Copy Mechanisms for Upstream Text Processing

A thesis submitted for the degree of

Bachelor of Engineering in Computer Science

Sameer Jain

2015A7TS0068P

Supervisor: Dr. Tanja Samardžić

Director, Language and Space Lab, University of Zurich

Co-supervisor: Dr. Yashvardhan Sharma

Associate Professor, Department of Computer Science and Information Systems, BITS Pilani

May 2019

Abstract

The upstream tasks of morphological inflection generation, morphological segmentation, and normalization, have recently been modeled as sequence-to-sequence learning tasks in the deep learning framework.

They can all be viewed as string transduction tasks in which significant portions of the output strings come directly from sections of the input strings. Recently, sequence-to-sequence neural models have been developed which are designed to perform well in settings where there is a need to often copy characters (or words) from input to output sequences. It is possible, therefore, for neural models for the tasks of inflection generation, segmentation, and normalization to be benefited from the use of such mechanisms.

These character copying neural models have already been successfully applied to morphological inflection generation. This thesis studies this case and applies the copy mechanism to existing neural models of the latter tasks.

Acknowledgement

The awareness I have had of my good fortune in finding a supervisor in Dr. Tanja Samardžić has only grown in the past few months as I have worked on this thesis under her guidance. I want to thank Dr. Samardžić, first of all, for giving me the opportunity to work with her for my bachelor's thesis. I want to thank her for her invaluable time and her willingness, on occasions when I sat in her office with a seemingly impenetrable concept staring at me, to walk alongside me, step-by-step and line-by-line, towards understanding and comprehension. I want to thank her for the days when after waking up in the morning with little idea of where my work was going, an hour-long conversation with her left me feeling refreshed and back in control with a clear direction. It is only now, as I look back at the last few months in their entirety with the perspective which comes at the end, that I see some key moments which had either found me lost or doing something which was going nowhere. Your interventions at these junctures are what got me out, Tanja. It is only now, at the end of my bachelor's thesis, that I fully comprehend why we need thesis supervisors. I owe you this comprehension. Thank you.

I want to thank my co-supervisor, Dr. Yashvardhan Sharma, for his constant support and for ensuring through our conversations over the last few months that all was well and that this thesis moved, both in terms of the relevance of content and the distribution of time across the semester, in sync with the expectations of BITS Pilani.

I want to thank Tatyana Ruzsics for teaching me in great detail (and with greater patience) the working of the University of Zurich ScienceCloud. I want to thank her for walking me through some of the trickier aspects of her paper which inspires a large part of this thesis. And of course, I want to thank her for letting me reuse her code.

I want to thank Massimo Lusetti for helping me undertand some of the datasets used during this thesis, and for reorganizing them so that my life became easier.

I want to thank my parents for a myriad of things which the observant reader of this acknowledgement might (rightly) suspect I can fill a book with. However, sticking to the most relevant in the interest of space and brevity, I want to thank them for giving me unconstrained freedom in life to pursue what I wanted and to drop what I did not, even when some of the choices I made were not the most obvious and especially when some of them brought me far from them. I want to thank them for their unshakable faith that I would do something worthwhile with this freedom.

Finally, I want to thank my sister, Avni, for ensuring through her questions and problems that I stayed razor-sharp with high school mathematics over the past four years. I want to thank her for being the dependable go-to person whenever I look for engaging no-nonsense conversation, and for being, in general, one of the wisest people I know.

Contents

Abstract i

Acknowledgement ii

Contents iv

List of Figures vi

List of Tables vii

1 Introduction 1

1.1 Motivation . . . 1

1.2 Research Question . . . 1

1.3 Thesis Structure . . . 2

2 Background 4

2.1 Tasks . . . 4

2.1.1 Inflection Generation . . . 4

2.1.2 Segmentation . . . 5

2.1.3 Normalization . . . 5

2.2 Architectures and Mechanisms . . . 6

2.2.1 Encoder - Decoder Mechanisms . . . 7

2.2.1.1 The Bidirectional Encoder . . . 7

2.2.1.2 The Decoder . . . 8

2.2.1.3 Encoder-Decoder . . . 8

2.2.2 Attention Mechanisms . . . 9

2.2.2.1 Soft Attention . . . 10

2.2.2.2 Hard Monotonic Attention . . . 12

2.2.3 Copy Mechanism . . . 16

3 Reinflection 20

3.1 Data . . . 20

3.1.1 SIGMORPHON 2016 . . . 20

3.1.2 ConLL - SIGMORPHON 2017 21
3.1.3 UniMorph Schema 21
3.2 Experiments 22
3.2.1 Feature Representation for Inflection Generation 22
3.2.1.1 Aharoni and Goldberg [2017]'s Feature Representation . . 23
3.2.1.2 Makarov et al. [2017]'s Feature Representation 23
3.2.1.3 The Task - Capturing Feature Types 24
3.2.1.4 The Procedure - Capturing Feature Types 24
3.2.1.5 The Problem - Differences between the Schema and the 2017 data 24
3.2.1.6 The Original Solution 25
3.2.1.7 The Modified Solution 28
3.2.2 Experiental Settings for Inflection Generation 29
3.3 Results 29

