The smoothing process modifies the MLE approach to assign non-zero probabilities to any N-gram, even if it is not observed in the training set. There are a number of smoothing algorithms like Laplace Smoothing, Good-Tuning Discounting, Witten-Bell Discounting and Kneser-Ney Smoothing. In this work, the Kneser-Ney method will be used for bi-gram and tri-gram models.

Kneser-Ney Smoothing

The Kneser-Ney method [46] is a backoff model based on absolute discounting. Absolute discounting is an interpolation method but it subtracts a fixed discount $\delta \in [0, 1]$ from non-zero counts, instead of multiplying with a weight.

With the Kneser-Ney method, instead of the unigram MLE count, a heuristic that can more accurately estimate the number of times a word w is expected to be seen in a new, unseen context is used for the backoff distribution. That heuristic is the *number of different contexts the word w appears in*. The more a word appears in different context, the more it is likely to appear in a new unseen context. The backoff probability of Kneser-Ney method, which is named as "continuation probability" is given as:

$$P_{CONT}(w_i) = \frac{|\{w_{i-1} : C(w_{i-1}w_i) > 0\}|}{\sum_{w\,i}|\{w_{i-1} : C(w_{i-1}w_i) > 0\}|} \tag{3.9}$$

and the Kneser-Ney probability is:

$$P_{KN}(w_i|w_{i-1}) = \begin{cases} \frac{C(w_{i-1}w_i)-\delta}{C(w_{i-1})}), & \text{if } C(w_{i-1}w_i) > 0 \\ \alpha(w_i)P_{CONT}, & \text{otherwise} \end{cases}$$

with a suitable discount value δ and a coefficient α on the backoff value which makes the sum of all probabilities equal to 1.

3.2 Perplexity

Statistical language models are usually evaluated on the basis of an intrinsic measure named as *perplexity*. Perplexity is the metric of how well a probability model (i.e. language model) predicts probability of the test data. A more intuitive definition can be a measure of on average how many different equally most probable words can follow any given word. For a test set with *N* words $w_1, w_2, ..., w_N$, the perplexity of the model on the test set is defined as probability of the test set, normalized by the number of words, N :

$$\begin{aligned} PP &= (P(w_1, w_2, ..., w_N))^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1, w_2, ..., w_N)}} \end{aligned} \tag{3.10}$$

using the chain rule to expand W in Equation 3.10:

$$\begin{aligned} PP &= (P(w_1, w_2, ..., w_N))^{-\frac{1}{N}} \\ &= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1, ..., w_{i-1})}} \end{aligned} \tag{3.11}$$

If the test set is the whole sequence of words, one after the other, then the sentence boundaries are also taken into consideration in probability computation through the beginning and ending markers i.e. $< s >$ and $< /s >$. Ending marker $< /s >$ is included in the total count of word tokens N as well.

As can be seen from Equation 3.11, minimizing the perplexity means maximizing the test set probability according to the language model. The model that assigns the highest probability to the test data predicts it more accurately. Based on this, lower perplexity generally indicates a better model. However, perplexity is not a definite way of determining the usefulness of a language model. A model with lower perplexity can be a better one at predicting the next word when a set of previous words are given in a test set. But its performance may not be the same on the real life data if the test set is not a good sample of the real life data. In general, perplexity is a useful metric for comparing language models on the *same* test data. Also, two models can be compared by perplexity only if they use the same vocabulary.

Perplexity is derived from the cross-entropy concept of the Information Theory [47]. *Entropy* is a measure of information which can be adapted in language processing to measure to how well a given grammar matches a given language, how much information does a particular grammar contain and how predictive a given N-gram grammar is about the next word could be [45].

Entropy of a sequence of variables, as in the case of a word sequence in a grammar, can be calculated through a variable that will range over these sequences.

Per-word entropy or *entropy rate* is the entropy of the sequence divided by the number of words in the sequence:

$$\frac{1}{n}H(w_1,\ldots,w_n) = -\frac{1}{n}\sum_{(w_1,\ldots,w_n)\in L} p(w_1,\ldots,w_n)logp(w_1,\ldots,w_n) \tag{3.12}$$

For calculation of the *true* entropy of a language, which can be accepted as a stochastic process, sequences of infinite length should be considered. Entropy rate of a language L is defined as:

$$\begin{aligned} H(L) &= -\lim_{n\to\infty}\frac{1}{n}H(w_1,,w_n) \\ &= -\lim_{n\to\infty}\frac{1}{n}\sum_{(w_1,\ldots,w_n)\in L} p(w_1,\ldots,w_n)logp(w_1,\ldots,w_n) \end{aligned} \tag{3.13}$$

If the language is ergodic and stationary[1], the summation over all possible word sequences can be discarded. In this case, a single, long-enough sequence of words can be used for estimation of the entropy and the entropy rate of a stochastic process.

$$(H)(L) = -\lim_{n\to\infty}\frac{1}{n}logp(w_1\ldots w_n) \tag{3.14}$$

[1]A language is stationary if the probability distribution of the words do not change with time and a language is ergodic if its statistical properties can be deduced from a single, sufficiently long sequence of words

When the actual probability distribution p that generated some data is not known, then approximation to the entropy is calculated as the *cross-entropy*. With the cross-entropy, a model m which approximates p is used for calculation of sequence probabilities. Cross-entropy of m on p is defined as:

$$H(p,m) = -\lim_{n\to\infty}\frac{1}{n}\sum_{(w_1,\ldots,w_n)\in L} p(w_1,\ldots,w_n)logm(w_1,\ldots,w_n) \tag{3.15}$$

According to Equation 3.15, sequences come from the probability distribution p but the log probabilities are calculated according to the model m. With the assumption that the process is stationary and ergodic as in Equation 3.14, the cross-entropy can be approximated by taking a single, sufficiently long sequence instead of summation of all possible sequences:

$$H(p,m) = -\lim_{n\to\infty}\frac{1}{n}logm(w_1,\ldots,w_n) \tag{3.16}$$

Since the probabilities are calculated according to the model m, the cross-entropy is an upper bound for the true entropy and $H(p) \leq H(p,m)$. Using that relation, two models can be compared over the cross-entropy values; the more accurate a model m, the lower the cross-entropy and the closer the cross-entropy to the entropy. Again, an approximation of cross-entropy of a model $M = P(w_i|w_1 \ldots w_N)$, is possible by using a sufficiently long sequence, W, with a fixed length N, instead of an infinite one:

$$H(W) = -\frac{1}{N}logP(w_1,\ldots,w_N) \tag{3.17}$$

The definition of perplexity (*PP*) of a model P on a sequence of words W is the exp of the cross-entropy:

$$\begin{aligned} PP(W) &= 2^{H(W)} \\ &= P(w_1,\ldots,w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1,w_2,...,w_N)}} \\ &= \sqrt[N]{\prod_{i=1}^{N}\frac{1}{P(w_i|w_1,...,w_{i-1})}} \end{aligned} \tag{3.18}$$

Perplexity can be seen as the weighted average branching factor of a language expressing the average number of equi-probable words that can follow a given word. It provides a measure on the prediction search space of the language, just as the entropy provides a measure of the size of a search space.

3.3 Integration of Language Models to The Decoding Process

The Viterbi Algorithm as explained in Section 2.1.2 use a simplifying assumption that if the ultimate best path for the entire observation sequence goes through a state q_i, then the best path up to and including q_i must be a part of the best path. That assumption limits the algorithm to make use of a bi-gram model, since a trigram model violates it by allowing the probability of a word to be based on the previous two words [45]. A best trigram path may not be lying on the global best path of the entire word sequence, even though one of its components does.

A solution for this problem is the *lattice expansion* method to change the lattice so that all of the N-grams appear on it. Another method of language model integration is to generate multiple hypothesis in the decoding process and then re-rank them using a higher order language model, which is called *lattice rescoring*.

3.3.1 Lattice Expansion

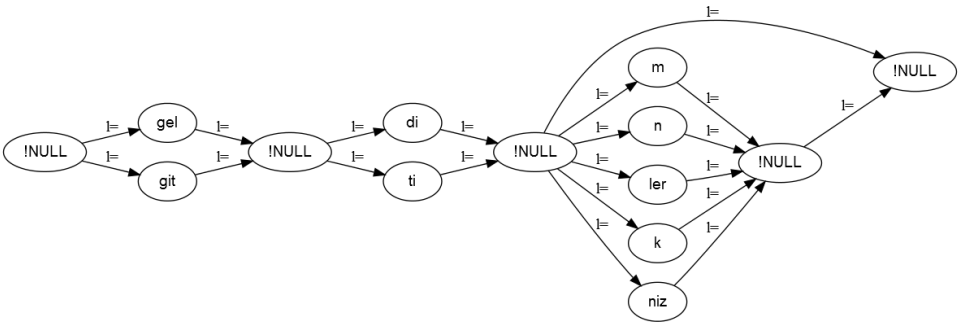

Figure 3.1: A simple lattice (network) with no language model scores

The lattice expansion method is based on adding new nodes and edges as duplications of existing ones to fulfill representation of all possible N-grams. The transitions between nodes are then used to reflect the language model. Lattice expansion can be used in single-pass decoding. In order to accommodate all N-grams in a given network, a unique $(N-1)$-word context should be created for each transition [48].

In Figure 3.1 a simple grammar network is drawn as a digraph. It has 14 nodes of which 5 is NULL nodes which are used for simplification of the network. In the initial setting no language model is applied to the network hence language model scores for transitions are empty. If the network lattice is expanded with a bi-gram language model, no new nodes are required for bi-gram score integration but the structure of the lattice and edges are fundamentally changed to represent each bi-gram uniquely, as shown in Figure 3.2. Finally in Figure 3.3, by integration of tri-gram scores to the original lattice, there are newly added nodes and many edges provide unique bi-gram context for each tri-gram transition. As can be seen from that example, lattice expansion with higher-order

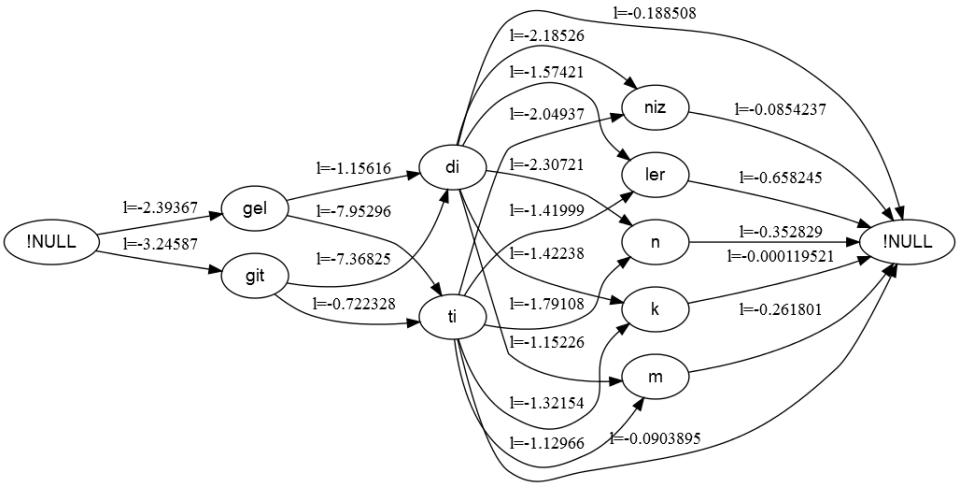

Figure 3.2: The simple lattice expanded with a bi-gram model

language models increases the complexity of the decoding lattice in terms of nodes and transitions between them.

3.3.2 Lattice Rescoring

As an alternative to single-pass decoding, transcriptions of a decoding process can be generated through gradual integration of the language model. With the multi-pass approach [49], a lattice can be generated with lower order knowledge sources, which are then rescored with higher order ones. Multi-pass decoding has been the dominating approach in automatic speech recognition domain for some time [50, 51]. It aims to lower the computational complexity of building the optimized search network in the first-pass, while achieving still a reasonable accuracy with rescoring.

Within the context of lattice rescoring, a lattice is a weighted directed acyclic graph with paths from the start state to a final state representing an alternative decoding hypothesis, weighted by its recognition score for a given handwriting input. A lattice is created for each test sample by running the decoder with some decoding network. Each hypothesis in the lattice contains the optical model likelihood and the language model probability separately represented. Each of these two scoring parts can be replaced with a more sophisticated model to achieve better accuracy [45].

The motivation for using higher-order knowledge on a lattice is that a better scoring will direct the decoding process to the path which will result in the correct transcription. However, the correct transcription should be included in the lattice at the first hand. The lattice error rate is a measure to understand the quality of a lattice. It is the word error rate if a perfect knowledge source leads the decoder to the path that has the lower error rate.

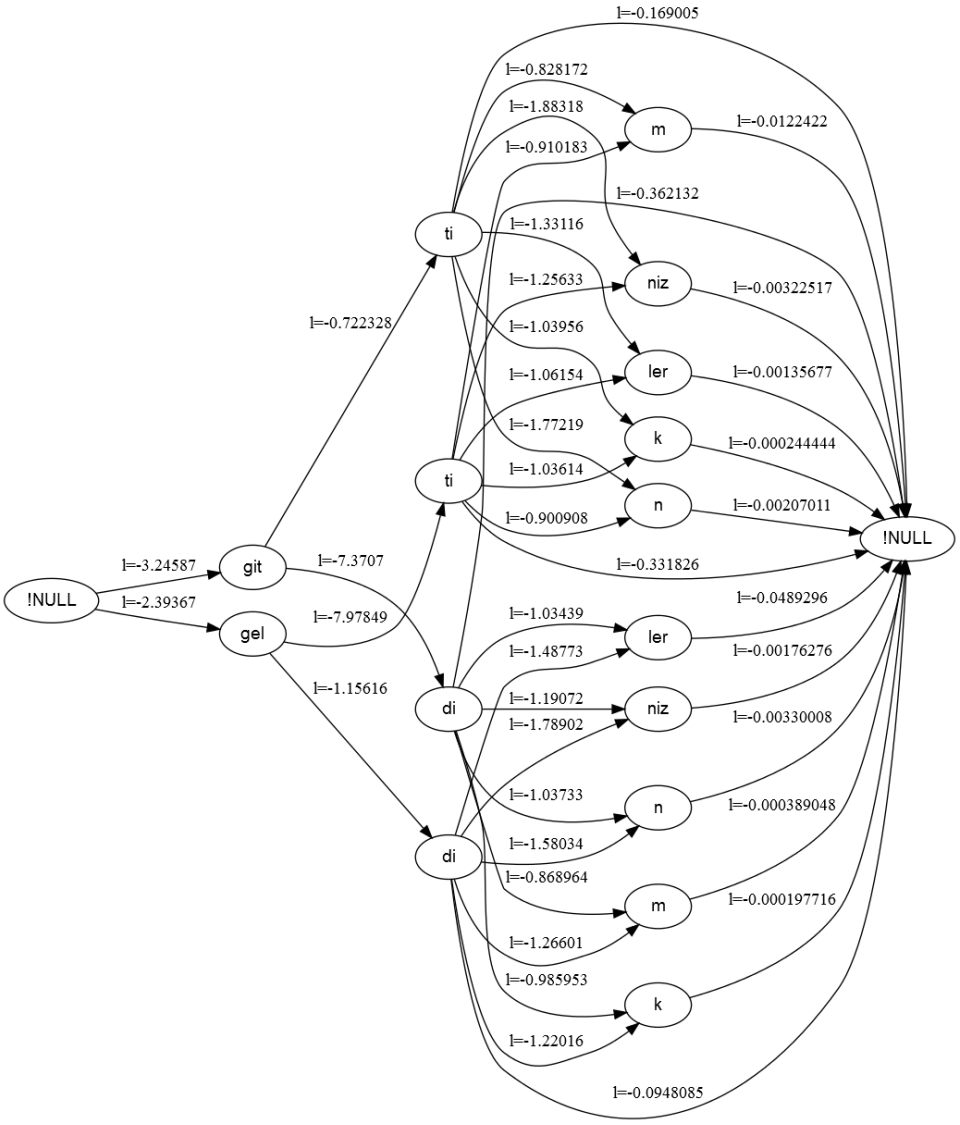

Figure 3.3: The simple lattice expanded with a tri-gram model

Chapter 4

Challenges in Turkish Online Handwriting Recognition

Turkish language is the most widely spoken member of Turkic languages. There are more than 80 million native speakers who live mostly in Turkey. Some other countries like Germany, Greece, Bulgaria and Northern Cyprus host considerable numbers of people who speak Turkish.

The main characteristics of the Turkish language are vowel harmony, lack of grammatical gender and extensive agglutination. Although the basic word order of Turkish is subject–object–verb, Turkish-in-use is considered to have a free word order which introduces a challenge in most Natural Language Processing (NLP) tasks.

Turkish alphabet has 29 letters: {'a', 'b', 'c', 'ç', 'd', 'e', 'f', 'g', 'ğ', 'h', 'ı', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'ö', 'p', 'r', 's', 'ş', 't', 'u', 'ü', 'v', 'y', 'z' }. Eight of them are vowels: { 'a', 'e', 'ı', 'i', 'o', 'ö', 'u', 'ü'} which are classified according to frontness-backness and roundedness-unroundedness. There is almost one-to-one relation between letters and phonemes, so the Turkish orthography is quite regular with a few exceptional cases.

New words are generated from roots by adding suffixes one after the other. Suffixes are either inflectional or derivational. Another classification of suffixes according to word category is either verbal or nominal. There are complex rules of morphotactics that govern the suffix order. Vowel harmony, consonant harmony, vowel ellipsis are the main orthographical and phonetic events that are observed with agglutination.

Looking from the perspective of handwriting recognition, Turkish script has many similarities with other Latin scripts like English which makes it possible to adapt strategies that work for them. However, there are some other issues which are particular to Turkish that should be taken into consideration separately.

The delayed strokes problem which is intensified with additional Turkish characters like 'ş','ç' and 'ö' is one of such problems. For example, Vural et al. report that about 35% of the recognition errors are due to delayed strokes in their handwriting recognition system [23]. The delayed strokes have detrimental effect on the optical modeling phase by addition of unnecessary variation. Also, they can affect the decoding process by introducing more confusion between similar-looking characters.