4 Segmentation 33
4.1 Data 34
4.2 Experiments 34
4.3 Results 35

5 Normalization 38
5.1 Data 38
5.1.1 ArchiMob 39
5.1.2 WUS 39
5.2 Experiments 39
5.3 Results 39

6 Discussion and Conclusion 41

References 44

List of Figures

1 Bidirectional Encoder 7
2 Decoder 8
3 Encoder-Decoder Mechanism 9
4 Encoder-Decoder model with Soft Attention 10
5 Soft Attention Mechanism 11
6 Encoder-Decoder model with Copy Mechanism 17

List of Tables

3.1 Inflection Generation Accuracy Scores 29
4.1 Morphological Segmentation Accuracy Scores 35
5.1 Normalization Accuracy Scores 39

1 Introduction

1.1 Motivation

Natural languages are defined by Partee et al. [1990] as *the languages we speak and use naturally to communicate with each other* and formal languages as languages which *are usually designed by people for a clear, particular purpose.*

The significance of natural language cannot be overstated in it being the channel for human thought and expression in all its nuance and ambiguity. Then, as observed by Partee et al. [1990], one of the the most important uses linguists make of formal languages is to represent the meaning of natural language: to attempt to formalize the latter while at the same time trying to capture and preserve all the subtlety that makes it human.

That, in a nutshell, defines the motivation behind the tradition of natural language processing from Chomsky's generative grammars to the probabilistic and statistical models of the 90s to the neural models of the current decade.

This thesis follows this school of neural models for natural language processing, relying on encoder-decoder models, which were brought to the fore in NLP research by Sutskever et al. [2014], and which have since been applied with great success to a number of sequence-to-sequence tasks ranging from downstream tasks like machine translation and question-answering to upstream tasks like normalization.

The thesis studies a specific architecture built on a basic encoder-decoder model - the copy mechanism, as employed by Makarov et al. [2017] for morphological reinflection and by See et al. [2017] for summarization, and applies it to the tasks of morphological segmentation and normalization.

1.2 Research Question

Aronoff and Fudeman [2011] define morphology as the branch of linguistics that deals with words, their internal structure, and the way they are formed. This thesis addresses and studies, in varying capacities, three tasks related to morphological text processing: morphological inflection generation, morphological segmentation, and normalization.

Each of these three tasks has been recently modelled as a sequence-to-sequence learning task in the deep leaning framework. Such modeling highlights the fact that a connecting thread among these tasks is the significant overlap, in terms of low string edit distances, between the elements of the input and output sequences in all three cases.

During the course of this thesis, I studied a sequence-to-sequence model for morphological inflection generation which uses a mechanism which copies characters from the source to the target sequence [Makarov et al., 2017], hence exploiting the overlap between the source and the target sequences. Makarov et al. [2017]'s encoder with the copy mechanism is able to achieve state-of-the-art results on morphological reinflection.

Therefore, we have the observation that the tasks of inflection generation, segmentation, and normalization share the property of having a small edit distances between their respective input and output sequences. Further, we have a model which employs the copy mechanism - an architecture which appears to be well suited to sequence-to-sequence tasks with similar input-output sequences, which has been applied successfully to inflection generation. Keeping these observations in mind, the research question that this thesis attempts to answer is the following:

Can the use of the copy mechanism bring about an improvement in the performance of sequence-to-sequence models for morphological segmentation and normalization?

1.3 Thesis Structure

This thesis is split into 6 chapters.

The first chapter serves as the introduction, outlines the motivation of the thesis and poses the research question.

Chapter - 2 deals with the background of tasks and architectures. The first part of the chapter defines the three tasks that this thesis talks about - inflection generation, segmentation, and normalization. The second part describes the neural network architectures used to address these tasks.

Chapters - 3 deals with inflection generation. It describes the corpora used for the task and discusses the representation of morphosyntactic features for inflection generation. Finally, it studies the copy mechanism in the context of a practical application, from which it is adapted for the tasks of segmentation and normalization.

Chapters 4 and 5 deal with segmentation and normalization respectively. Both of these chapters describe the datasets used and the incorporation of the copy mechanism with existing attention based neural network models for these tasks.

Chapter - 6 discusses the results obtained from the experiments on inflection, segmentation, and normalization, and concludes the thesis.

2 Background

This chapter describes the tasks this thesis addresses and the neural architectures it employs to address these tasks.

2.1 Tasks

This sections defines and describes the three tasks this thesis deals with: morphological inflection generation, morphological segmentation, and normalization. Further, it touches on the intuition on the relationship these tasks share with each other from a sequence-to-sequence learning perspective. This idea is developed further in chapters 4 and 5 and is the underlying idea behind the research question addressed by the thesis.

2.1.1 Inflection Generation

Merriam Webster defines inflection as the change of form that words undergo to mark such distinctions as those of case, gender, number, tense, person, mood, or voice. For example, the English word *communicate*, if it is to be used for the third person, singular, in the present tense, undergoes a change to become *communicates*.

Intuitively, inflection can be thought of as an upstream task for generation related tasks. If generation models have at their disposal robust inflection generation models which can, given root words and morphosyntactic tags extracted from contextual information, generate dependable inflected forms, then the task of forming coherent sentences from a sequence of words can be made easier.

This intuition for the usefulness of infection as an upstream task varies, of course, from language to language, increasing sharply as we go across the linguistic spectrum from high to low resource and orthogonally, from morphologically simpler to richer languages.

Among my own languages, for instance, going from English to Hindi represents a move in both those directions. In English, for instance, verbs do not inflect in gender, whereas in Hindi, they do. For example, the English sentence *I am eating an apple* can be translated into two different Hindi sentences: मैं सेब खा रहा हूँ (*main seb kha rah**aa** hoon*) or मैं सेब खा रही हूँ (*main seb kha rah**ee** hoon*) depending on whether the speaker is male or female. Going from Hindi to Sanskrit is an even bigger jump both in both directions towards lower resource and higher morphological richness. For example, the word *girl*, in English, has two forms: *girl* and *girls* based on whether the grammatical number is singular or plural, whereas the word बालिका , which is Sanskrit for *girl*, can have as many as 24 inflected forms on the basis of 3 possibilities for grammatical number and 8 possibilities for grammatical case.

2.1.2 Segmentation

The purpose of morphological segmentation is to decompose words into smaller units, known as morphemes, which are typically taken to be the smallest meaning bearing units in language. [Cotterell et al., 2016b]

As pointed out in Ruzsics and Samardžić [2017], a distinction can be made between two definitions of the segmentation task, depending on what the output of segmentation is. The first one of these, surface segmentation, partitions the input string into a sequence of substrings each of which corresponds to a morpheme. The second one, *canonical segmentation* [Cotterell et al., 2016b], can be seen as going a step further, to convert the substrings obtained as a result of surface segmentation into their canonical forms. For instance, the surface segmentation of the word *aggregation* would be $aggregat|ion$, whereas its canonical segmentation would be $aggregate|ion$, with the letter *e* appended to the first morpheme output by surface segmentation to bring the morpheme *aggregate* to its canonical form.

A look at the example given above suggests that when looked at from the perspective of a potential sequence-to-sequence learning task, segmentation presents input-output pairs which are similar looking strings. In concrete terms, if we consider the canonical segmentation of *aggregation* to *aggregate ion*, with a space separating the two morphemes, then the edit distance between the input and output strings is 2.

2.1.3 Normalization

Text normalization is the task of mapping non-canonical language, typical of speech transcription and computer-mediated communication, to a standardized writing.

[Lusetti et al., 2018]

Normalization can be used to bring terms for which there do not exist rigid rules for writing into a standard form, like mapping *U.S.A* and *USA* to *USA*.

Another one of its uses can be to identify whether different words come from the same root, like *bring*, *bringing* and *brought*, which come from the root *bring*. In this scenario, normalization can be seen as the reverse process of inflection - the first task described in this section. While inflection uses morphosyntactic features to transform root words into different forms which might be well suited to different grammatical contexts, normalization can strip words found in the wild of this grammatical context specificity, so that only the root remains and so that words which are different forms of each other can be identified to be related to each other by virtue of their origin from the same root. This idea would have been of great importance before the advent of distributional methods, when words were treated as discrete units and when the only other option to establish relationships were graph based methods which explicitly tried to capture relations between words.

In this work, normalization is addressed in a context closer to the example of USA. Concretely, this thesis addresses the normalization of Swiss German, which does not have an orthographic standard, but is nevertheless often used in the written form in the German speaking part of Switzerland for the purposes of informal communication. Since rules for writing Swiss German do not exist, there can arise differences among how different people write the same word. It is in this respect that this thesis addresses normalization, unless otherwise stated.

2.2 Architectures and Mechanisms

Each of the three tasks outlined in the previous section has been addressed by architectures based on character-level encoder-decoder models.