High productivity of Turkish creates another problem that arises in decoding phase. Turkish word formation is based on adding suffixes to a root word which can create a theoretically infinite lexicon. The closed-vocabulary recognition systems require a list of valid words to restrict the search space in deciding the most probable word as the output. Since it is not possible to list all words that can be generated in highly productive languages like Turkish, the number of Out-of-Vocabulary words is quite high in most of the cases which makes the simple word list based recognition not suitable for Turkish.

Another problem of Turkish related to the agglutinative nature of Turkish is the high level of confusability between word surface forms which increases with the vocabulary size. As words are created by affixation to root words, many words share the same stem and often differ only in suffixes by a single character. The problem is more severe when these suffixes have similar optical features, as in the example of *kitabımı* (my book-accusative) and *kitabını* (your/his/her book-accusative). In this case, the recognizer would have to chose between two alternatives with almost equal optical and linguistics scores. Even with 0% OOV rate, it is still difficult to make a correct recognition because of the high confusability in a large lexicon. While there is no solution to this problem, it is worth mentioning among problems pertaining to the recognition of Turkish text.

The problems arising from the delayed strokes and the high productivity of Turkish are discussed in detail in the following sections. Solutions proposed and developed within this work are presented in Chapter 5.

4.1 Delayed Strokes

One of the well-known problems in online handwriting recognition domain is the so-called *delayed strokes* that increase timing variations in online handwriting. A delayed stroke is "a stroke, such as the crossing of a 't' or the dot of an 'i', written in delayed fashion (not immediately after the corresponding character's body)". Writers have different writing practices as to when they write such strokes (right after the character body or after the word is written), which cause variations in the resulting sequence, which in turn degrades recognition performance. Each script has different strokes that are typically written in delayed fashion. These strokes can be either diacritical marks or integral parts of characters. Figure 4.1 shows an example word with delayed strokes.

Deciding whether a stroke is delayed or not requires knowing letter boundaries and cannot be done perfectly without full segmentation of a word into its letters. Nonetheless, delayed stroke detection and handling methods try to identify delayed strokes by their shape, size and locations, in order to reduce writing order variations.

Delayed strokes of Latin-based scripts can be investigated in three groups: 1) those that are written spatially above other strokes of the character, mostly without touching them, such as i-dots, umlauts (pair of dots) or other similar accents (e.g. accents grave and breve); 2) those that are written spatially below other strokes of the character, with or without touching them (e.g. cedilla and hook); and 3) those that are spatially overlapping with other strokes of the character, such as crosses of 'f','t','z', and 'x'. Turkish script has all the three types of delayed strokes in letters ('f','t','z', 'ç', 'ğ', 'i', 'j', ' 'ö', 'ş' and

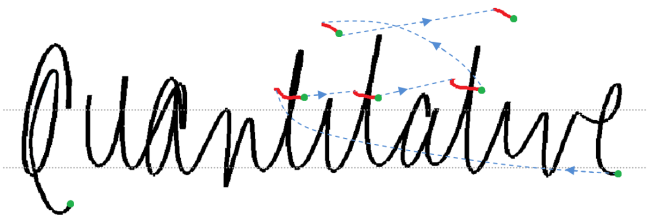

Figure 4.1: Delayed strokes in a sample of word 'Quantitative'. Red colored strokes are delayed. Pen-up positions are marked with green points. Dashed blue lines are the trace of the pen.

‘’ü’).

4.2 OOV and Vocabulary Explosion

The recognition dictionaries are designed to cover maximal number of words that can appear in the test data. Yet, there may still be some words that are not covered by the dictionary which are named as Out-of-Vocabulary (OOV) words. OOV words translates to errors in recognition, so it is important to chose a suitable dictionary size which minimizes OOV rates while still remaining manageable in terms of decoding complexity.

Vocabulary explosion can be defined as high rate of increase in vocabulary size of a language due to its morphological productivity. Agglutinative languages like Turkish, Finnish and Korean suffer from vocabulary explosion because of high numbers of words that are produced by derivational processes. In case of the recognition dictionaries, the inflection process contributes to vocabulary increase as well.

4.2.1 Turkish Morphology and Its Effects on The Recognition Vocabulary

Morphology is the study of words, rules governing word formations and relationships between words in a language. Structural analysis of words and parts of words as stems, root words, prefixes, and suffixes and functional analysis as parts-of-speech are investigated under morphology as well.

Morpheme is defined as the smallest grammatical unit, which is meaningful itself, in a language. Words are made out of morphemes. Morphemes that have meanings of their own are roots and other morphemes that have grammatical functions are considered as affixes.

Affixes can be of type prefix, infix or suffix according to the place they are attached in a word. Two categories of affixes are 1) *inflectional (grammatical)*: those that does not change the basic meaning of the word but its meaning regarding number, gender, person, tense, modality etc. 2) *derivational*: those that produce new lexical items from the word, which may not be of the same grammatical category anymore.

Morphotactics is the set of rules governing how morphemes are used to form words, i.e. how they are ordered in a word. The complexity of Turkish morphology and morphotactics makes it a challenging case for natural language processing [52]. All the affixes are suffix in Turkish. It has a highly productive inflectional and derivational morphology. New word formations are generated by adding suffixes to root words. Each morphological unit (morpheme) adds a definite meaning. A grammatical affixation can result in a new word which is practically not in use which still has a comprehensible meaning. Almost always suffixes are applied according to vowel and consonant harmonies which is due to correspondence between Turkish pronunciation and orthography.

It is possible to form hundreds of new words from a single root and a series of suffixes which leads to vocabulary explosion. Suffixes can change the part-of-speech class a word belongs to as well. A famous example from [45] depicts modifications in meaning of a word by addition of each individual suffix:

uygar +laş +tır +ama +dık +lar +ımız +dan +mış +sınız +casına
civilized +BEC +CAUS +NABL +PART +PL +P1Pl +ABL +PAST +2PL +AsIf

where Turkish word of *"uygarlaştıramadıklarımızdanmışsınızcasına"* can be translated into English as "(behaving) as if you are among those whom we could not civilize". Functions and meanings of morphemes are given as:

- BEC: "become"
- CAUS: the causative verb marker
- NABL: "not able"
- PART: past participle form
- P1PL: 1st person plural possessive agreement
- 2PL: 2nd person plural
- ABL: ablative (i.e. "from/among") case marker
- AsIf: derivational suffix to form an adverb from a finite verb

Some suffixes can be applied repeatedly which leads to a theoretically infinite lexicon. So, it is practically impossible to list all morphologically correct word forms. Even with large vocabularies the Out-of-Vocabulary (OOV) word rates are very high compared to other languages. Increasing the vocabulary size as a solution to OOV problem increases the computational and memory requirements of recognition systems. Higher perplexity in n-gram models is another problem resulting from large vocabulary size. A model with a higher perplexity has higher uncertainty and more confusion in prediction.

Chapter 5

Proposed Solutions to Turkish Online Handwriting Recognition Problems

The delayed strokes problem has a direct effect on optical model of the recognizer while insufficient vocabulary coverage due to high OOV rate affects the recognition performance through the decoding process. In this section, a number of solutions are proposed for these two problems. Evaluation results of the proposed approaches are given in Chapter 8.

5.1 A Solution for The Delayed Stroke Problem

A stroke is a pen trajectory starting with a pen-down point and ending with a pen-up point. It can thus be a full character, a part of a character or several characters written consecutively. When a stroke is separated from the character body it belongs to by one or more strokes, it is said to be 'delayed'. For instance the dot of an 'i' or the cross of a 't' can be delayed, when the dot or cross is not written immediately after the corresponding letter body.

Delayed strokes occur in multi-stroke characters, but not every multi-stroke character is written in delayed fashion. For instance, uppercase characters are typically written one character at a time, hence even multi-stroke letters (e.g. 'E') are not written with delay. In fact, each script has different strokes that are typically written in delayed fashion. These strokes can be either diacritical marks or integral parts of characters. Hence, the delayed stroke problem should ideally be examined for each language/script.

The first step of handling of delayed strokes is detecting such unorderly written strokes correctly. An exact delayed stroke detection can only be done *after* recognition, or more specifically after letter boundaries are known, by considering those letter parts that are written separately from the corresponding character bodies. For instance, the dot of an i is not considered delayed if it is written right after the letter body, even though it involves a pen-up movement with a backward move of the pen. Nonetheless, there have been various definitions, such as calling all backward moves after pen-up as delayed strokes, so as to detect and handle delayed strokes automatically during preprocessing.

Once such a working definition is at hand, the delayed strokes can be detected and then handled according to a chosen method, of which there are a few. In the remainder of the paper, the terms "definition", which is consistent with previous work, and "algorithm" are used interchangeably, to refer to the procedure used to describe/detect delayed strokes automatically. Figure 5.1 shows some examples of characters with diacritical marks as delayed strokes from the UNIPEN dataset.

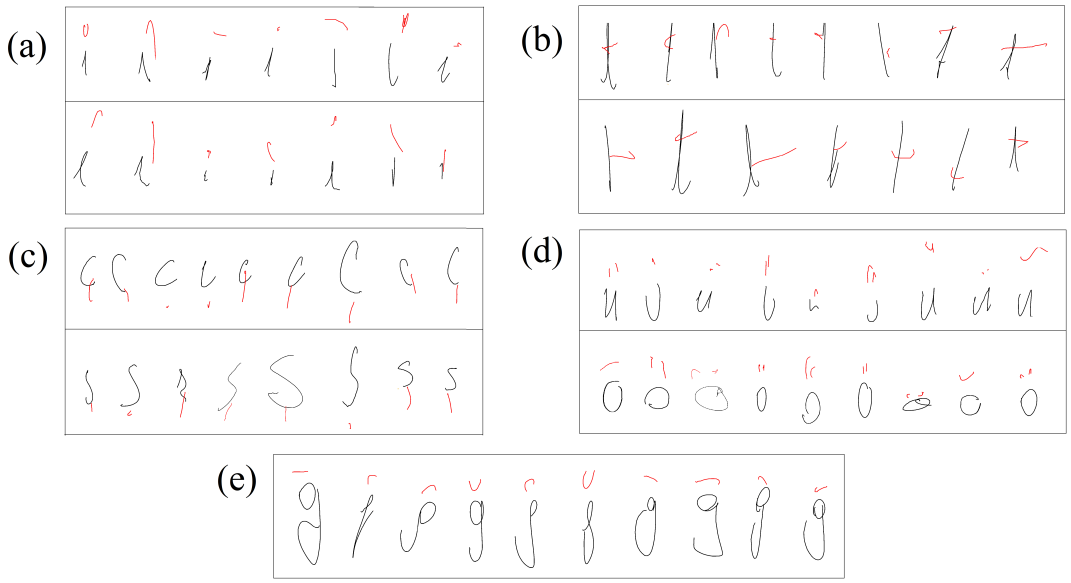

Figure 5.1: Samples of characters with potential delayed strokes: (a) 'i' with dot, (b) 't' with cross, (c) 'ç' and 'ş' with cedilla, (d) 'ü' and 'ö' with umlaut and (e) 'ğ' with breve.

In this work, the problem of delayed strokes are considered in two steps: 1) correct and comprehensive detection of delayed strokes, 2) application of the most appropriate method of handling on the detected strokes. In this work, the problem of delayed strokes are considered in two steps: 1) correct and comprehensive detection of delayed strokes, 2) application of the most appropriate method of handling on the detected strokes.

5.1.1 Existing Definitions

The definition given in the beginning of Section 1 ("strokes separated from the corresponding character body by other stroke(s)") is not very useful for *automatically* detecting delayed strokes. There are other definitions in literature for delayed strokes, proposed in the context of automatically detecting and handling them. For instance, [7] defines delayed strokes as:

> *. . . strokes such as the cross in 't' or 'x' and the dot in 'i' or 'j', which are sometimes drawn last in a handwritten word, separated in time sequence from the main body of the character.*

Another definition is given by [53] as:

> *. . . usually a short sequence written in the upper region of the writing pad, above already written parts of a word, and accompanied by a pen movement to the left.*

Finally, [2] identify delayed strokes as:

> *. . . those strokes that are written above already written parts, followed by a pen-movement to the left.*

A new practical definition which can be used for detection of delayed strokes directly is made in this work. It starts with the minimal definition based on a backwards movement, which expectedly marks too many strokes as delayed due to its very general/simple description:

> *. . . a new stroke starting with a backwards pen movement from the last pen-up point.*

Improving the minimal definition is possible through incorporation of script-specific features such as absolute and relative size and x-, and y-position of the stroke with threshold values learned from samples from the target script. Adding more constraints increases detection precision for the cost of increasing complexity of the definition.

In the next section, the minimal definition is expanded for English to obtain the proposed definition. The new definition is learned *automatically* from the handwriting statistics learned from the UNIPEN dataset. Specifically, a subset of 1,000 random words are marked manually for presence and type of delayed strokes: each sample is visually inspected at stroke level and the strokes that correspond to a dot or a cross of a character are marked, along with whether they are 'delayed' or '"regular'.

This 1,000-word training set contains a total of 5,124 strokes and a total of 816 dots and crosses that can be written in delayed fashion. Of these 816 strokes, 332 are delayed (225 i-dots and 107 t-crosses), while the rest (484) are not. Overall, the number of non-delayed strokes is 4792. Details of the UNIPEN dataset itself can be found in Section 6.1.

After generating the ground truth dataset, the decision tree learning algorithm is used to minimize the delayed stroke classification error, subject to some constraints regarding the tree size.

5.1.2 Proposed Definition for Delayed Strokes in English

The English script uses 26 letters from the Latin alphabet. Parts of letters and diacritical marks can be written in delayed fashion: dots for the letters 'i' and 'j', bar-like strokes (crosses) in 'f','t','z', and 'x', and diacritical marks in borrowed words. Delaying dot-type strokes is very common, followed by crosses, while diacritical marks like accent, umlaut and cedilla are used mostly in loan words like naïve, café and façade.

A delayed stroke definition for English is formulated by concentrating on dots and crosses, as they cover the overwhelming majority of delayed strokes in English. Indeed, all of the strokes that are delayed in the randomly selected 1,000-word training subset of UNIPEN subset are either i/j-dots or crosses.

The definition is learned *automatically* using the CART Decision Tree learning algorithm with 10-fold cross-validation on the 1,000-word dataset. First, each stroke of a word is described in terms of the following set of measurements which conveys information about the shape of the stroke itself and its position withing the global context of the word it belongs to:

- positions w.r.t. baseline, corpusline and midline: as percentage of sampling points lying above these lines
- height of bounding box/width of bounding box
- normalized height of bounding box : height/corpus_height
- normalized width of bounding box : width/corpus_height
- depth of the stroke: distance to the middle point from line connecting two ends
- normalized stroke length: stroke_length/corpus_height
- stroke curvature : angle between lines connecting ends to the middle point

As the data is highly unbalanced (332 delayed strokes vs. 4792 regular strokes), random subsampling is applied to regular strokes, so that the ratio of positive and negative examples is 1/4. Also, a higher cost (x2) is set for the misclassification of the delayed strokes (False Negatives). Class prior probabilities are empirically determined from class frequencies in the dataset. When the training is complete, the full tree is pruned to keep the number of rules small, to make the definition simple and for better generalization.

The resulting tree classifies a stroke in a given word as 'delayed' or 'regular' based on the features of that stroke. The rules of the tree can be extracted, yielding a working definition for automatic detection of delayed strokes. In Algorithm 1, the procedure for detecting the delayed strokes according to the new definition derived from the tree rules is presented.

The threshold for backward movement, which is the distance skipped backwards over the last written letter, is set to average character width. The number of characters is estimated using a heuristic method given in [54] while the baseline and corpus-line are calculated by regression through minima and maxima method as described in [2].

Based on the upper and lower regional characteristics of strokes, a discrimination for the type is also made, by simply considering whether there are points in the upper region of the detected delayed stroke. Those with points in the upper region are labeled as crosses, while others are considered as dots.

Input: **W**: A "word" (a set of strokes)
S: A stroke in **W**
Output: Return True if **S** is a delayed stroke and False otherwise
$\mathbf{W}_{end}$ = x-coordinate of the last pen-up before **S**
$\mathbf{S}_{beg}$ = minimum of the x-coordinates in **S**
height = normalized height of bounding box of **S**
$\mathbf{W}_{ch\_width}$ = average character width in **W**
$\mathbf{W}_{c\_line}$ = y-coordinate of the corpus line of **W**
$\mathbf{W}_{c\_height}$ = difference between y-coordinates of the corpus line and the base line of **W**

if $\boldsymbol{W}_{end}$-$\boldsymbol{S}_{beg} \geq \boldsymbol{W}_{ch\_width}$
AND *0.86% or more of points in* ***S*** *are above* $\boldsymbol{W}_{c\_line}$
AND height $<$ *1.45**$\boldsymbol{W}_{c\_height}$ **then**
  Return True;
else
  Return False;
end

Algorithm 1: Proposed definition for detecting delayed strokes (see above for definitions)

Detecting All Dots and Crosses

The new definition finds dots and crosses that are delayed, but any subsequent handling of delayed strokes can potentially increase variation in writing if *all* (delayed or not) dots and crosses are not handled in the same way. For instance with the approach of removing delayed strokes, some of the characters will be stripped of the delayed parts while their counterparts with *non-delayed* strokes are left intact.

In order to study this issue, a new definition is developed for detecting *all* dots and crosses –whether they are delayed or not–, using the same decision tree learning approach (without enforcing a backward movement constraint), and using the appropriate data (the 816 strokes corresponding to the *all* dots and crosses in the training dataset and randomly selected 3,000 strokes from the rest). The procedure for detecting the delayed strokes according to the new definition is shown in Algorithm 2.