A common skeletal architecture on which most of the mechanisms described and used during this thesis are built consists of an encoder which captures the essence of the input sequence, and a decoder which generates the output sequence by modeling a probability distribution over all the characters in the vocabulary.

This basic architecture is augmented with combinations of attention and copy mechanisms, and decisions are made regarding the pieces of information which are to be used, if any, on which the decoder conditions its output, to achieve the best possible results on the previously outlined tasks.

In keeping with this view of considering different parts of an attention based encoder-decoder architecture as building blocks which we aim to arrange in the most productive manner possible, the remainder of this section describes each of these components in detail, and then goes on to describe how they can be pieced together to work as cohesive and coherent units.

2.2.1 Encoder - Decoder Mechanisms

Encoder-decoder models, as the name suggests, consist of two parts. The encoder reads the input sequence element by element (character by character or word by word, for instance), and generates a vector representation which captures the input sequence. The decoder uses the vector representation generated by the encoder to output the target sequence, element by element. The following sections go into some detail regarding both these components.

2.2.1.1 The Bidirectional Encoder

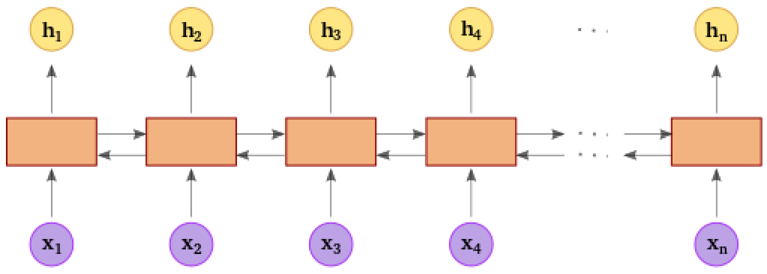

Figure 1: Bidirectional Encoder

In all encoder-decoder models used in this thesis, the encoder is a single-layer bidirectional LSTM.

At the higher level, as shown in Figure 1, the encoder takes in the input sequence of character embedding vectors X = (x_1, x_2, ..., x_n) and produces encoder hidden states h_i for each character in the input sequence.

At the lower level, each encoder cell gets, apart from the character embedding for that step, the encoder states of the previous and next cells in the sequence. The character embedding, along with the encoder state of the previous cell, is used by the current cell to obtain the forward encoder state $\vec{h_i}$, while the character embedding, along with the encoder state of the next cell, is used to obtain the backward encoder state $\overleftarrow{h_i}$.

The encoder is bidirectional in that each encoder state h_i is obtained by the concatenation of the forward and backward states, that is, $h_i = [\vec{h_i}; \overleftarrow{h_i}]$.

2.2.1.2 The Decoder

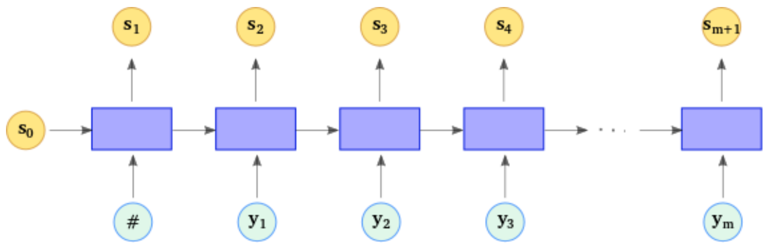

Figure 2: Decoder

In all encoder-decoder models used in this thesis, the decoder is a single-layer unidirectional LSTM.

The decoder receives one character embedding at each time step, apart from the hidden state from the previous timestep. At train time, the character embedding it receives corresponds to the previous character from the actual output (i.e., the character expected to be predicted in the previous timestep). This procedure of using the ground truth at each time step is termed *teacher forcing*. On the other hand, at test time (when there no correct output to compare against), the decoder receives the character embedding corresponding to the previous character predicted by the decoder itself.

The decoder (hidden) state output at timestep t is s_t. As Figure 2 shows, at the first timestep, a special <START> character is fed to the decoder (denoted by in the figure). The initial hidden decoder state, s_0, can be fed externally, which brings us to a discussion on how an encoder and a decoder work together.

2.2.1.3 Encoder-Decoder

As shown in Figure 3, one of the ways of feeding the encoded input sequence to the decoder is to initialize the decoder state with the encoding of the input, i.e., with the final output of the final state of the encoder.