Evaluation of the Proposed Definition

The performance of the three definitions (minimal, proposed and DetectAll) are given in Table 5.1. As can be seen here, the minimal definition based on the often cited backward movement has a very high false positive rate (25.1%), while the proposed definition has about 10.3% error rate (equal false positive and negative). Detecting all dot and cross-like strokes has a lower error rate; however as it is computed over all strokes (delayed or not), the absolute number of errors is almost twice as much as the proposed method (297 versus

Input: **W**: A ”word” (a set of strokes)
S: A stroke in **W**
Output: Return True if **S** is a delayed stroke and False otherwise
height = height of bounding box of **S**
depth = depth of **S**
above_C = percentage of points above corpusline in **S**
above_M = percentage of points above
(baseline + corpusline)/2 in **S**

if ***above_M*** $\geq$ *87%*
AND height $<$ *1.9*corpus height*
AND ((above_C $\geq$ *42%* ***AND depth*** $<$ *381)*
OR (above_C $<$ *42%* ***AND height*** $<$ *0.56*corpus height))* **then**
  Return True;
else
  Return False;
end

Algorithm 2: DetectAll: Definition for detecting all (delayed or not) dots and crosses

Table 5.1: Accuracies of definitions.

Definition	FN	FP
1 Minimal	7.5% (25/332)	25.1% (301/1200)
2 Proposed	10.2% (34/332)	10.3% (124/1200)
3 DetectAll	2.6% (21/816)	9.2% (276/3000)

158).

The type labeling is evaluated separately, following the proposed definition. For the 298 (=332-34) strokes that are correctly detected as delayed stroke, type identification is correct for 95.30%. On the other hand, out of the 34 missed delayed strokes (not recognized as delayed), about 40% are dot strokes and 60% are crosses.

5.1.3 Delayed Stroke Handling Alternatives

There have been a variety of alternatives to address the variations caused with delayed strokes, namely discarding (removing) delayed strokes altogether; embedding delayed strokes as if they are written after each corresponding character (also called reordering); and their variations. Each has its own advantages and shortcomings. Specifications of a language and its script should be considered carefully in order to chose a best-fitting solution.

Removal

In this approach, delayed strokes are identified and removed before the recognition phase [55, 56]. Here the assumption is that they are somewhat superfluous and may not be necessary for recognition. While this assumption may largely hold for English, many more words may become indistinguishable in the absence of dots and accents in certain languages, such as Turkish and Arabic.

Information about removed delayed strokes can be used in post-processing for lexicon reduction [55] and disambiguation of similar word bodies [57]. While useful, the shortcoming of these post-processing attempts is that recognition cannot make use of the information about delayed strokes. Also, there is a risk of removing correct words during lexicon reduction due to faulty delayed stroke detections.

Embedding

Another alternative to deal with delayed strokes is to embed them in the writing sequence, such that they are reordered to appear after the corresponding letter. This reordering normalizes the sequence to a canonical writing order, thus reduces or removes sequence variations and extensions. Embedding is illustrated in Figure 5.2.

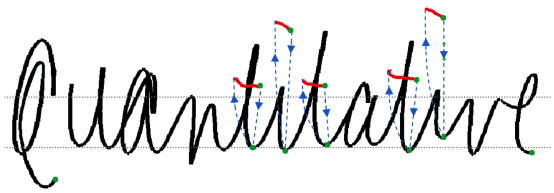

Figure 5.2: Detected delayed strokes (shown in red) are embedded as the arrows indicate.

Some handcrafted rules are used for deciding attachment points of delayed strokes in [58]. For Arabic, a vertical projection of detected delayed strokes is used in [59], while a more complex approach of segmenting the word body into strokes and sub-strokes to find correct projection locations of delayed strokes, is used in [60].

The main difficulty with this approach is to find the correct attachment points of delayed strokes, which can in turn adversely affect recognition. For English, the rules are designed for each type of delayed strokes separately. While there is some difficulty in deciding attachment points for i-dots, this is less of a problem for t-crosses.

In this study, for any stroke that is classified as an i-dot by some detection mechanism, first the x-coordinate of the local y-maximum point, $nearest\_max$, which lies closest to that stroke is found. Then the nearest y-minimum point, $nearest\_min$, that comes after $nearest\_max$ in the x-axis is decided. The i-dot is attached right after $nearest\_min$ by modification of the $nearest\_min$ as a pen-up point.

As for the t-crosses, the decision on the attachment point is based on the observation that crosses are in relation with letter bodies most of the time. If there are any pen-up points lying under a t-cross stroke and within its x-coordinate range, the cross stroke

is attached right after the one with the greatest x-coordinate. Otherwise, a sequential search for a convenient attachment position is made starting from the point with greatest y-coordinate lying under the t-cross. The search stops when the next point in the sequence has greater y-coordinate than the previous one. The cross stroke is inserted to its new position afterwards.

Hat feature

In this approach, the delayed strokes are *removed* but the existing feature vector is expanded with a binary feature, to indicate the location of the removed strokes [53, 54, 2]. The binary feature takes on the value of 1 at locations that were under a removed delayed stroke, and zero otherwise (see Figure 5.3).

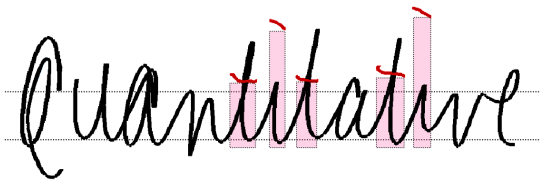

Figure 5.3: Hat-feature value is 1 for points lying below the delayed strokes (in the pink rectangular area) and 0 for the rest.

An alternative is to keep the original input and *add* the hat feature to obtain an extended feature set. In this case, the hat feature serves to highlight the presence of a delayed stroke event.

Delayed strokes as alphabetical symbols

In this approach, delayed strokes are considered as special characters in the alphabet [61, 7, 23]. Words which contain characters that can be written in delayed fashion are represented with all the alternative forms (corresponding to the possible writing orders) in the lexicon. For instance, if the i-dot is represented with a '.' and a cross is represented by a '-', then, the word '*it*' has three possible spellings in the dictionary: {i.t-, it.-, it-.}. While this seems like the best approach in terms of not losing information or interfering with the writing order, it is not suitable for mid-to-large scale vocabulary tasks, as the hypotheses space can grow dramatically.

Adding at the end

Another method for handling delayed strokes is moving them to the end of the word such that they are appended to the last sampling point of the word, in the order of appearance in the writing direction [62]. The main issue with this method is that it is not suitable for sub-word based recognition systems, such as character HMMs, as it separates delayed strokes from corresponding letter bodies with respect to time.

Removal and Embedding Hybrid

A hybrid method of removing the i-dots and embedding the t-crosses is another alternative. This hybrid approach was due to the observation that deciding on the correct position of a dot and embedding it is difficult, as dots are not constrained to be in a precise position with respect to the corresponding letter body, while crosses are. Furthermore, while rare, some writers omit writing dots altogether in certain languages, so removing all i-dots may be beneficial, as it would make the dotted and not dotted versions of the words match.

5.1.4 Proposed Handling Method for Delayed strokes

An effective handling method is employed to remove unnecessary variation effect of the delayed strokes, once they are detected using the proposed definitions. In order to find the most effective one for a particular dataset-recognizer pair, a series of experiments is done using the proposed definitions. Results are presented in Chapter 8 to be consistent with the layout of the thesis. For the sake of completeness in the section, according to these results, removal of the delayed strokes using the proposed definition in Algorithm 1 performs best for all the datasets used in this research.

5.2 A Solution to The High OOV Rate in The Recognition Dictionary

Turkish lexicons have exponential growth rate because of its highly productive nature. This phenomenon is observed as high OOV rates in the dictionaries of automatic recognition systems.

In order to deal with the high growth rate of vocabulary in languages with complex morphology as Turkish, vocabularies based on units smaller than full-words has been proposed and applied for language modeling in speech recognition domain recently[63, 64, 65, 66, 67]. These *sub-lexical* units can be obtained by splitting words statistically or grammatically.

The unsupervised statistical approach of splitting words into sub-lexical units, *or morphs*, can be defined as finding likely segmentation points in an unannotated large training corpus when trying to iteratively optimize a given function [68, 67]. A usual optimization function is Minimum Description Length (MDL) by which minimum size vocabulary with the lowest number of distinct morphs is obtained. The advantage of using a statistical approach is its independence of any language-specific linguistic processing and hand-crafted segmentation rules. Also, it can process all words and does not suffer from the OOV problem, with inclusion of the letters as sub-lexical units.

Words can be decomposed into several sub-lexical units according to the grammar of a language. Syllables, morphemes, stem-ending pairs and even letters can be grammatical sub-lexical units. A recognition vocabulary can be built with combinations of these unit types.

Morphemes are obtained by morphological analysis which is the rules-based process of decomposition of words into constituent grammatical units. Similarly, the process of stemming, which is reducing inflected and derived words to their word stem, can be used to yield stems and endings (grouping of suffixes).

Using a grammatical approach to obtain sub-lexical units has the advantage of preventing grammatically invalid productions via language information. A shortcoming of the grammatical approach based on morphological analysis is the use of a root list which may not be complete. Then, some of the words can not be segmented into morphemes which makes them OOV words.

Each sub-lexical unit type brings its own advantages and shortcomings in terms of dictionary size, recognition success and computational costs. Word-only networks improve recognition performance at the cost of very large vocabulary size and OOV problem. When morphemes and stems/roots are chosen as grammar units, morphotactics and orthographic rules should be integrated into the network (see Section label:Stems and Morphemes as recognition units). Also, recognition performance tend to be lower due to shorter units in the network. Stem-ending pairs combine the advantages of word-based and morpheme-based approaches (see Section sssec:stemEnding). Endings are group of suffixes that are longer units compared to morphemes and size of recognition vocabulary is strictly smaller than that of word-based solutions. Some phonetic and orthographic rules like vowel harmony can be enforced in stem-ending networks. Syllable-based solution is the simplest one yet it suffers from lower recognition performance due to shorter units. In addition, its recognition output can not be post-processed to correct substantial errors in suffix sequences as in the case of stem-ending or morpheme based systems, since the separation of the stem and suffix parts is not possible within this solution.

Coverage of a lexicon is a measure to tell what proportion of the words in the test set are covered by the recognition lexicon. There is a tradeoff between lexicon coverage and the recognition accuracy since increasing the lexicon size also increases the number of alternatives in decoding. A related term is Out-of-Vocabulary (OOV) words that refers to words not covered by the lexicon. Coverage and OOV rates are used for evaluating fitness of lexicons.

In this work, the grammatical approach is taken and several grammatical units are used to generate alternative recognition vocabularies. Each vocabulary type is evaluated based on its coverage on the training text corpus.

5.2.1 Text Corpus Preparation

The BOUN corpus which is introduced in Section 6.1.2 is automatically generated from mostly web resources so it requires some cleaning and formatting to be useable in the proposed vocabulary generation methods. It is preprocessed before extracting grammatical words, stems, endings and morphemes.

As a first step, all entities containing non-alphabetic characters (punctuation, numerals, monetary sings etc.) are removed to obtain 408M words. Both upper and lower case letters are included as for handwriting task letter case information is necessary most of the times. Secondly, the words are analyzed morphologically to extract stems and suf-

fixes. Words are decomposed into stems and endings (i.e. suffix groups) usually by an open-source morphological analyzer for Turkish, the TRmorph [69], which is based on finite-state transducer technology. 335M words are successfully analyzed by extraction of at least one stem and they will be used for language modeling and vocabulary building. Some 73M words which can not be analyzed include proper names, foreign words and names, misspellings and strings of characters which do not constitute grammatical words and finally words erroneously concatenated together. Extracted units are utilized individually or in combinations for generating several recognition lexicons and language models.

Table 5.2 summarizes the number of grammatical units extracted by morphological analysis. It is observed that average number of stems extracted per word is about 4.5. About 20% of the words are left undecomposed during analysis for they are either in root form or proper names. When undecomposed words are excluded, average stem per word becomes 5.7.

Table 5.2: Grammatical units obtined from BOUN corpus

Type	Unique	Total
Words	1,578,553	334,758,204
Stems	69,013	1,528,104,572
Endings	138,226	1,146,201,689
Morphemes	864	2,414,760,215

5.2.2 Stems and Endings As Recognition Units

It is a known issue that morphological analysis yields multiple decompositions for some of the words. Sometimes, it is due to stems generated from the same root by derivation. For example,the decomposition of word *gözlükçü* (optician) whose root is 'göz' yields three stems including the root :

göz+lük+çü: göz ⟨N⟩⟨lHk⟩⟨N⟩⟨cH⟩⟨N⟩

gözlük + çü: gözlük ⟨N⟩⟨cH⟩⟨N⟩

gözlükçü: gözlükçü ⟨N⟩

Here, ⟨N⟩ denotes a nominal form and H is the set of vowels {'ı', 'i', 'u', 'ü'}.

In other cases, there are stem(s) and root(s) that share the same spelling. For instance, word *adaya* may have one of these three roots; one verbal root '*ada*' (to devote) and two nominal roots '*ada*' (island) and '*aday*' (nominee):

ada+ya: ada⟨N⟩⟨yA⟩⟨N⟩ (to the island)

ada+ya: ada⟨V⟩⟨yA⟩⟨V⟩ (may he devote -optative-)

aday+a: aday⟨N⟩⟨yA⟩⟨N⟩ (to the nominee)

Again, ⟨N⟩ and ⟨V⟩ denote nominal and verbal forms respectively and A is the set of vowels {'a', 'e'}.

For this research, all the different stems and endings are kept in vocabulary in case of multiple decompositions. Although some of the vocabulary items get their frequency of occurrence artificially increased, coverage of the vocabulary increases as well with this approach. Also, data sparseness problem of language modeling can be alleviated moderately. As stated in [70], the size of the current corpus does not cover all language usage and should be extended accordingly. A possible negative side-effect is a decrease in HMM recognition performance due to shorter vocabulary units which can be compensated by use of Word Insertion Penalty(WIP) (see Section 7.2.1) to some extent.

Vowel harmony is applied during suffix affixation, such that vowels of suffixes change to comply with the vowel harmony rules in Turkish. For example, for vowel group H={'i', 'ı', 'u', 'ü'}, the verbal suffix dH is realized as -di in "gel+di", -dı in "al+dı", -dü in "gör+dü"' and -du in "oku+du" depending on the last vowel of the stem they are attached to.

One solution to have affixation complying the vowel harmony is to group stems according to their last vowels and then specifying which suffixes are attached to which group of stems as it is done in [71]. In this work, a post-processing step is employed for correcting vowel harmony discrepancies after recognition. This approach allows somewhat a more flexible decoding especially when it is considered that only a small portion of the all possible endings can be included in the language model. If one form of realization is absent from the model (for example dü) while another one which is functionally the same but different in realization is included, the system can still generate the correct ending after post-processing as in "gel+dü" → "gel+di".

5.2.3 Stems and Morphemes as Recognition Units

Suffixes can be used as recognition units individually instead of being packed into suffix groups as in the case of stem-endings solution. As already explained previously, Turkish word formations are generated by adding suffixes to root words. Each one of these suffixes is a morphological unit (morpheme) that adds a definite meaning. Finite State Automata (FSA) formalism provides a suitable representation for morphotactics, i.e. the set of rules governing how morphemes ordered in a word in Turkish. The morphemes in a word can be obtained by a morphological analysis.

The FSA definitions in [72] is considered as a base case and they are extended with addition of more inflectional suffixes since the basic machine does not include them all. Extended Backus-Naur form (EBNF) grammar formalism is used for representation of the FSA, which can be accessed from [1]. The extended FSA is validated on the endings extracted from the text corpora. For this purpose, a lexer and a parser are generated semi-automatically by *The BNF Converter* which is a compiler construction tool generating a

[1] https://tinyurl.com/y8qlq48m

compiler front-end from a Labelled BNF (LBNF) grammar [73]. It is provided as a free software under GNU General Public License by Chalmers University of Technology and University of Gothenburg. A Flex lexer and a Bison parser are generated from the FSA grammar.

Flex-generated lexers scan through the input to find a matching pattern for lexical tokens that are given as regular expressions. Sometimes multiple patterns can match the same input. Such ambiguities are solved by matching the longest possible string every time the scanner matches input, by default. However, this feature prevents parsing the grammar accurately. For example, when the lexer scans "köşe+sinde" (*at its corner*), whose suffixes should be tokenized as ⟨si⟩⟨nde⟩, it outputs the suffix part as ⟨sin⟩⟨de⟩. Although ⟨si⟩⟨nde⟩ are both valid lexical tokens, that ordering of them is grammatically incorrect. When the parser receives the set of incorrect tokens, it fails to parse as expected. In order to avoid shorter but correct tokens being discarded in favor of longer ones, inputs to the lexer are given one character at a time which solves the problem.

Another source of ambiguity that affects the validation process of the FSAs is multiple alternative derivations for a given input. To show on a simple example, given the grammar:

$$A \rightarrow A + A | A \times A | A | x$$

and the input $x + x \times x$, there are two possible derivations The first one is:

$$\begin{aligned} A &\rightarrow x + A \\ A &\rightarrow x + A \times A \\ A &\rightarrow x + x \times A \\ A &\rightarrow x + x \times x \end{aligned}$$

with x parse tree as in the Figure 5.4
and the second one is:

$$\begin{aligned} A &\rightarrow A \times A \\ A &\rightarrow A + A \times A \\ A &\rightarrow x + A \times A \\ A &\rightarrow x + x \times A \\ A &\rightarrow x + x \times x \end{aligned}$$

with x parse tree as in the Figure 5.5

A standard solution for such ambiguities is Generalized LR parsers which is an extension of an LR (left-to-right, rightmost) parser algorithm to handle non-deterministic

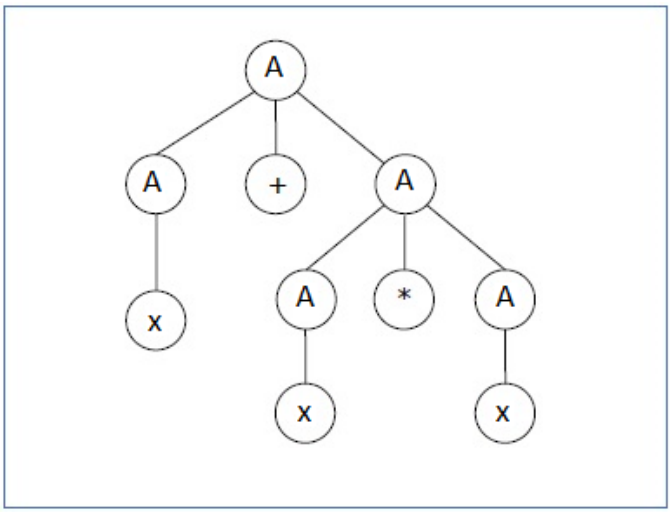

Figure 5.4: The first derivation for the input $x + x \times x$

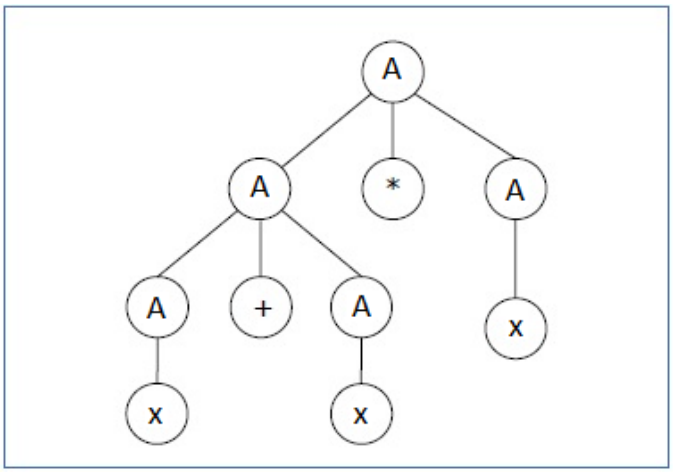

Figure 5.5: The second derivation for the input $x + x \times x$

and ambiguous grammars [74]. The GLR algorithm works in a manner similar to the LR parser algorithm but it allows multiple transitions in case of conflicts in the action to be taken. When a conflicting transition occurs, the parser process goes parallel and the GLR parser processes all possible interpretations of a given input in a breadth-first search.

This work use the Bison parser generator [75] to generate a GLR parser to handle the ambiguity in the grammar rules when there are more than one derivation tree for a given input. Hence, all possible parses of a given ending is generated so to validate it against the designed FSA grammar.

5.2.4 Vocabulary Size Determination

It is crucially important to select an appropriate size for the recognition vocabulary as it has direct effects on the recognition performance. Coverage of train and test sets by

the chosen vocabulary and OOV rates are useful in deciding vocabulary size. Also, the criteria for selection of words in the vocabulary is equally important.

In this work, the frequency of occurrence is utilized for selection of vocabulary items. Vocabulary sizes are decided by setting a fixed OOV rate threshold on the training corpus. Table 5.3 gives details of coverages on lexicons of BOUN corpus itself and the test dataset used for the recognizer. In order to keep vocabulary size manageable, a limit of maximum 5% OOV rate on the BOUN corpus is set as threshold which is met first with the most frequent 130K words. Test set coverage rate is much smaller, and the theoretical maximum accuracy with the chosen lexicon is 75% for the proposed recognizer. Actually, even the whole corpus can not cover more than 87.5% of out test vocabulary. The OOV words that are not covered by the 130K-word vocabulary is given in Appendix A.

Table 5.3: Coverage rates of word-based vocabularies

Frequency of occurrence	Size	Corpus coverage	Test coverage
most frequent 3%	50K	90.0%	62.6%
most frequent 5%	80K	93.2%	69.6%
most frequent 7.5%	120K	94.4%	72.3%
most frequent 8.2%	130K	95.7%	**75.6%**
most frequent 9.5%	150K	96.3%	77.6%
most frequent 100%	1,57M	100.0%	87.5%

Table 5.4: Coverage rates for stem-ending-based vocabularies

Unit	Frequency of occurrence	Size	Corpus coverage	Test coverage
Stem	most frequent 10%	7K	96.6%	82.0%
Ending	most frequent 5%	7K	97.9%	90.7%
Stem-ending			95.3%	83.9%
Stem	most frequent 7%	5K	94.5%	76.7%
Ending	most frequent 5%	7K	97.9%	90.7%
Stem-ending			94.0%	82.1%
Stem	most frequent 10%	7K	96.6%	82.0%
Ending	most frequent 4%	5.5K	97.24%	87.3%
Stem-ending			**95.0**%	**83.6**%
Stem	most frequent 10%	7K	96.6%	82.0%
Ending	most frequent 3%	4.1K	96.1%	82.4%
Stem-ending			94.6%	82.7%

In the same manner, a suitable vocabulary size is decided for stem-endings and stem-morphemes solutions for vocabulary building. If a stem is missing from a stem-ending vocabulary, then any word which has that particular stem is not covered as well. Similarly, a missing ending translates to recognition errors for all words containing that ending.

All possible decompositions of a word are contained in the stem and ending lists. A caveat in this calculation is when a word has multiple decompositions. In this case a word is accepted as covered if it has at least one decomposition of which stem and ending are covered by stem and ending vocabularies simultaneously. Continuing with the *gözlükçü* (optician) example in Section 5.2.2, there are two stem-ending pairs i.e. {göz+lükçü, gözlük+çü} and one stem with no ending i.e {gözlükçü} with morphological decomposition. It is sufficient to contain one of these three analysis in the stem-ending vocabulary to have the word gözlükçü covered by the vocabulary.

In order to calculate coverage rate of stem-ending vocabularies a first is to generate all possible combinations of stems and endings (including empty endings) and then to check existence of each word in the corpus and the test set, within the combinations. Various coverage rates are given in Table 5.4. Stem and ending list are tested on individually on stem and ending lists of the corpus and the test set. Coverage of a stem-list-ending-list pair is calculated on words of the corpus and the test set.

In order to make a direct comparison between word-based and stem-ending-based vocabularies, a pair of suitable top-n percentages is searched for choosing stems and endings such that OOV rate would not be greater than 5% as with the word-based system. It is observed that increasing the number of stems has higher boosting effect on coverage than increasing number of endings (see Table 5.4. This is justifiable as many of the stems are made of a root and derivational suffixes. Top 10% of stems and 4% of endings are chosen according to frequency of occurrence to achieve enough coverage with the smallest vocabulary size in the stem-endings design.

As for the stem-morpheme based vocabulary, the morpheme FSA is evaluated on endings(suffix groups) as described in Section 5.2.3. With its 342 unique morphemes, the FSA covers over 95% of all endings occurring in BOUN corpus and 100% of the test set. Uncovered suffix groups include those that are misspelled (ex. -abilecekleri $\rightarrow$ -abileceleri, -mişcesine $\rightarrow$ -mişçesine), ungrammatical groups (ex. -ıyorkendi, -ymuştu, -tıydı), concatenation of -mı/-mi/-mu/-mü and -da/-de (ex. -ıyorlarmıymış, -masınada). Aiming at a directly comparable system, the same stem list with the stem-ending solution is used in the stem-morpheme based vocabulary.

Chapter 6

Software and Resources

This chapter introduces several datasets and software that are used in realization of the proposed recognition system. Datasets are used for training and testing of the optical model and the language models of the recognizer. Software are used in both processing the datasets and training and testing the system modules.

6.1 Datasets

6.1.1 Datasets for Optical Model Training

Optical model of HMM recognizers are trained on a set samples by Baum-Welch algorithm (see Chapter 2). Large amount of train data is required for training of state-of-the-art recognizers. This work utilizes two online handwriting dataset which will be detailed subsequently.

UNIPEN

There are a few standard online handwriting datasets (IRONOFF [76], UNIPEN [77], IAMonDB [78]) that have been widely used in handwriting recognition research. The isolated word collection of the UNIPEN dataset is used for this work, as it is the largest publicly available collection with a large vocabulary. The UNIPEN online handwriting database is a collection of handwritten samples (containing sets of characters, digits, words and texts), collected by a consortium of 40 companies and institutes over time. The whole collection consists of a total of 5 million characters by writers from all around the world, representing a wide variety of writing styles. UNIPEN is considered to be a difficult dataset since it incorporates samples from diverse number of writers from very different background. The number of words contributed by different writers is highly unbalanced, ranging from less than 5 to more than 500. In fact, more than 2,000 writers submitted very small number of words, specifically between 1 and 5, whereas there are 59 writers who contributed more than 400 words. Table 6.1 shows details of the words-per-writer distribution.

This work uses the isolated word collection (category 6) of the current publicly available version called $train\_r01\_v07$ training dataset. This collection contains 75,529 cursive or mixed-style words in total. The lexicon size is 13,913 words, with separate upper and lowercase versions for some of the words. Figure 6.1 shows a selection of samples from that dataset.

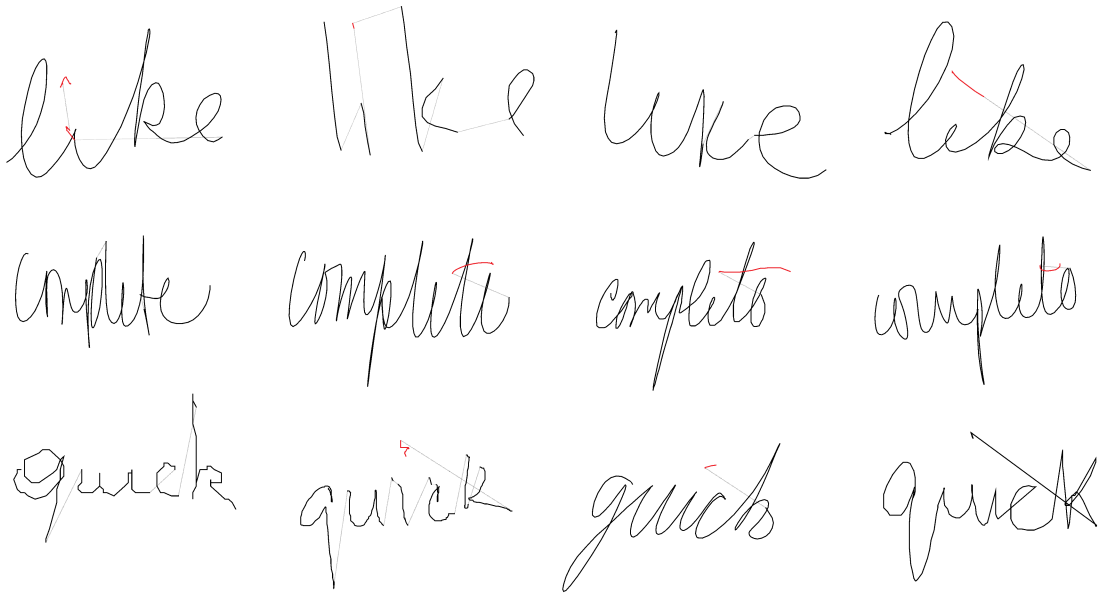

Figure 6.1: Sample handwritten words from the UNIPEN dataset. Strokes that are written separately from character body are shown in red.

Table 6.1: UNIPEN word contribution per writer distribution

Number of words per writer	Number of writers
1-5	2,616
6-99	444
100-199	131
200-399	46
400-523	59

There is no common standard for how to split the UNIPEN data as training, test and development/validation sets. There is a development release called devtest_r01_v02 used in some works [7], but it is not publicly available. Hence, different works use different splits and thus report results on different portions of the UNIPEN dataset, rendering them not directly comparable.

In order to have reproducible and comparable results, two different split scheme is designed and used in this work. The first one is designed to generate training, testing

and validation subsets from $train\_r01\_v07$-category 6. The second one is for using the dataset in cross-validation type experiments.

UNIPEN Split for General Use

The UNIPEN dataset is split into train, validation and test sets considering some of the issues like writer-independence, writer distribution by contribution and size.

In order to reflect the original unbalanced writer-contribution distribution of the whole UNIPEN, the split is done such that each of the three sets include some of the writers in each of the five categories of Table 6.1.

Furthermore, a writer is included in only one of the subsets, so the final division can be used for writer-independent recognition tasks. Another issue to consider is to make validation and test subsets have a similar vocabulary size. The validation set is further divided into two sets each such that one of the validation sets, Validation1, is used for validation in training of the recognizer while the other one is left over for further use in the future such as language model optimization. Similarly, the test set is also divided into two subsets, to enable testing with different size lexicons. Table 6.2 shows the figures related to the subsets.

Table 6.2: UNIPEN data set split

Subset	Num. writers	Num. samples	Lexicon size
Train	2,005	50,027	10,980
Test	650	11,483	5,500
Test_1	366	3,542	2,000
Test_2	545	7,941	3,500
Validation	643	14,019	5,945
Validation_1	322	6,999	4,026
Validation_2	321	7,020	3,550

UNIPEN Split for Cross-Validation Experiments

In the second splitting scheme, the data is divided randomly into 10 random but similar subsets for 10-fold cross validation experiments and the split is made publicly accessible[1]. Specifically, the subsets are roughly equal in number of writers (~ 320), number of samples ($\sim 7,000$) and size of lexicon (3,500). The split also takes the skewed contribution per writer distribution of the UNIPEN (i.e. some with 1 sample and some with more than 400) into account and maintains similar distributions in each of the subsets. No writer appears in more than one subset, so the evaluation is writer-independent. Lexicons of the subsets are not the same but 1,000-words were found to be shared between 5 of the subsets.

[1] https://tinyurl.com/y7m3rv5w

The ElementaryTurkish dataset

There is no Turkish online/offline handwriting dataset which is publicly available. As a part of this research, a collection of online isolated word samples are generated using Android Tablet PCs. A special software is designed and implemented for this purpose. There is no constraint enforced on the writing style, baseline compliance or writing area. Volunteers, mainly adults, which come from different backgrounds contributed by writing around 100 words selected from elementary Turkish text books on average. That dataset,which will be referred as the ElementaryTurkish dataset hereafter, is made publicly available [2]. Figure 6.2 shows a selection of samples from that dataset.

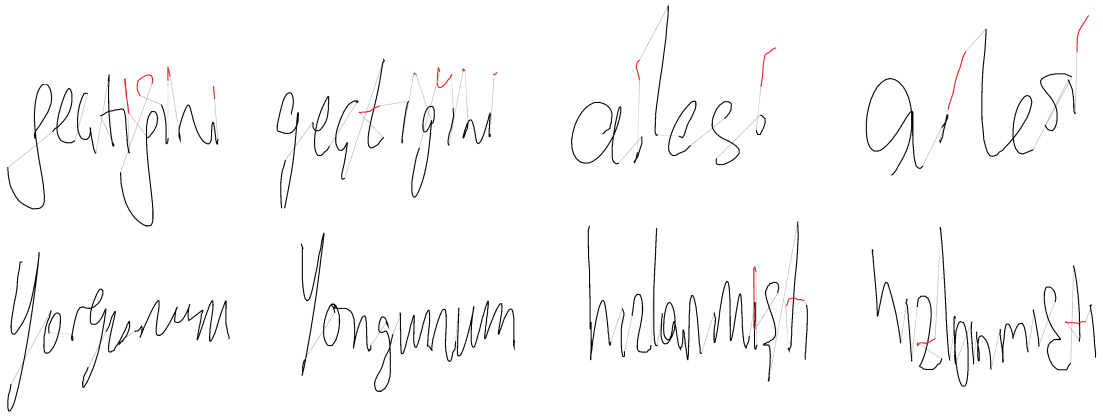

Figure 6.2: Sample handwritten words from the ElementaryTurkish dataset. Strokes that are written separately from character body are shown in red.

The ElementaryTurkish dataset contains around 10K samples of isolated words from the 2,089-word lexicons of 1st and 2nd Grade Turkish books, written by 113 different writers including children.

The train set includes 7,360 samples from a 1956-word lexicon by 79 writers and the test set contains 2,500 samples from a 2089-word lexicon written by 34 writers in the test set. In this work, the test set is further divided into two parts to spare a validation set as well. With this split, 804 samples with a lexicon of 800 unique words is used for testing. That dataset split is designed such that the writers are not overlapping. The test set lexicon is covered by the train set lexicon.

6.1.2 Language Modeling Corpus

Large corpora is essential for statistical natural language applications. However, there are quite a few publicly available Turkish text corpora which are mostly limited in size and coverage. One recent contribution on large corpus building in Turkish is BOUN Web corpus which has been used in different studies before [70, 63, 79]. In this work, the

[2] https://tinyurl.com/yc93rcf5

Table 6.3: BOUN Corpus details

Corpus	Words	Tokens	Types
NewsCor	184M	212M	2.2M
Milliyet	59M	68M	1.1M
Ntvmsbnc	75M	86M	1.2M
Radikal	50M	58M	1.0M
GenCor	239M	279M	3.0M
BOUN Corpus	423M	491M	4.1M

BOUN corpus is employed for building statistical language models as well as several recognition dictionaries.

BOUN-Web text corpus

The BOUN Corpus is composed of four sub corpora, that are all collected from web sites by means of a web crawler script [70]. Three of the sub corpora are derived from websites of three major newspapers in Turkish and they are referred as BOUN NewsCor.The other sub corpus is from a sampling of Turkish web sites and named as BOUN GenCor. The BOUN Corpus is encoded in paragraph and sentence level in XML Corpus Encoding Standard (XCES). Statistics about the BOUN Corpus can be found in Table 6.3.

6.2 Software

6.2.1 TR-Morph

TRmorph is an open-source morphological analyzer for Turkish based on finite-state transducer technology [69]. It has tools for morphological segmentation, stemming and lemmatization, guessing unknown words, grapheme to phoneme conversion, syllabification/hyphenation and morphological disambiguation. It vocabulary is built semi-automatically from 800 million tokens collected from the Web.

In this work, morphological segmentation tool is used for decomposing words into a stem and a set of suffixes.