Another way can be to initialize the decoder state randomly or in another such encoder-independent fashion, and then to condition the decoder on the final output of the encoder by feeding the latter into the decoder along with the embedding of

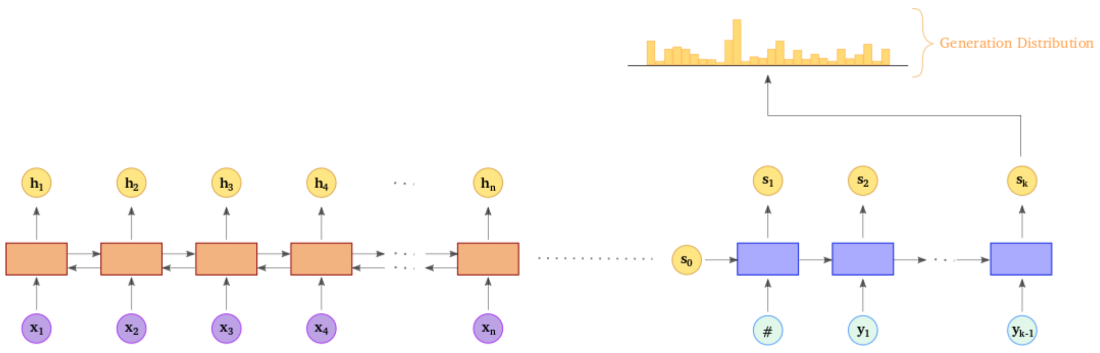

Figure 3: Encoder-Decoder

the previous character at each timestep. Taking this idea a little further brings us to attention mechanisms.

As described in section 2.2.1.2, the decoder state at a timestep t is s_t. This state is used to obtain the distribution over vocabulary terms which is used to determine the output for this timestep.

2.2.2 Attention Mechanisms

The previous section concluded with an illustration of decoders as conditioned generators conditioned, in that case, on the encoding of the entire input sequence. In practice, this means feeding the final output of the encoder to the decoder at each timestep. Attention mechanisms address the rigidity of such a model by allowing encoder outputs at different source side timesteps to be fed to the decoder depending on the current state of the decoder.

Therefore, attention mechanisms allow the decoder output to be conditioned on different parts of the input sequence depending on what the decoder is producing at any moment. The intuition behind why such an architecture captures the real world better, in some cases, as compared to the previously described architecture without attention, can be seen in the context of translation.

Given a sentence in the source language, feeding the decoder the entire encoded source sentence at each timestep translates to a human translator memorizing the source sentence in its entirety, understanding its meaning, and writing the target sentence word by word on the basis of this meaning present in her memory. In the real world, this process need not be this stringent. In practice, the human translator can look back at the source sentence repeatedly and focus on certain parts of it depending on where she finds herself in the partially completed target language sentence. This freedom is what attention mechanisms attempt to capture, not only in machine translation, but also in other sequence-to-sequence tasks.

In concrete terms, attention mechanisms combine information from the current decoder state with the information about different parts of the input sequence captured by the encoder states. This combination allows an attention mechanism to generate an attention distribution, which is used to identify the sections of the input sequence on which to focus. Attention mechanisms then take a weighted sum of the encoder states with weights coming from the attention distribution, to generate a *context* vector. This context vector can be used along with the current decoder state to generate the output, or it can be fed back to the decoder to condition its output on.

During this thesis, we use two types of attention mechanisms - soft attention and hard monotonic attention, which differ in how they generate the attention distribution and in the form of the attention distribution itself. The rest of this section delves deeper into each of these two mechanisms.