6.2.2 SRILM

SRILM is a well-known and widely used toolkit for building and applying statistical language models [80]. It can be used free of charge. The research community has been using the toolkit in a variety of application domains including speech recognition, machine translation, tagging and segmentation and handwriting recognition. SRILM has compatibility with HTK lattice files format(i.e. SLF). Tools that will be used in this work are:

- lattice-tool : For lattice operations including size reduction, pruning, null-node removal, weight assignment from language models, lattice word error computation, and decoding of the best hypotheses.
- ngram-count : For generating and manipulating N-gram counts, and estimation of N-gram language models from the counts.

6.2.3 HTK

The Hidden Markov Model Toolkit (HTK) is one of the most popular software for building Hidden Markov Model systems [81]. It is a portable toolkit for building and manipulating Hidden Markov Models. Although initially designed for speech recognition research, HTK has been used for numerous other applications including research in speech synthesis, character recognition and DNA sequencing domains.

A list of the tools that are used in this study grouped by their functions is:

- Data manipulation :HCopy, HLEd, HHEd
- Training : HCompV, HInit, HERest
- Recognition : HLRescore, HParse, HVite, HResults

Chapter 7

Turkish Online Handwriting Recognition System

This chapter contains the implementation details of the recognition system. It describes stages of building the recognizer with particular design choices at each step. Evaluation of the resulting recognizer can be found in Chapter 8.

There are two distinct processes in building the recognizer. The first one is to decide on some design parameters regarding the architecture of the recognizer and to make necessary adjustment to reach the most appropriate system configuration possible. The second one is to implement a final system using the optimized parameters from the first process.

The dataset split described in Section 6.1.1 is used in the first process of deciding the system configuration. After the parameter optimization, the final system is trained with the whole UNIPEN set and the training set of ElementaryTurkish (ET) dataset.

7.1 Optical Modeling

The optical model is created by taking a large database of handwriting and using special training algorithms given in Chapter 2 to create statistical representations for characters in the language. These statistical representations are called Hidden Markov Models (HMMs) and each character has its own HMM.

An HMM system has a number of parameters that should be decided before it is trained. Topology type, number of states, number of mixtures in case of continuous density models and means and covariances of densities are the parameters to consider (see Section 2.1.3 for details). This section lists the chosen parameter values of the proposed recognition system.

7.1.1 System Architecture

A left-to-right HMM topology, where allowed transition for a given state are to itself and to the next one, is chosen for all the characters. All character HMMs have the same

number of states as 20 which is decided empirically. In total, 64 character models are built: 26 characters in the English alphabet which include 23 characters common to the Turkish alphabet and six additional Turkish characters (i.e 6 characters (¸ç, ğ, ı, ö , ş and ü)) in uppercase and lowercase forms. Since the number of English words in the dataset is much higher than that of the Turkish words, and the test data consists of only Turkish words, learning the characters out of context is a more suitable approach. Because of this reason, all of the models are context-independent.

In the emitting states, the observation probability distributions are estimated by 35 mixtures of Gaussian components that have diagonal covariance matrices. The number of Gaussians is decided using an iterative approach as suggested in [36]. Starting from an initial model, with 3 mixtures, the number of mixture components are increased by one via splitting the Gaussian distribution with the highest weight until no further improvement was obtained on a validation set. The mean vectors of resulting Gaussian distributions are defined as the mean of the original one perturbed by plus or minus 0.2 standard deviations. After each split the whole system of models is trained for a particular number of iterations until the next split. For the case of this work, four iterations of training is applied before the next splitting.

7.1.2 Preprocessing

For scale variations a size normalization procedure is applied in the form of rescaling the height of the writing to 1000 pixels, while the width is scaled accordingly keeping the aspect ratio. Skew normalization is simply done by correcting the baseline angle and it is found to be useful in increasing the recognition accuracy. For slant normalization, histogram method of [21] is preferred. The baseline and corpus-line are calculated by *regression through minima and maxima* method as described in [2].

Equidistant re-sampling is applied by linear interpolation on pen trajectory. The number of sample points is set proportional to the number of characters within the word. Re-sampling frequency is decided empirically to be 50 points per character. The number of characters is known a priori for the training set, from the known labels of the input words. As for the test set, a heuristic method for estimating the number of characters in the input word as described in [54] is employed. With this method, the number of lines crossing the mid-line (horizontal line between the baseline and corpus-line) is used to estimate the number of characters in the word.

Delayed strokes introduce an extra source of variation so they must be handles properly before proceeding to the training phase. According to the explanations in Chapter 5 on delayed strokes handling and the results presented in Chapter 8, unorderly written strokes are removed as a final step in preprocessing before feature extraction. The strokes are detected by using the proposed definition of Section 5.1.2.

7.1.3 Features

Different handcrafted features are used in literature for the representation of online handwriting. Starting from a wider set of features that are commonly used [7, 54, 82, 2, 53,

62, 60], a subset of 8 features are selected through extensive experiments, according to word recognition performance of the system. Then, considering the errors made by the system, a new feature (distance to median y-value) is designed within this work. The new feature adds some global context information to the feature vector, by indicating the vertical position of the frame with respect to the median y-values in the whole sequence. This information is relatively robust if there isn't a heavy slope on the baseline and helped improve the overall recognition accuracy. The nine features are:

- *delta*: differences from the x- and y-coordinates of the previous point;
- *sine and cosine of angle* between x-axis and the line joining consecutive points;
- *curvature angle*: the angle between the lines to the previous and the next point;
- *vicinity linearity*: average squared distance of each point in the vicinity to the straight line from the first to the last vicinity point;
- *vicinity slope*: a pair of features such that cosine and sine of the angle of the straight line from the first to the last vicinity point;
- *pen-up/down*: a binary feature showing whether a sampling point is an up point (pen is lifted up here) or a down point (pen is touching the writing pad);
- *normalized x*: the x-position taken after high-pass filtering, i.e. after subtracting a moving average of 5 previous points' x-values from the real horizontal position.
- *distance to median y-value*: distance to the median y-value of the given sample point sequence.

7.1.4 Training

Parameters of each HMM model are initialized by HCompV tool in HTK so that all component means and all covariances are set equal to the global data mean and covariance. HERest tool is used to train the models with the Baum-Welch algorithm. The models are trained iteratively and the performance is assessed on the validation set every 10 iterations and the training is completed when no more improvement is observed.

7.2 Language Modeling

This section describes the details about how the language modeling is implemented with the HTK and the SRILM tools that are introduced in Section 6.2.

A word network is a graphical representation of valid, grammatical expressions in a recognition system. It enforces the linguistic constraints during the decoding phase. As an example, if the task is sentence recognition and the recognition vocabulary contains words, the network shows how the words can be ordered to generate sentences in terms of the task grammar.

In the HTK environment, word networks are defined in Standard Lattice Format (SLF) which consists of a list of nodes and a list of arcs. The nodes represent words and the arcs represent the transition between words. Special null nodes are used for reducing the number of arcs required. HTK provides a tool, *HParse*, which converts recognition grammars written with a notation based on extended Backus-Naur Form (EBNF) to word networks. N-gram language models can be integrated to word networks as explained in Section 3.3. Figure 7.1. shows a simple grammar and its representation with SLF and word network. There are four valid sentences in this grammar : BEN OKULA GELDIM (*I came to the school*), BEN EVE GELDIM (*I came home*) , BEN OKULA GITTIM (*I went to the school*), BEN EVE GITTIM (*I went home*) . In that case, these four expressions are the only sentences that can be output by the corresponding recognizer.

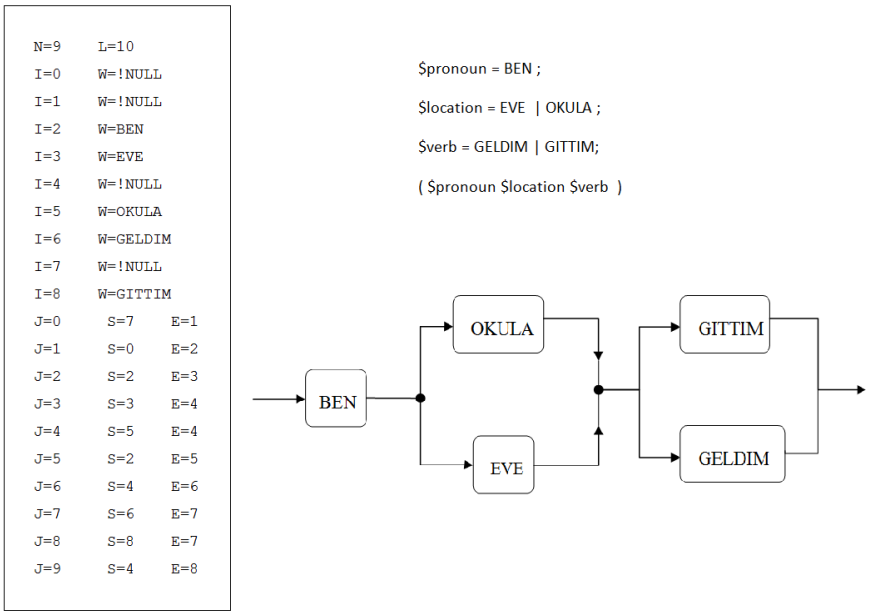

Figure 7.1: A simple grammar, and its corresponding word network and SLF representation

Using the EBNF grammar notation of HTK, rules can be defined to put constraints on ordering of letters, syllables, stem-ending pairs, morphemes, morphs and words when forming valid expressions in a given language. It is possible to combine different unit types in the same network as proposed in [63, 64]. Figure 7.2 shows an example grammar made of morphemes with its corresponding network.

A HTK word network generated with the stem-ending approach would have variable complexity based on how much constraint is applied on stem-ending matches. At the base level, a stem can be followed by any ending. In that case, only the N-gram language model employed may have an affect on the matches. When the stems and suffixes are divided into nominal and verbal categories, its possible to allow matching of stems with endings

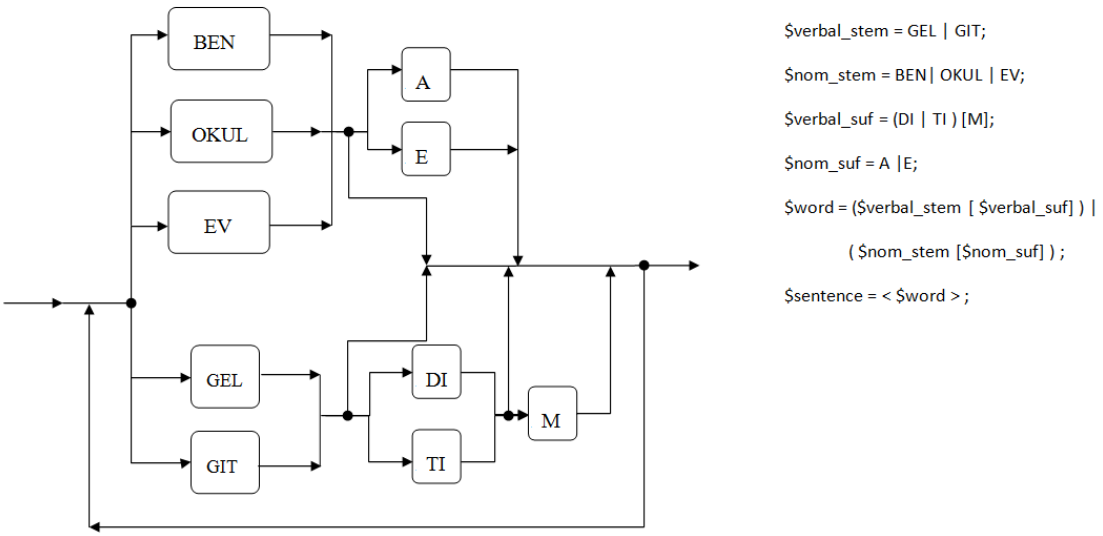

Figure 7.2: A simple stem-morpheme grammar and its network

from the same category only. Another constraint can be as enforcing vowel harmony between stems and endings as applied in [64].

Without applying an N-gram language model or any other constraints on stem-ending matches, a grammar with N stems and M endings has $2\times(N+M)+2$ links and $N+M+4$ nodes as shown in Figure 7.3.

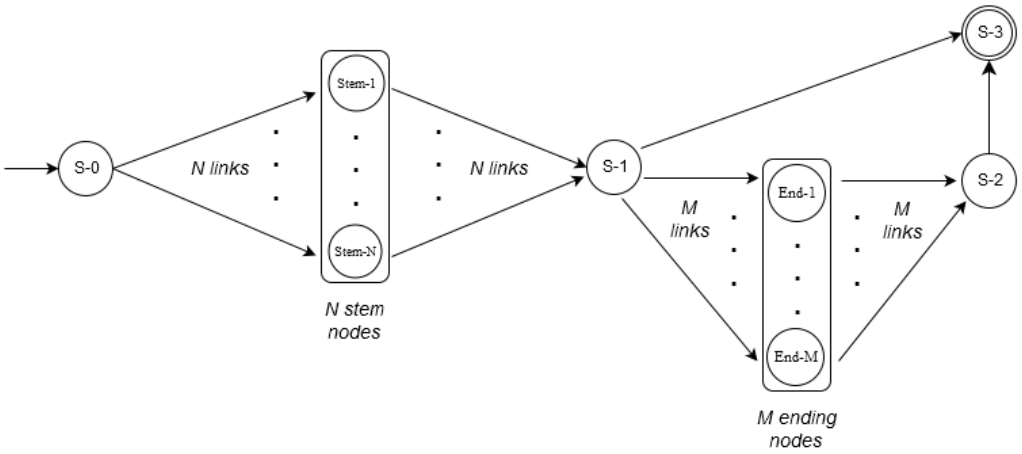

Figure 7.3: A HTK network for a generic stem-ending grammar

The network for stem-morpheme solution is the most complex one since it incorporates the FSAs of verbal and nominal morphotactics. Figure 7.4 shows a high-level view of the stem-morpheme grammar, where the nodes of suffix FSA are:

De_V_to_V: Derivational suffixes which derive a verbal stem from another verbal stem

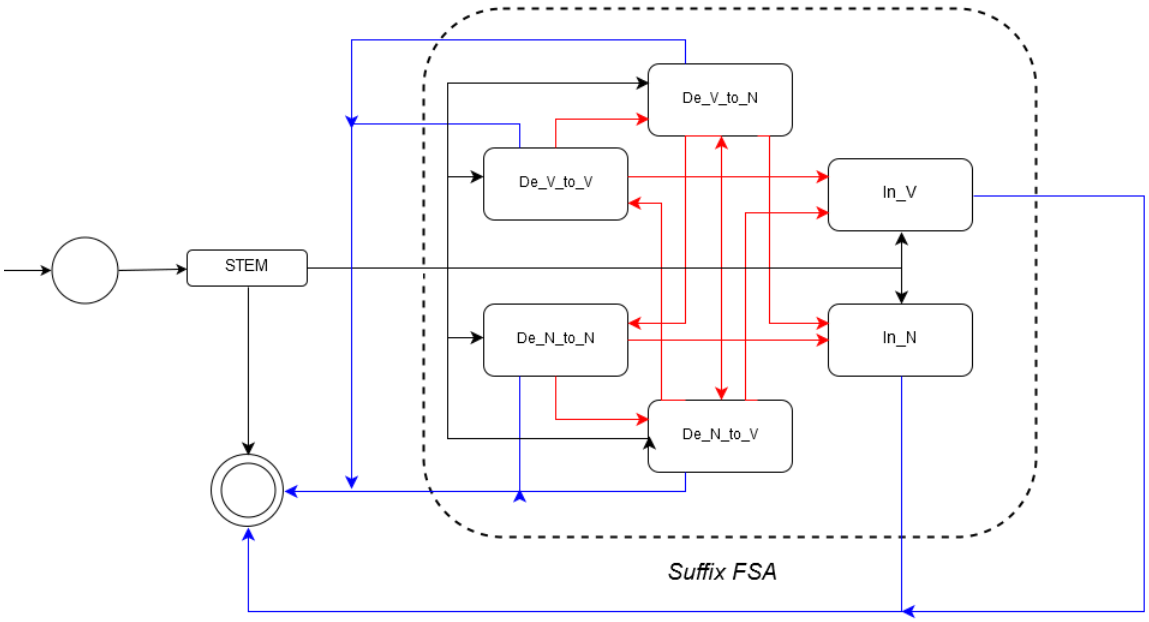

Figure 7.4: A high-level view of the stem-morpheme network

De_N_to_N: Derivational suffixes which derive a nominal stem from another nominal stem

De_V_to_N: Derivational suffixes which derive a nominal stem from a verbal stem

De_N_to_V: Derivational suffixes which transform verbal stem from a nominal stem

In_V: Verbal inflectional suffixes

In_N: Nominal inflectional suffixes

Table 7.1 shows the number of nodes and links for all three vocabulary solutions, i.e. word-based, stem-ending-based and stem-morpheme-based. The stem-morpheme-based vocabulary design has many redundancy that can be reduced by merging nodes sharing the same set of incoming and outgoing nodes. The reduced stem-morpheme network is obtained by using SRILM lattice-tool iterative reduction process to make two forward-backward node merging passes on the original network.

7.2.1 Decoding

HVite, HTK's Viterbi decoding tool, is utilized for recognizing the test set after the training is complete. HVite is run in lattice generation mode with N-best decoding where N is set to 15. A pruning of the search space is applied through the maximum number of active models parameter which is set to 10,000. The grammar networks used at this stage do not

Table 7.1: HTK decoding network sizes of alternative vocabulary types in terms of number of nodes and links

Vocabulary type	Number of nodes	Number of NULL nodes	Number of links
Words	130,003	3	260,001
Stem-ending	12,472	4	24,938
Stem-morpheme	73,750	1401	151,050
Stem-morpheme (reduced)	13,634	677	52,531

incorporate any N-gram language model and they reflect only the constraints imposed by the grammars.

Language Model Integration

As explained previously in Section 2.1.2, the recognizer tries to find the (word) sequence with the maximum probability during the decoding phase:

$$\hat{W} = argmax P(O|W)P(W) \tag{7.1}$$

The probability has two components here, the optical model score as the observation likelihood $P(O|W)$ and the language model score as the language prior $P(W)$ that should be weighted before combined together. A language scaling factor, LSF, is applied to the language probability in the form of an exponent ([45]):

$$\hat{W} = argmax P(O|W)P(W)^{LSF} \tag{7.2}$$

which has the effect of decreasing the value of the language model. Another addition to Equation 7.2 is the Word Insertion Penalty (WIP) which makes it probable to balance insertion errors versus deletion errors in order to minimize Word Error Rate:

$$\hat{W} = argmax P(O|W)P(W)^{LSF} WIP^{N} \tag{7.3}$$

where N is the length of the input sequence. LSF and WIP are hyperparameters and the optimum values are decided empirically on a separate validation set. They usually have considerable effects on the recognition performance.

2-gram and 3-gram language models are created by using SRILM's ngram-count tool that first builds an internal N-gram count set from the BOUN Web Corpus and then estimates a backoff N-gram model using the Kneser-Ney discounting method.

Two different methods of decoding with a language model, lattice expansion and lattice rescoring are explained in Section 3.3 previously. Both methods are experimented for recognition of the test data.

In lattice rescoring, previously generated lattices are rescored with these N-gram models by SRILM's lattice-tool. After rescoring, the lattices are decoded again to obtain new recognition results.

As an alternative to lattice rescoring, a single pass recognition with a grammar network which is expanded with the language model is tried. SRILM's lattice-tool is utilized in expansion mode for this purpose. Lattice expansion process modifies the lattice so that all possible N-grams appear in the lattice. It may add new nodes and links to achieve that.

7.3 Post-processing

None of the grammars put constraints to force vowel harmony, consonant harmony and other orthography rules, so the results from Viterbi decoding stage is post-processed for correction of orthographical errors. Table 7.2 shows some examples of modifications on the recognition results in the post-processing stage. The impact of that process on recognition accuracy is reported in Chapter 8.

Table 7.2: Examples of modifications on the raw recognition results in the post-processing stage

Raw	Postprocessed	Applied Rule
konuş + ti	konuş + tu	vowel harmony
sokak + dan	sokak + tan	consonant harmony: in suffix
küçük + ü	küçüğ + ü	consonant harmony: in stem
kitab + da	kitap + ta	consonant harmony:in stem, in suffix
duy + tı	duy + du	vowel harmony, consonant harmony: in suffix
büyük + im	büyüğ + üm	vowel harmony, consonant harmony: in stem

Chapter 8

Results

This chapter presents the results of two main groups of experiments. The delayed stroke handling methods which are detailed in Section 5.1.3 are evaluated with the detection algorithms of Section 5.1. In another set of experiments, vocabulary design alternatives of Section 5.2 are tested along with N-gram models. Word accuracy rate is the evaluation metric used for all experiments.

8.1 Evaluation of Delayed Stroke Handling Methods

8.1.1 Methodology

The UNIPEN cross validation split (see Section 6.1.1) is used in that set of experiments. Other than this, the system setting is the same with the one described in Section 7.1.

In this study, five of these subsets which share a common set of 1,000-words in their lexicons are chosen for testing effects of proposed algorithms and handling methods with respect to lexicon sizes as well. For the 1,000-word tasks, number of test samples ranges between 3720 and 4393.

Two series of experiments are conducted with 3,500-word and 1,000-word lexicons in order to be comparable to results with same lexicon sizes reported in literature. In the first set of experiments, each one of the 5 subsets serves as the hold-out test set while the rest of the 9 subsets are used for training. In the second set of experiments, test sets are reduced so that they contain only the samples of the 1,000 common words.

The chosen delayed stroke methods for evaluation are:

- (1) No handling: The recognition performance without any delayed stroke handling is given as the baseline.
- (2) Removal of all delayed strokes: Delayed strokes are detected and removed from the sample word.
- (3) Embed all : All of the delayed strokes, i.e. i-dot and t-cross type delayed strokes, are reordered in time so that they are repositioned after the corresponding letter body.

- (4) Remove i-dots, embed t-crosses as proposed in this paper.
- (5) Binary hat-feature representation of delayed strokes with removal.
- (6) Binary hat-feature representation of delayed strokes without removal, but extended feature representation.

The handling methods are tested with appropriate definitions to achieve their intended effect. Thus embedding and hat feature approaches use the proposed definition, while removing was tested with both definitions.

8.1.2 Results on UNIPEN

Table 8.1 shows the results obtained over 5 test subsets with the 3,500-word lexicon and those obtained with the 1,000-word lexicon are given in Table 8.2. According to these results, removal of the delayed strokes using the proposed definition in Algorithm 1 performs best, with 2.01 and 2.13% points above the baseline (i.e. no special handling). The second best approach is the hybrid method, followed by embedding. All three methods are found to show statistically significant improvements over the baseline in the 5 experiments, while the hat-feature based approaches fail to excel over the baseline. The statistical significance is calculated using paired t–tests with results of the five sets of experiments.

Table 8.1: Results for the 3,500-word task on UNIPEN.

Handling method	Definition	Accuracy (%)
Baseline		81.01 $\pm$ 2.66
Remove all	Proposed	**83.02** $\pm$ 2.32
Remove dots - embed crosses	Proposed	82.73 $\pm$ 2.35
Embed all	Proposed	82.20 $\pm$ 2.82
Remove all	DetectAll	80.79 $\pm$ 2.29
Hat-feature, without removal	Proposed	79.12 $\pm$ 2.66
Hat-feature, with removal	Proposed	79.07 $\pm$ 2.69

It is clear that different handling methods have different shortcomings. Embedding delayed strokes for order correction is heavily afflicted with unaligned delayed strokes which are observed to be of i-dots in most of the time (Figure 8.1.a and Figure 8.1.b). Unusual writing styles which do not comply with the heuristics for deciding attachment points are another source of error (Figure 8.1.c). Also, incorrect type detection of delayed strokes sometimes leads to incorrect repositioning when type-specific embedding strategies are applied (Figure 8.1.d). Lastly, since this method always integrates dots and crosses to character bodies, recognizers model such characters with those strokes. So, it suffers from omitted dots and crosses more than some other methods. The main problem with removing delayed strokes method is losing distinguishing information, which in turn leads to confusion with other characters.

Table 8.2: Results for the 1,000-word task on UNIPEN.

Handling method	Definition	Accuracy (%)
Baseline		83.99 ± 3.75
Remove all	Proposed	**86.12** ± 3.03
Remove dots - embed crosses	Proposed	85.44 ± 3.29
Embed all	Proposed	85.08 ± 3.73
Remove all	DetectAll	84.54 ± 3.15
Hat-feature, without removal	Proposed	81.72 ± 4.14
Hat-feature, with removal	Proposed	82.76 ± 3.28

Case errors are common for all methods. If the case errors are ignored, scores increase by ~ 1.6 points in 1,000-word task and ~ 2.9 points in 3,500-words task which changes the best recognition rates by the removal method as 87.73% (1,000-word) and 85.92% (3,500-word).

Table 8.3: Results on the ElementaryTurkish dataset.

Handling method	Definition	Accuracy (%)
Baseline		89.14
Remove all	Proposed	**91.17**
Embed all	Proposed	90.58
Remove dots - embed crosses	Proposed	89.94
Remove all	DetectAll	86.82
Hat-feature, with removal	Proposed	91.05

8.1.3 Results on ElementaryTurkish Dataset

Since the number of training samples of ElementaryTurkish dataset is very small, UNIPEN training data and Turkish data are combined for training. Specifically, models of characters which are common for English and Turkish are trained with both UNIPEN and ElementaryTurkish samples, while the others are trained with relevant samples in the corresponding dataset.

Table 8.3 shows performances of 3 methods on ElementaryTurkish dataset. Removal of delayed strokes according to the proposed definition achieves a 2,03 points improvement over the baseline. Performances of hat-feature, embedding method and the hybrid methods are above the baseline by 1,91, 1,44 and 0,8 points respectively.

8.1.4 Discussion

A set of observations based on the results of experiments and analysis of error cases is given below:

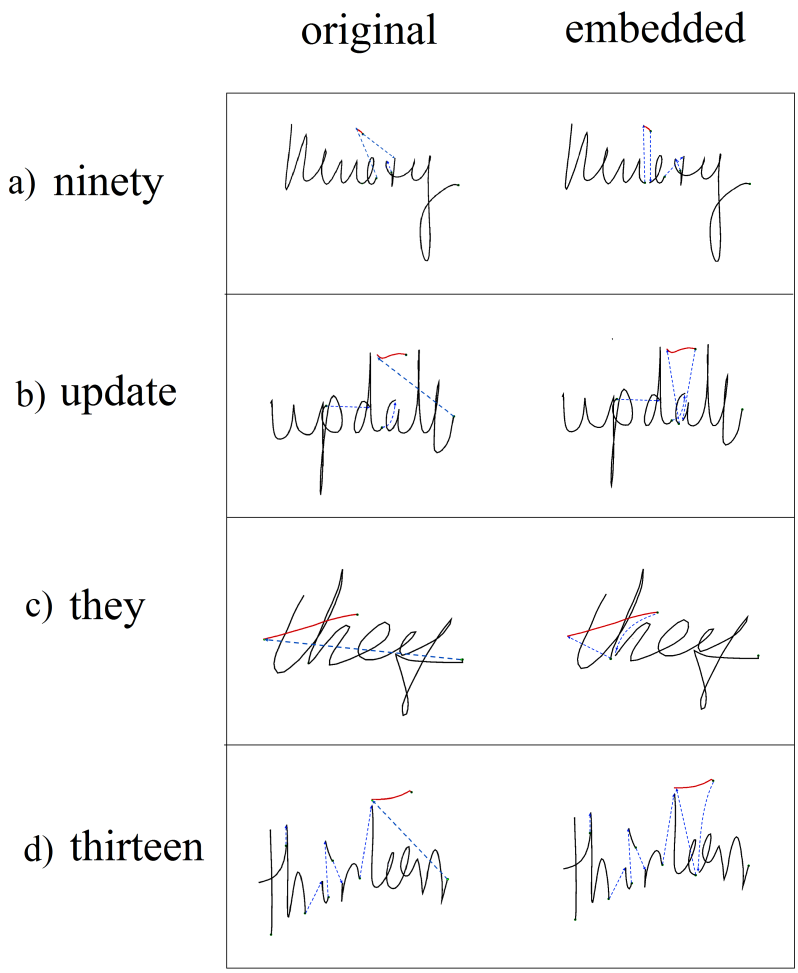

Figure 8.1: Examples for sources of error with resulting incorrect embeddings: (a) and (b) unaligned delayed strokes; (c) unusual writing style; (d) incorrect type detection.

- Removing all delayed strokes performs the best for both English and Turkish. This is a surprising finding for Turkish, since diacritical marks help differentiate between many similar words in Turkish (e.g. "ol" vs "öl"; "oldu" vs "öldü"); that is most probably due to the small lexicon where the amount of collision is significantly smaller. Furthermore, extra Turkish characters become the same as their diacritic-free counterparts in English (e.g. 'ö' becomes 'o' and 'ç' becomes 'c') with the removal methods. Considering this, the character models trained with both UNIPEN and ElementaryTurkish datasets with the removal methods are utilized, rather than the ones trained with the the latter one only. The use of the larger data can be another factor in high performance of the removal method.

- Removal, embedding and their hybrid and hat-feature, all perform better than baseline, for both English and Turkish.

- The relatively lower success of the embedding approach can be attributed to faulty decisions at choosing the attachment point when relocating a delayed stroke. With more accurate decisions and more Turkish samples for training, this method has potential to achieve better performance.
- The hat feature approaches which were used in a number of studies [53, 54, 2] and were reported to bring a small (0.5%) improvement [53], are observed to underperform the baseline for English
- The hat feature method shows a significant improvement over the baseline for Turkish. The slightly lower performance compared to the removal method can be explained by the use of binary feature which is the only difference between these two methods.
- Removal of *all* i-dots and t-crosses, whether delayed or not, performs worse than the baseline in English and Turkish, indicating that orderly written delayed strokes are useful for recognition.

8.1.5 State-of-the-art

There are a limited number of studies on isolated word recognition using the UNIPEN dataset. In one of the earlier studies, Hu et al. [7] use a two-stage delayed stroke modeling combined with N-best decoding for large vocabulary tasks. For testing, they use a subset of devtest_r01_v02 that include more than 1,500 samples from 100 writers which are not in the training set and report recognition rates of 90.5% and 87.2% for 1,000- and 2,000-word lexicons respectively. Unfortunately these test sets are not publicly available.

Marukatat et al. use Neural Networks (NNs) to predict the emission probabilities in a hybrid system combining HMMs with NNs [12]. They train the system with 30K words by 256 writers from UNIPEN dataset. They do not give particulars of their test set, but report 80.1% and 77.9% word recognition rates for multi-writer and writer-independent (omni-writer) recognition tasks respectively, with a 2,000-word lexicon.

Gauthier et al. also use a hybrid approach, with the combination of an online HMM-NN and an offline HMM system [19]. Using 40K words written by 256 writers from UNIPEN dataset for training, the combined classifier is tested on *parts of* a 1K sample set written by the *same* training set writers, all chosen from lowercase words. They report a 87% recognition rate with a 1,500-word lexicon, which decreases to 79% when the lexicon size is increased to 10,000 words.

The proposed recognition system of this thesis achieves 86.12% accuracy as the average best result with the 1,000-word lexicon. That result is in accordance with what has been reported on the UNIPEN database before, using similar size lexicons. However, none of the results reported so far are directly comparable, as they have been obtained using different subsets of the UNIPEN databases or different conditions (e.g. writer-independent vs. multi-writer) or different test sample selection procedures. For instance, [7] composes its test set from $devtest\_r01\_v02$ which is not publicly available and [19] uses the same writers with those of their training set. In this work, the training set is split

into 10 writer-independent subsets completely randomly to avoid any biases. Also, larger test sets are used in terms of both number of samples (3.5-4K for 1,000-word test) and number of writers for more reliable evaluation.

In addition to varying test set and lexicon size differences, many of the reported results are obtained with a subset of the writers. As stated before, UNIPEN dataset includes more than 3K writers in $train\_r01\_v07$ which show a large variance in writing styles. All of the available data is used in both training (2,800 writers, 65K samples) and testing, for a realistic setup. Also, large test sets (3.5-4K for 1,000-word test) are used rather than randomly selecting a subset of the available data.

8.2 Evaluation of Vocabulary Design Alternatives

8.2.1 Methodology

The system settings described in Section 7.1 are used for generating a baseline without a language model. The baseline has a delayed stroke handling mechanism as removal of such strokes. Alternative recognition vocabulary solutions in Section 5.2.4 are evaluated with and without N-gram language models. Specifically, bi-gram language models of stem-ending and stem-morpheme representations are trained on the Boun Web Corpus. A tri-gram model is trained for only the latter one.

Integration of the language models are done with a lattice rescoring and lattice expansion methods. All of the tests are conducted on the test set of ElementaryTurkish dataset.

The sizes of lexicons and test set coverage rates for alternative vocabulary designs are shown in Table 8.4. It should be noted that, the stem-ending and stem-morpheme vocabularies derived from the ET Dataset do not have 100% coverage on the test set, since the test words include some OOV as proper names.

Table 8.4: Alternative vocabulary designs using the BOUN Web Corpus

Source	Unit type	Size	Test coverage
ET	Word	800	100%
	Stem-ending	873+495	96.8%
	Stem-morpheme	873+342	97.8%
BOUN	Word	130K	75.6%
	Stem-ending	7K+5.5K	83.6%
	Stem-morpheme	7K+342	96.6%

Table 8.5 gives the perplexity values of N-gram language models on the ET test set which are calculated by the SRILM ngram toolkit. *ppl* is the geometric average of $\frac{1}{probability}$ of each token, i.e., perplexity while *ppl1* is the average perplexity per word excluding the $< /s >$ tokens.

Table 8.5: Language model perplexities according to vocabulary unit type

Unit type	N-gram	ppl	ppl1
Stem-ending	bi-gram	125	1853
Stem-morpheme	bi-gram	44	170
Stem-morpheme	tri-gram	36	132

8.2.2 Results

Table 8.6 serves as a baseline with word-based vocabulary results. Table 8.7 and Table 8.8 shows performances of sub-lexical vocabulary solutions with and without language models. All of the results shows the performance in terms of accuracy.

The recognition performances with vocabularies of the test set are labeled as *ET Dataset*. They are listed to show the utmost accuracy possible with the maximum test set coverage and the minimal vocabulary size within a particular setting.

Table 8.6: Word-based recognition results

Lexicon	Result
ET Dataset	91.7%
BOUN-Top130K	63.8%

Lattice expansion method performs far below the lattice rescoring technique for the stem-morpheme vocabularies, mainly because of the huge size and complexity of resulting lattices. So they are not included in the results in Table 8.8. However, with the expanded lattice, higher recognition rates are achieved, which is most probably due to use of a full search space instead of the reduced lattices as the rescoring method does.

According to the results, without using a language model, the word-based vocabulary shows the higher performance by 63.8%. Stem-ending and stem-morpheme vocabularies are quite below that accuracy rate.

Table 8.7: Stem-endings-based recognition results

Stem & Ending	LM	Raw	Post-processed
ET Dataset	-	52.5%	63.0%
	2-gram (expansion)	79.8	79.8
	2-gram (rescoring)	62.7	62.9
BOUN	-	44.6%	49.0%
	2-gram (expansion)	67.8%	67.9 %
	2-gram (rescoring)	62.9%	63.3 %

With the incorporation of the language models, the stem-ending design with a bi-gram model performs the best with an accuracy of 67.8%. If the results are post-processed, the accuracy increases slightly to 67.9%.

Table 8.8: Stem-morphemes-based recognition results

Stem	LM	Raw	Post-processed
ET Dataset	-	46.6%	52.6%
	2-gram	51.0%	54.4%
	3-gram	51.5 %	55.3%
BOUN	-	36.8%	41.2%
	2-gram	48.1%	49.6%
	3-gram	50.3%	51.3%

Stem-morpheme vocabularies achieve the highest recognition accuracy as 51.3% by integration of the tri-gram model through rescoring.