2.2.2.1 Soft Attention

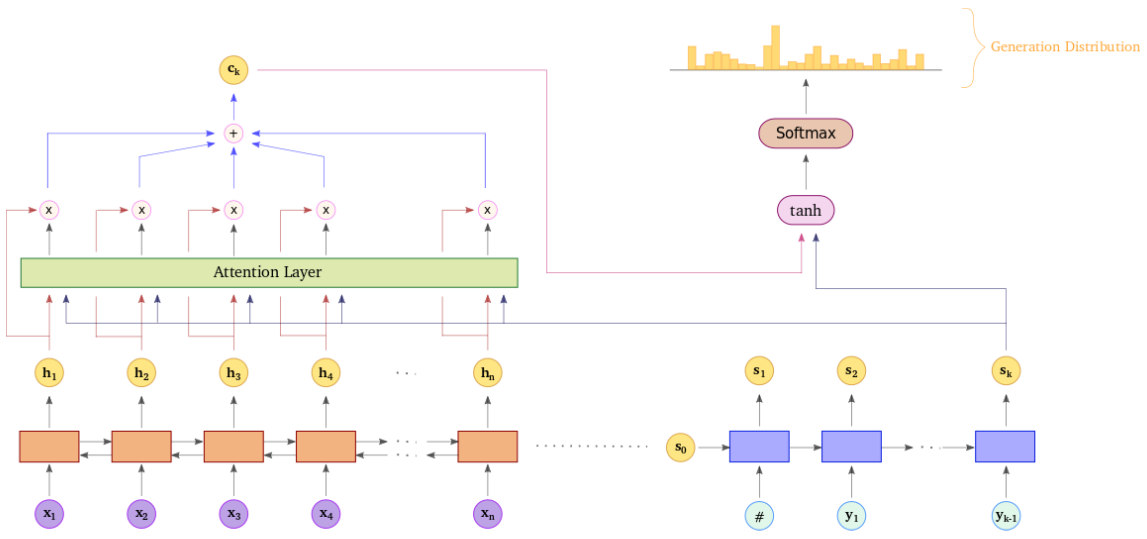

Figure 4: Encoder-Decoder with Soft Attention

In an an encoder-decoder model with soft attention, the attention can be distributed over multiple elements of the input sequence. Concretely, let the input character sequence by denoted by the vector $X = [x_1, x_2, ..., x_n]$, as shown in figure Figure 4, where n is the length of the input sequence. The attention layer uses the current decoder state with the encoder states to output the attention distribution. For the attention distribution represented by the vector $A = [a_1^k, a_2^k, ..., a_n^k]$, where a_i^k is the attention on x_i at the decoder side timestep k,

$$\forall i \in [1..n],\ a_i^k \in [0, 1] \tag{2.1}$$

$$\Sigma a_i^k = 1 \tag{2.2}$$

With a so defined attention distribution, the soft attention mechanism outputs a context vector which is a weighted sum of the encoder states with weights taken from the attention distribution. Different weights for different encoder states represent different amounts of attention on different parts of the source sentence.

This concludes a high-level view of a soft attention mechanism when integrated with an encoder-decoder setup. For a low-level view, refer to Figure 5, which shows a magnified view of the attention layer shown in Figure 4.