Comparing the results obtained by ElementaryTurkish dataset, which shows the utmost performance any of the vocabulary alternatives can have, the order with respect to accuracies is as expected: word-based vocabulary surpasses the others by 91.0% accuracy and stem-ending vocabulary is the second one with 79.8%. The stem-morpheme solution has the least accuracy as 55.3%.

8.2.3 Discussion

According to the results shown in the Table 8.6, Table 8.7 and Table 8.8 , the highest accuracy (91.7%) is obtained when the word-based lexicon is derived from the test set itself, which is the baseline. When the large, general purpose word-based lexicon derived from the BOUN Corpus is used, the accuracy drops significantly to 63.8%. The large drop in performance can be mainly attributed to the low test set coverage of that lexicon (75.6% as seen in Table 8.4), as OOV rates directly impacts overall accuracies. Another factor affecting the performance is the higher confusability among words in a larger lexicon.

In contrast, the general purpose stem+ending lexicons extracted from the BOUN Corpus and used with bi-gram language model applied via lattice expansion, results in the best accuracy of 67.9%. It should be noted that despite a much smaller lexicon and having the same text corpus coverage, this accuracy is better than the one obtained with a general purpose word-based lexicon (63.8%). With a stem+ending lexicon derived from the test set itself, the highest accuracy is 79.8%.

Based on these results, it is found to be important to use a language model with stem+ending lexicons, without which the accuracy falls significantly below word-based lexicon results (49.0% versus 63.8%). Stem-morpheme lexicons benefit from the use of language models as well. As for the methods for incorporating language models, lattice expansion performs better than lattice rescoring for the stem+ending lexicons with the bigram model. With the expanded lattice, higher recognition rates are achieved, probably due to the use of the full search space instead of the reduced lattice. In contrast, lattice expansion performs far below the lattice rescoring technique for the stem-morpheme vocabularies, mainly because of the huge size and complexity of resulting lattices. Post-processing is not very useful for both stem-ending and stem-morpheme lexicons when

N-gram models are used, but it improves the accuracy slightly when no language modeling is applied.

Analysis of errors of different vocabulary types shows that, each vocabulary has different shortcomings and advantages. For example, stem-ending solution benefits from having the suffixes in groups, which avoids the risk of lower recognition performance due to shorter units as in the case of stem-morpheme vocabulary. On the other hand, the stem-morpheme solution scales better as can be seen by a comparison to the results from using only the ElementaryTurkish dataset (ET Dataset) which has a 1956-word lexicon. The difference between tri-gram results of ET Dataset and BOUN dataset stem-morpheme vocabularies is only 4 points, although the lexicon size of the latter is significantly larger.

When the lexicon size gets larger, the difference between stem-ending and stem-morpheme vocabularies decreases to 16.6 points from 24.3. Similarly, in the large vocabulary setting, stem-morpheme performance is below that of word based vocabulary by only 11.8 points.

Nevertheless, stem-morpheme vocabularies suffer from increased optical confusability in decoding process due to shorter units.

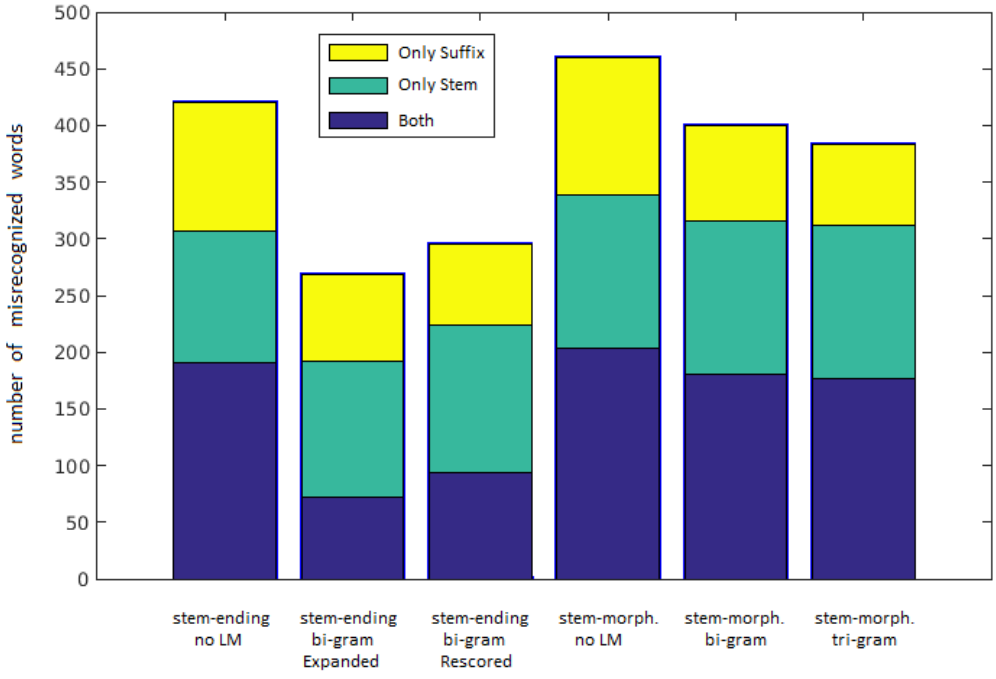

Figure 8.2: Distribution of misrecognized parts with the total number of errors for each vocabulary type

Figure 8.2 shows the error distribution according to misrecognized word parts as stem or suffixes. Integration of the language models have a clear positive effect as decreasing the errors in suffix parts. However, it can not help with the stem errors in stem-morpheme

vocabularies and even the number of misrecognized stems increases in stem-ending vocabularies.

Table 8.9 shows some examples from the misrecognized words by the best performing stem-ending system while Table 8.10 shows misrecognized words by the best performing stem-morpheme system, along with their incorrect predictions.

Ground Truth	Recognized
ağırdı	çağırdı
Kitabını	Kitabın
baktık	taktik
Futbolcu	Futbola
kitapla	kitaptı

Ground Truth	Recognized
Akşam	Aksam
Hayal	Hayat
ördeği	önceyi
dönüyorsa	donuyorsa
bulunan	bilinen

Table 8.9: Examples from recognition errors with large lexicon stem-ending vocabulary

Ground Truth	Recognized
Ağaçların	ihtiyac+ının
ağırdı	çağır+tı
bağlanınca	bağlan+ınız
gergedanın	gergi+lerin
günlerinde	öğün+lerinle
istiyorum	iş+tiyorum

Ground Truth	Recognized
kanatlarım	Kanat+ışım
okuduğunu	okul+cuğunu
oynamalıyız	oyna+mışlıyız
paylaşır	Dal+ışın
Ressam	Bas+sam
şeklinde	sek+tirte

Table 8.10: Examples from recognition errors with large lexicon stem-morpheme vocabulary

A misrecognized stem may cause a suffix error since the search is directed to presumably a wrong part of the search space, especially when a language model is used. Hence, it is crucially important to recognize the stem part correctly. The problem intensifies with lattice rescoring method where only a part of the whole search space used for decoding. An incorrect recognition in the stem part leads to generation of a lattice may not include the correct stem and the suffix(es) within the n-best hypotheses.

During the bi-gram calculation for stem-ending vocabularies, the probability of a word W is calculated as

$$P(< s > stemending < /s >) = P(stem| < s >)P(ending|stem)P(< /s > |ending) \tag{8.1}$$

and

$$P(stem| < s >) = \frac{Count(< s > stem)}{Count(< s >)} = \frac{Count(stem)}{Count(words)} \tag{8.2}$$

since starting and ending markers, $< s >$ and $< /s >$, are added to each word automatically to give the proper bi-gram context to stem and suffix parts. So, the frequency of a stem in the text corpora is becomes crucially important when decoding is done with the bi-gram (and generally N-gram) model for stem-ending and stem-morpheme vocabularies.

It is observed that the BOUN Web Corpus contains some abbreviations and proper names that frequently appear in the news context or on the Internet texts although they are rare in the daily usage. For example, the abbreviation *CIA* is the most frequent 6017th word while frequency order of *Richard* and *show* in the corpus are 4862 and 6867 respectively. Since the stems of sub-lexical vocabularies are chosen according to frequency of occurrence, stem recognition is negatively affected by this situation.

Post-processing is not useful for stem-ending designs as long as an N-gram model is incorporated. It is justifiable since the suffixes are extracted in groups from the corpus, so they do not need any involvement for orthography rules. As for the stem-morpheme vocabularies, post-processing helps increasing the accuracy in all cases, but its significance diminishes with use of the language models which is an indication of their effectiveness.

To the best of the author's knowledge, there is no reference to a *large vocabulary* online Turkish handwriting recognition system in the literature. Hence, a direct comparison with another study is not possible for this work. In offline handwritten Turkish text recognition, Yanikoglu and Kholmatov use the HMM letter models previously developed for English, by mapping the Turkish characters to the closest English character (the input of the word güneş is recognized as gunes) [83]. They report 56% top-10 word recognition rate using an 17,000-word lexicon obtained from a newspaper corpus.

Chapter 9

Conclusion

9.1 Summary

In this thesis an online handwriting recognition system for isolated Turkish words is designed and implemented using the HMMs for the first time in the literature. The proposed system achieves state-of-art results comparable to those obtained for English, through careful study of all components of the recognition system, from preprocessing to language modeling. Two main problems specific to Turkish language is defined and explained. Specifically, the delayed strokes problem due to a large number of diacritical marks and high OOV rates and high vocabulary similarity due to the agglutinative nature of the language are discussed thoroughly and some methods are proposed to overcome these problems. Moreover, a dataset of around 10,000 online Turkish handwritten words is made publicly available to the researchers for the first time.

A working definition for describing what constitutes a delayed stroke is systematically developed, keeping the definition simple while covering most instances. A clear evaluation of the definition is made to asses its accuracy.

Using the new definition, several delayed stroke handling methods that are suggested in the literature are evaluated for their effectiveness in English and Turkish text. By making the experimental setting, including the dataset split, publicly available, reproducible and comparable results are presented. It is found that recognition accuracy can be increased with proper handling of the unorderly written strokes.

The results of experiments are mostly in parallel for English and Turkish. Both removing delayed strokes and embedding them in the writing order improve over the baseline (i.e. not doing any preprocessing) for the two scripts. Removal of all delayed strokes is the best option for both languages, with improvements over the baselines are up to 2.13% and 2.03% points for English and Turkish respectively. Embedding is the second best approach presumably due to the nontrivial task of deciding correct attachment point for reordering.

The overall system performances are assessed as 86.12% with a 1000-word lexicon and 83.03% with a 3,500-word lexicon for English, which are both in accordance with the state-of-the-art. The proposed recognition system achieves 91.7% word recognition

accuracy with a middle-sized, 1,950-word lexicon with 0% OOV for Turkish.

Another contribution of this work is the investigation of suitable solutions for the OOV problem within the context of large vocabulary *online handwriting* recognition. Using large text corpora, alternative recognition vocabularies with various grammatical units are created and then evaluated for their coverage on both the training corpus and the test set. Also, N-gram language models that are derived from the same corpora are integrated to decoding process by means of lattice expansion and lattice recognition techniques. Within this effort, effective search space reduction is made by using a finite state automaton of suffixes that is represented as a context-free grammar.

The best performance, 67.9% is obtained by the stem-ending vocabulary that is expanded with a bi-gram model, with a lexicon of size 12,500 (stems and endings total). This result is superior to the accuracy of the word-based vocabulary (63.8%) with the same training corpus coverage of 95%. It can be concluded that the stem-ending vocabulary design has the potential of being a viable solution to the high OOV rate problem of Turkish.

Use of a finite state automaton for representation of suffix morphotactics in the decoding network brings a significant improvement on the coverage with an increase of 21 points over the word-based vocabulary from 75.6% to 96.6%. Also, it provides an effective means for reduction of the decoding network by enforcing the morphotactics which helps better recognition. However, the recognition accuracy still falls below other approaches because of shorter recognition units which are known to lead to high optical and lexical confusability.

According to these results, the standard techniques of the literature may not be sufficient for general purpose Turkish handwriting recognition. A reference to the results reported by studies on Turkish speech recognition or large vocabulary handwriting recognition for other languages shows that the obtained results are in parallel with these in terms of language modeling impact although they are by no means directly comparable.

9.2 Future Perspectives

Addition of more features which can capture aspects of handwriting other than the current feature set may help increasing the optical modeling of the proposed system. Discriminative training [84, 85] of HMMs is another alternative which can be experimented in the future.

A promising alternative in the decoding process is to use the Weighted Finite State Transducer (WFST) [86] technology, which has became very popular in speech recognition recently [65, 70]. The WFST provides unified framework for describing models with increased search efficiency via optimization algorithms.

Also the recent deep learning techniques have a good potential for improving the recognition accuracy. However, much more Turkish handwriting data is needed in that case.

The embedding (i.e. order correction) method in delayed stroke handling can be improved in detection of attachment points. Currently, these points are decided by using

some hand-crafted rules based on a limited set of samples. A systematic analysis on a large, manually marked data and especially Turkish data may reveal more robust criteria for this purpose. Actually, current Turkish dataset is quite small and should be extended with more samples according to a well-defined dataset design.

A considerable amount of decoding errors occur at stem part of the words when sub-lexical units are used in the vocabulary. Selection of stems is based on frequency of occurrence in the current system. Processing the BOUN Web Corpus to remove unlikely proper names and abbreviations will be helpful in increasing stem recognition rate, and in turn overall performance. Also larger and more balanced text corpora will be much useful in creating vocabularies more suitable to a general purpose recognition system.

The complexity of decoding networks can be reduced by simple division of stems and endings into nominal and verbal categories. The suffix FSA of the stem-morpheme vocabularies already handles nominal and verbal suffixes separately. A further improvement with stem-ending vocabularies can be classification of stem-ending pairs according to Turkish orthography and specifically the vowel and consonant harmonies.

Finally, use of character N-gram models in combination with other lexical units can be integrated to the current system for recognition of OOV words.

Appendix A

Out-of-Vocabulary Words

The ElementaryTurkish dataset vocabulary items that are absent from the 130K word vocabulary which is created from the BOUN Web Corpus as explained in 5.2.4 are listed in this section. The list is prepared as case-sensitive.

A.1 OOV Items for the 130K Word-based Vocabulary

- Abla
- Ağacımız
- Ağlama
- Ahtapot
- Akşam
- Anlamı
- Anne
- Arkadaşlar
- ATA
- atamıyorlardı
- Atladım
- AZİM
- BAHAR
- Bak
- bakakaldın
- başarabildim
- Başarıları
- Başarınızı
- batmaktan
- Bazen
- Bazı
- Bence
- Bilin
- bilmecelerin
- binerim
- Birden
- Birer
- bireylerinizi
- bisikletine
- Bisikletini
- Biz
- Bizim
- böcektir
- boyadım
- boyayınız
- Böylece
- Boyu
- Buda
- Bugün
- Buluşumu
- büyüsem
- çağrıştırmaktadır
- Çalım
- çarşafım
- Dağdan
- DEĞERLENDİRMESİ

- DENİZATI
- dinlemeliyiz
- dişlerim
- doğduklarında
- dönüyorsa
- Dostlarımla
- Dünya
- Dünyaya
- duramayıp
- Elimizde
- Elma
- EREN
- eşleştiriniz
- Etkinlik
- Evine
- Fildir
- filsin
- Futbolcu
- gagalamasın
- gagalıyor
- Garip
- geçmeyenlerin
- Gelen
- gergedanın
- gidebilsin
- Gittikçe
- göremezsek
- görseli
- görseller
- Görsellerin
- görsellerinde
- görsellerle
- götürmem
- gözcüyüz
- gücenmişti
- Güneşi
- Güneştir
- Hadi
- Harika
- hatırlamıyordu
- Hayal
- Hayalinizle
- Hayatta
- Hayır
- Hayretle
- HAYVANIMI
- hecelerden
- Hepimiz
- Herkes
- Hikayede
- hızlanmıştı
- hoşlandınız
- ışıldatıp
- Istakoz
- izciyiz
- kaçalım
- kaçındım
- KADIOĞLU
- Kalaylı
- kaldırdığımda
- kanatlarım
- kararmıştı
- Kardeşlerin
- Karşımda
- Kirpinin
- Kısaltılmıştır
- Kitabını
- kızması
- KONSERİ
- Konuşmak
- Konusu
- korunağı
- KÜÇÜK
- Kullanacağım
- kümeste
- Kurşun
- KUŞKUCU
- kuşkulanan

- kutucuğu
- martılardan
- Merhaba
- Metindeki
- Meyvelerden
- Ne
- Ördeklerin
- Ormanda
- ORMANDAKİ
- Oynarken
- oyuncağım
- özlemeye
- Peşimden
- Pikniğe
- pufluyordu
- RESİMLER
- Ressam
- rüyamı
- Rüzgar
- salıver
- Sarkaç
- Sarsın
- şaşırtabilirler
- Sayfa
- Sayfalarını
- Sayın
- Sebzelerden
- Sen
- Sevdiğim
- Sıcacık
- Sıcak
- sınıflayarak
- Sonunda
- sürmemişti
- Suya
- Tahterevalli
- Tam
- TANITIYORUM
- TAVUK
- Tavuklar
- tavuktu
- TEKRAR
- temada
- Temamız
- Teyzeye
- TEZCAN
- Tilkispor
- Tilkisporlu
- Toplam
- Toplandık
- Tüh
- Tuttuğum
- Uçak
- uçun
- UÇURTMA
- uçurtmam
- uçurtmanız
- Uçuşumuz
- umursamadı
- üretiniz
- uykusundaydık
- UYUDUĞU
- uyusam
- Vah
- Varılmaz
- Yarışmaya
- yaşayabilmiş
- yavrusuydum
- yemeyeyim
- Yorgunum
- yorulmuşsun
- Yüzünü
- ZENGİN

Bibliography

[1] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[2] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel coNNectionist system for unconstrained handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 5, pp. 855–868, 2009.