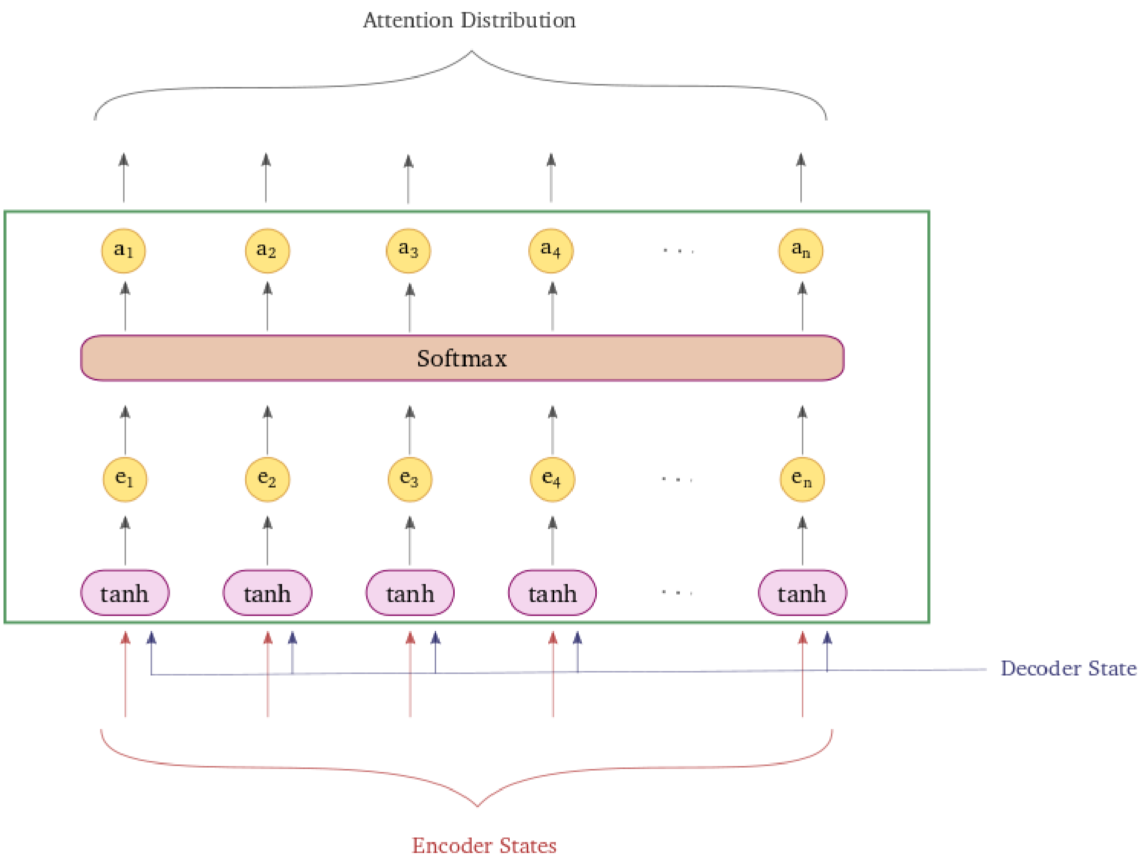

Figure 5: Attention Mechanism

As shown in Figure 5, the current decoder state s_k is concatenated with each of the encoder states h_i and passed through a tanh layer (or any other suitable function). The aim of this is to combine information regarding where the decoder currently is in the target sequence, along with information from all the parts of the input sequence, to help the mechanism decide on the sections of the input on which to focus. The values resulting from the tanh layer, represented by the vector $E^0 = [e_1^k, e_2^k, ..., e_n^k]$ (where, to reiterate, i is the index over the number of elements in the input sequence (n)), are passed through a softmax layer to obtain a probability distribution of the attention weights represented by the vector A as defined earlier.

As Figure 5 shows, the attention layer performs two steps - the first of mixing the information from the encoder and decoder sides, and the second of generating the distribution vector A from the vector of unnormalized weights, E. These steps are performed in accordance with the following equations: (according to the paper by See et al)

$$e_i^k = v^T tanh(\boldsymbol{W_h h_i} + \boldsymbol{W_s s_k} + b_{attn}) \tag{2.3}$$

$$\boldsymbol{a^k} = softmax(\boldsymbol{e^k}) \tag{2.4}$$

Once the attention distribution vector A is calculated using equation 2.4, it is used to calculate the context vector - a weighted sum of the encoder states which is supposed to be a combined representation of the encoder states with weights given to each state according to the magnitude of attention on the corresponding element of the input sequence.

The context vector c_k, as shown in figure Figure 2.2.2.1, is calculated using the following equation:

$$\boldsymbol{c_k} = \sum_{i=1}^{n} a_i \boldsymbol{h_i} \tag{2.5}$$

2.2.2.2 Hard Monotonic Attention

While soft attention distributes weights among the encoder states such that each weight can fall in the range $[0, 1]$ and such that the sum of all weights is 1, hard attention further restricts the first of these two conditions to focus on just one of the encoder states. In other words, while the context vector in the case of soft attention is obtained by a weighted sum of the encoder states, in hard attention, the context

[0]The superscript k refers to the k^{th} timestep on the decoder side, and is omitted wherever evident from the context, as in Figure 5

vector *is* one of the encoder states (which, in a rigorous sense, is also a weighted sum of encoder states, but with all but one weights set to 0).

Concretely, as done in the case of soft attention, let the input character sequence by denoted by the vector $X = [x_1, x_2, ..., x_n]$, where n is the length of the input sequence. Then, for a hard attention distribution represented by the vector $A = [a_1, a_2, ..., a_n]$, where a_i is the attention on x_i,

$$\exists i \in [1..n] \text{ such that } a_i = 1 \tag{2.6}$$

$$\forall j \in [1..n] - \{i\},\ a_j = 0 \tag{2.7}$$

A typical hard attention model replaces the deterministic method used in soft attention with a stochastic sampling method like Monte Carlo, but during the course of this thesis, we focus on a specific variation of the hard attention model - hard monotonic attention - proposed by Aharoni and Goldberg [2017] for morphological inflection generation.

Aharoni and Goldberg [2017]'s model applies the idea of a hard attention mechanism in the context of an encoder-decoder neural network to essentially implement a finite-state transducer as a component of the string transduction task of inflection generation. The remainder of this subsection walks through the aspects of this model.

1. **The Alignment Model**: The monotonic hard attention model relies on the external statistical character alignment model of Sudoh et al. [2013] to obtain an alignment from the input sequence (the lemma) to the output sequence (the inflected form). This character alignment model outputs three types of alignments:

   - 0-to-1 alignment: a character in the output sequence corresponds to none of the characters in the input sequence
   - 1-to-0 alignment: a character in the input sequence corresponds to none of the characters in the output sequence
   - 1-to-1 alignment: a character in the input sequence corresponds to one character in the output sequence

2. **Monotonicity of Alignment:** Aharoni's transducer is simplified based on the assumption that the alignment between the input sequence and the output sequence is monotonic. This means that for any two characters - x_i and x_j - in the input sequence, and the two characters from the output sequence - y_a

and y_b - to which x_i and x_j are respectively aligned, $a < b$ if and only if $i < j$.

3. **The STEP Action:** The monotonicity assumption entails that the attention pointer does not move backwards. Therefore, attention can stay on the current element, or move to the next element. Aharoni et al introduce a $STEP$ action to model the latter. (The reason why staying on the current element or moving to the next exhausts all possibilities is explained in the algorithm breakdown section)

4. **Redefinition of the Transduction Task:** Although morphological inflection generation is a sequence transduction task from a string (x) to another string (y), the Aharoni model replaces the search for y with a search for a sequence of step and write actions (s), where $\Sigma_s = \Sigma_y \cup STEP$. This allows for a convenient way to model the movement of the attention pointer.