[3] R. Plamondon and S. N. Srihari, "On-line and off-line handwriting recognition: A comprehensive survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 63–84, 2000.

[4] B. M. Al-Helali and S. A. Mahmoud, "Arabic online handwriting recognition (AOHR): A survey," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 33:1–33:35, 2017.

[5] N. Tagougui, M. Kherallah, and A. M. Alimi, "Online Arabic handwriting recognition: a survey," *IJDAR*, vol. 16, no. 3, pp. 209–226, 2013.

[6] D. S. DoermaNN and S. Jaeger, eds., *Arabic and Chinese Handwriting Recognition - SACH 2006 Summit College Park, MD, USA, September 27-28, 2006 Selected Papers*, vol. 4768 of *Lecture Notes in Computer Science*, Springer, 2008.

[7] J. Hu, S. G. Lim, and M. K. Brown, "Writer independent on-line handwriting recognition using an HMM approach," *Pattern Recognition*, vol. 33, no. 1, pp. 133–147, 2000.

[8] A. Biem, "Minimum classification error training for online handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 7, pp. 1041–1051, 2006.

[9] M. Liwicki and H. Bunke, "Handwriting recognition of whiteboard notes," in *Proceedings of the 12th Conference of the International Graphonomics Society*, pp. 118–122, 2005.

[10] M. Liwicki and H. Bunke, "Handwriting recognition of whiteboard notes - studying the influence of training set size and type," *IJPRAI*, vol. 21, no. 1, pp. 83–98, 2007.

[11] M. Schenkel, I. Guyon, and D. Henderson, "On-line cursive script recognition using time-delay neural networks and Hidden Markov Models," *Mach. Vis. Appl.*, vol. 8, no. 4, pp. 215–223, 1995.

[12] S. Marukatat, T. Artières, P. Gallinari, and B. Dorizzi, "Sentence recognition through hybrid neuro-markovian modeling," in *6th International Conference on Document Analysis and Recognition, ICDAR 2001, 10-13 September 2001, Seattle, WA, USA*, pp. 731–737, 2001.

[13] J. Schenk and G. Rigoll, "Novel hybrid NN/HMM modelling techniques for on-line handwriting recognition," in *10th International Workshop on Frontiers in Handwriting Recognition, IWFHR 2006, IAPR. , La Baule, France, Oct 2006*, pp. 619—6230, 2006.

[14] S. Garcia-Salicetti, B. Dorizzi, P. Gallinari, and Z. Wimmer, "Maximum mutual information training for an online neural predictive handwritten word recognition system," *IJDAR*, vol. 4, no. 1, pp. 56–68, 2001.

[15] M. Liwicki, A. Graves, H. Bunke, and J. Schmidhuber, "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks," in *Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR*, pp. 367–371, 2007.

[16] A. Graves, S. Fernandez, M. Liwicki, H. Bunke, and J. Schmidhuber, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Advances in Neural Information Processing Systems 20* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), pp. 577–584, Cambridge, MA: MIT Press, 2008.

[17] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel coNNectionist system for unconstrained handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 5, pp. 855–868, 2009.

[18] É. Caillault and C. Viard-Gaudin, "Mixed discriminant training of hybrid ANN/HMM systems for online handwritten word recognition," *IJPRAI*, vol. 21, no. 1, pp. 117–134, 2007.

[19] N. Gauthier, T. Artières, P. Gallinari, and B. Dorizzi, "Strategies for combining on-line and off-line information in an on-line handwriting recognition system," in *6th International Conference on Document Analysis and Recognition, ICDAR 2001, 10-13 September 2001, Seattle, WA, USA*, pp. 412–416, 2001.

[20] S. Jäger, S. Manke, J. Reichert, and A. Waibel, "Online handwriting recognition: the NPen++ recognizer," *IJDAR*, vol. 3, no. 3, pp. 169–180, 2001.

[21] M. Liwicki and H. Bunke, "HMM-based on-line recognition of handwritten whiteboard notes," in *ICFHR*, pp. 595–599, 2006.

[22] M. Liwicki and H. Bunke, "Feature selection for HMM and BLSTM based handwriting recognition of whiteboard notes," *IJPRAI*, vol. 23, no. 5, pp. 907–923, 2009.

[23] E. Vural, H. Erdogan, K. Oflazer, and B. A. Yanikoglu, "An online handwriting recognition system for Turkish," in *Document Recognition and Retrieval XII,DRR 2005,San Jose, California, USA, January 16-20, 2005, Proceedings*, pp. 56–65, 2005.

[24] A. Çapar, K. Tasdemir, Ö. Kilic, and M. Gökmen, "A Turkish handprint character recognition system," in *Computer and Information Sciences - ISCIS 2003, 18th International Symposium, Antalya, Turkey, November 3-5, 2003, Proceedings*, pp. 447–456, 2003.

[25] K. Kaplan, H. M. Ertunç, and E. Vardar, "Handwriting character recognision by using fuzzy logic," *Fırat University Turkish Journal of Science & Technology*, vol. 12, pp. 71 – 77, 2017.

[26] S. U. Korkmaz, G. Kirçiçegi, Y. Akinci, and V. Atalay, "A character recognizer for Turkish language," in *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pp. 1238–1241, 2003.

[27] N. Arica and F. T. Yarman-Vural, "Optical character recognition for cursive handwriting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 6, pp. 801–813, 2002.

[28] M. Şekerci, "Turkish coNNected and slant handwritten recognition system," Master's thesis, Trakya Üniversitesi, The address of the publisher, 7 2007. An optional note.

[29] B. Yanıkoğlu, A. Gogus, and E. İnal, "Use of handwriting recognition technologies in tablet-based learning modules for first grade education," *Educational Technology Research and Development*, vol. 65, no. 5, p. 1369–1388, 2017.

[30] C. Lee and B. Juang, "A survey on automatic speech recognition with an illustrative example on continuous speech recognition of mandarin," *IJCLCLP*, vol. 1, no. 1, 1996.

[31] E. Levin and R. Pieraccini, "Planar hidden markov modeling: From speech to optical character recognition," in *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pp. 731–738, 1992.

[32] A. Kaltenmeier, T. Caesar, J. M. Gloger, and E. Mandler, "Sophisticated topology of hidden markov models for cursive script recognition," in *2nd International Conference Document Analysis and Recognition, ICDAR '93, October 20-22, 1993, Tsukuba City, Japan*, pp. 139–142, 1993.

[33] T. Plötz and G. A. Fink, "Markov models for offline handwriting recognition: a survey," *IJDAR*, vol. 12, no. 4, pp. 269–298, 2009.

[34] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *Ann. Math. Statist.*, vol. 41, pp. 164–171, 02 1970.

[35] L. E. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bull. Amer. Math. Soc.*, vol. 73, pp. 360–363, 05 1967.

[36] S. Günter and H. Bunke, "Optimizing the number of states, training iterations and gaussians in an HMM-based handwritten word recognizer," in *7th International Conference on Document Analysis and Recognition, (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pp. 472–476, 2003.

[37] M. Schambach, "Model length adaptation of an HMM based cursive word recognition system," in *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pp. 109–113, 2003.

[38] M. Zimmermann and H. Bunke, "Hidden markov model length optimization for handwriting recognition systems," in *Proceedings of the Eighth International Workshop on Frontiers in Handwriting Recognition, IWFHR 2002, Niagara-on-the-Lake, Ontario, Canada, August 6-8, 2002*, pp. 369–374, 2002.

[39] N. Cirera, A. Fornés, and J. Lladós, "Hidden markov model topology optimization for handwriting recognition," in *13th International Conference on Document Analysis and Recognition, ICDAR 2015, Nancy, France, August 23-26, 2015*, pp. 626–630, 2015.

[40] K. T. Abou-Moustafa, M. Cheriet, and C. Y. Suen, "On the structure of Hidden Markov Models," *Pattern Recognition Letters*, vol. 25, no. 8, pp. 923–931, 2004.

[41] J. van Oosten and L. Schomaker, "A reevaluation and benchmark of Hidden Markov Models," in *14th International Conference on Frontiers in Handwriting Recognition, ICFHR 2014, Crete, Greece, September 1-4, 2014*, pp. 531–536, 2014.

[42] R. Rosenfeld, "Two decades of statistical language modeling: Where do we go from here?," in *Proceedings of the IEEE*, vol. 88, pp. 1270–1278, 2000.

[43] R. Rosenfeld, "A maximum entropy approach to adaptive statistical language modelling," *Computer Speech & Language*, vol. 10, no. 3, pp. 187–228, 1996.

[44] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, 2003.

[45] D. Jurafsky and J. H. Martin, *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence, Prentice Hall, Pearson Education International, 2009.

[46] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," in *1995 International Conference on Acoustics, Speech, and Signal Processing, ICASSP '95, Detroit, Michigan, USA, May 08-12, 1995*, pp. 181–184, 1995.

[47] C. E. Shannon, "A mathematical theory of communication," *Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[48] F. Weng, A. Stolcke, and A. Sankar, "Efficient lattice representation and generation," in *The 5th International Conference on Spoken Language Processing, Incorporating The 7th Australian International Speech Science and Technology Conference, Sydney Convention Centre, Sydney, Australia, 30th November - 4th December 1998*, 1998.

[49] A. Ljolje, F. Pereira, and M. Riley, "Efficient general lattice generation and rescoring," in *Sixth European Conference on Speech Communication and Technology, EUROSPEECH 1999, Budapest, Hungary, September 5-9, 1999*, 1999.

[50] S. M. Siniscalchi, J. Li, and C. Lee, "A study on lattice rescoring with knowledge scores for automatic speech recognition," in *INTERSPEECH 2006 - ICSLP, Ninth International Conference on Spoken Language Processing, Pittsburgh, PA, USA, September 17-21, 2006*, 2006.

[51] E. Arisoy and M. Saraclar, "Lattice extension and rescoring based approaches for LVCSR of Turkish," in *INTERSPEECH 2006 - ICSLP, Ninth International Conference on Spoken Language Processing, Pittsburgh, PA, USA, September 17-21, 2006*, 2006.

[52] K. Oflazer, "Turkish and its challenges for language processing," *Language Resources and Evaluation*, vol. 48, no. 4, pp. 639–653, 2014.

[53] S. Jäger, S. Manke, J. Reichert, and A. Waibel, "Online handwriting recognition: the NPen++ recognizer," *IJDAR*, vol. 3, no. 3, pp. 169–180, 2001.

[54] M. Liwicki, A. Graves, H. Bunke, and J. Schmidhuber, "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks," in *In Proceedings of the 9th International Conference on Document Analysis and Recognition, ICDAR 2007*, 2007.

[55] S. Abdelazeem and H. M. Eraqi, "On-line Arabic handwritten personal names recognition system based on HMM," in *2011 International Conference on Document Analysis and Recognition, ICDAR 2011, Beijing, China, September 18-21, 2011*, pp. 1304–1308, 2011.

[56] A. M. Alimi, "An evolutionary neuro-fuzzy approach to recognize on-line Arabic handwriting," in *4th International Conference Document Analysis and Recognition, ICDAR 1997, 2-Volume Set, August 18-20, 1997, Ulm, Germany, Proceedings*, pp. 382–386, 1997.

[57] J. Ha, J. K. S. Oh, and Y. Kwon, "Unconstrained handwritten word recognition with intercoNNected Hidden Markov Models," in *Third International Workshop on Frontiers in Handwriting Recognition, IWFHR 1993, IAPR. Buffalo, USA, May 1993*, pp. 455–560, 1993.

[58] N. S. Flann, "Recognition-based segmentation of on-line cursive handwriting," in *Advances in Neural Information Processing Systems 6, 7th NIPS Conference, Denver, Colorado, USA, 1993*, pp. 777–784, 1993.

[59] F. Biadsy, J. El-Sana, and J. Habash, "Online Arabic handwriting recognition using Hidden Markov Models," in *Tenth International Workshop on Frontiers in Handwriting Recognition, IWFHR 2007, IAPR. Newyork,USA, 2006*, 2006.

[60] I. Abdelaziz, S. Abdou, and H. Al-Barhamtoshy, "Large vocabulary Arabic online handwriting recognition system," *CoRR*, vol. abs/1410.4688, 2014.

[61] J. Hu, M. K. Brown, and W.Turin, "Handwriting recognition with Hidden Markov Models and grammatical constraints," in *Fourth International Workshop on Frontiers in Handwriting Recognition, IWFHR 1994, IAPR.Taipei, Dec 1994*, 1994.

[62] V. Ghods, E. Kabir, and F. Razzazi, "Effect of delayed strokes on the recognition of online Farsi handwriting," *Pattern Recognition Letters*, vol. 34, no. 5, pp. 486–491, 2013.

[63] E. Arisoy, H. Dutagaci, and L. M. Arslan, "A unified language model for large vocabulary continuous speech recognition of Turkish," *Signal Processing*, vol. 86, no. 10, pp. 2844–2862, 2006.

[64] H. Erdogan, O. Buyuk, and K. Oflazer, "Incorporating language constraints in sub-word based speech recognition," in *Recent Advances of Neural Network Models and Applications - Proceedings of the 23rd Workshop of the Italian Neural Networks Society, WIRN 2013, Vietri sul Mare, Salerno, Italy, May 23-25, 2013*, pp. 98 – 103, 01 2005.

[65] H. Sak, M. Saraclar, and T. Güngör, "Morphology-based and sub-word language modeling for Turkish speech recognition," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2010, 14-19 March 2010, Sheraton Dallas Hotel, Dallas, Texas, USA*, pp. 5402–5405, 2010.

[66] H. Sak, M. Saraclar, and T. Gungor, "Morpholexical and discriminative language models for Turkish automatic speech recognition," *IEEE Trans. Audio, Speech & Language Processing*, vol. 20, no. 8, pp. 2341–2351, 2012.

[67] M. Creutz, T. Hirsimäki, M. Kurimo, A. Puurula, J. Pylkkönen, V. Siivola, M. Varjokallio, E. Arisoy, M. Saraclar, and A. Stolcke, "Morph-based speech recognition and modeling of out-of-vocabulary words across languages," *TSLP*, vol. 5, no. 1, pp. 3:1–3:29, 2007.

[68] V. Siivola, T. Hirsimäki, M. Creutz, and M. Kurimo, "Unlimited vocabulary speech recognition based on morphs discovered in an unsupervised maNNer," in *8th European Conference on Speech Communication and Technology, EUROSPEECH 2003 - INTERSPEECH 2003, Geneva, Switzerland, September 1-4, 2003*, 2003.

[69] c. Çöltekin, "A set of open source tools for Turkish natural language processing," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)* (N. Calzolari, K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, and S. Piperidis, eds.), pp. 1079–1086, European Language Resources Association (ELRA), 2014.

[70] H. Sak, T. Güngör, and M. Saraclar, "Resources for Turkish morphological processing," *Language Resources and Evaluation*, vol. 45, no. 2, pp. 249–261, 2011.

[71] H. Erdogan, O. Buyuk, and K. Oflazer, "Incorporating language constraints in subword based speech recognition," in *IEEE Workshop on Automatic Speech Recognition and Understanding*, 2005.

[72] K. Oflazer, "Two-level description of Turkish morphology," in *Sixth Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference, 21-23 April 1993, Utrecht, The Netherlands*, 1993.

[73] M. Forsberg and A. Ranta, "BNF converter," in *Proceedings of the ACM SIGPLAN Workshop on Haskell, Haskell 2004, Snowbird, UT, USA, September 22-22, 2004*, pp. 94–95, 2004.

[74] M. Tomita, "LR parsers for natural languages," in *10th International Conference on Computational Linguistics and 22nd ANNual Meeting of the Association for Computational Linguistics, Proceedings of COLING '84, July 2-6, 1984, Stanford University, California, USA.*, pp. 354–357, 1984.

[75] GNU.org, "Gnu bison." Accessed: 2076-12-14.

[76] C. Viard-Gaudin, P. M. Lallican, P. Binter, and S. Knerr, "The IRESTE On/Off (IRONOFF) dual handwriting database," in *Fifth International Conference on Document Analysis and Recognition, ICDAR 1999, 20-22 September, 1999, Bangalore, India*, pp. 455–458, 1999.

[77] The UNIPEN Consortium, "The UNIPEN project." Accessed: 2017-12-26.

[78] M. Liwicki and H. Bunke, "IAM-OnDB - an on-line English sentence database acquired from handwritten text on a whiteboard," in *Eighth International Conference on Document Analysis and Recognition, ICDAR 2005, 29 August - 1 September 2005, Seoul, Korea*, pp. 956–961, 2005.

[79] H. Sak, M. Saraclar, and T. Gungor, "Morpholexical and discriminative language models for Turkish automatic speech recognition," *IEEE Transactions on Audio, Speech & Language Processing*, vol. 20, no. 8, pp. 2341–2351, 2012.

[80] S. S. Technology and R. Laboratory, "Srilm - the sri language modeling toolkit." Accessed: 2017-12-26.

[81] HTK, "The Hidden Markov Model toolkit htk." Accessed: 2017-12-26.

[82] A. Graves, S. Fernández, M. Liwicki, H. Bunke, and J. Schmidhuber, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First ANNual Conference on Neural Information Processing Systems, NIPS, Vancouver, British Columbia, Canada, December 3-6, 2007*, pp. 577–584, 2007.

[83] B. Yanikoglu and A. Kholmatov, "Turkish handwritten text recognition: a case of agglutinative languages," in *Document Recognition and Retrieval X, 22-23 January 2003, Santa Clara, California, USA, Proceedings*, pp. 227–233, 2003.

[84] T. M. T. Do and T. Artières, "Maximum margin training of gaussian HMMs for handwriting recognition," in *10th International Conference on Document Analysis and Recognition, ICDAR 2009, Barcelona, Spain, 26-29 July 2009*, pp. 976–980, 2009.

[85] P. C. Woodland and D. Povey, "Large scale discriminative training of hidden markov models for speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 25–47, 2002.

[86] M. Mohri, "Finite-state transducers in language and speech processing," *Computational Linguistics*, vol. 23, no. 2, pp. 269–311, 1997.