   Even as the sequence s is being constructed at prediction time, it simultaneously performs the control step of deciding when the attention pointer (which governs which part of the encoded sequence would go ahead to be fed into the decoder which is outputting s) is going to be moved to the next character of x. This task - moving the attention pointer which (in part) governs its own construction - is one that could not have been done by y had it been chosen as the output sequence. This, hence, is the reason for introducing the intermediate sequence s, from which y can be obtained by selecting the non-step character outputs.

5. **The algorithm to obtain *s*:** Aharoni and Goldberg [2017]'s algorithm to obtain s is reproduced here, and is followed by a step-by-step walk-through of

the same.

Algorithm 1: Generates the oracle action sequence $s_{1:q}$ from the alignment between $x_{1:n}$ and $y_{1:m}$

Data: a, the list of either 1-to-1, 1-to-0 or 0-to-1 alignments between $x_{1:n}$ and $y_{1:m}$

```
1  Initialize s as an empty sequence;
2  for $a_i = (x_{a_i}, y_{a_i}) \in a$ do
3      if $a_i$ is a 1-to-0 alignment then
4          s.append(step);
5      else
6          s.append($y_{a_i}$);
7          if $a_{i+1}$ is not a 0-to-1 alignment then
8              s.append(step);
9      end
10 end
11 return $s$;
```

- The algorithm uses the output of the character alignment model referenced above. This output is a list of 1-to-0, 0-to-1, and 1-to-1 alignments between x and y.
- Line - 3: Outer *if* condition - In case of a 1-to-0 alignment, since there is no character on the output side of the hard alignment corresponding to the currently attended input character, the attention pointer is moved forward, i.e., the STEP action is appended to s.
- Line - 5: Outer *else* condition - Control enters the current else block in the case of a 0-to-1 or a 1-to-1 alignment, i.e., in the case where there exists a character on the output end of the current alignment. As given at line - 6, this character is written to s.
- Line- 7: Inner *if* - This is perhaps the most interesting section of this pseudocode. This is what ensures that for any 1-to-0 or 1-to-1 alignment (that is, for any alignment where there is a character on the input side), the character on the input end is read before
  - the corresponding character on the output end is written to s (in case of 1-to-1) or
  - the decision to move attention to the next input character is made (in case of 1-to-0)

  This is perhaps the right place to make explicit the observation that the attention pointer is moved to the next character in the input sequence *only* when a *STEP* is encountered on the control sequence *s*. Hence, it is possible (and in fact, happens often) that across two consecutive timesteps, the vector going from the encoder to the decoder remains unchanged at the encoding of the currently attended character. The only input to the decoder which would change between these two timesteps is the embedding of the previously predicted output (since the third input to the decoder - the feature vector, also remains unchanged).

Going back to the observation that the attention pointer does not move unless a *STEP* is encountered, consider the case where we have two consecutive 1-to-1 alignments. For ease of reference, let these be $x_1 - y_1$ and $x_2 - y_2$. If we dry run this case on the pseudocode, we see that when the first 1-to-1 alignment is encountered, control moves to the else statement and the output character y_1 is written to s. Now, if the inner *if* statement is absent, then the loop of the algorithm simply moves to the next 1-to-1 alignment, and the next output character, y_2, is written to s.

Next, consider what happens when this section of s is encountered at training time. Initially, attention would be at x_1 and the corresponding output character (assuming the model is predicting correctly at this point) would be y_1. The next element in s would be y_2. Now, since the only way attention moves forward is a *STEP*, and since that is absent here, at the next timestep, attention would remain at x_1 while on the output sequence we would have moved to y_2. Thus, the requirement that the input character corresponding to each alignment should be read before the corresponding output character is written to s would be violated.

2.2.3 Copy Mechanism

The copy mechanism can be useful, theoretically, to addresses sequence-to-sequence learning problems which have significant overlaps between the source and target sequences. It builds on attention mechanisms by reusing attention information to determine when to copy parts of the source sequence to the target sequence.

Just like a vanilla encoder-decoder and an encoder-decoder with attention, an ACM[0]

[0]**A note on terminology:** I borrow termnology from Makarov et al. [2017] to abbreviate encoder-decoder with hard attention and a copy mechanism to *HACM*. Further, I use the abbreviation *SACM* when referring soft attention based models, and the abbreviation *ACM* to refer to attention based models with copy mechanism generally. HACM, as described in this thesis, is suggested by Makarov et al. [2017], and SACM, as described, is suggested by See et al.

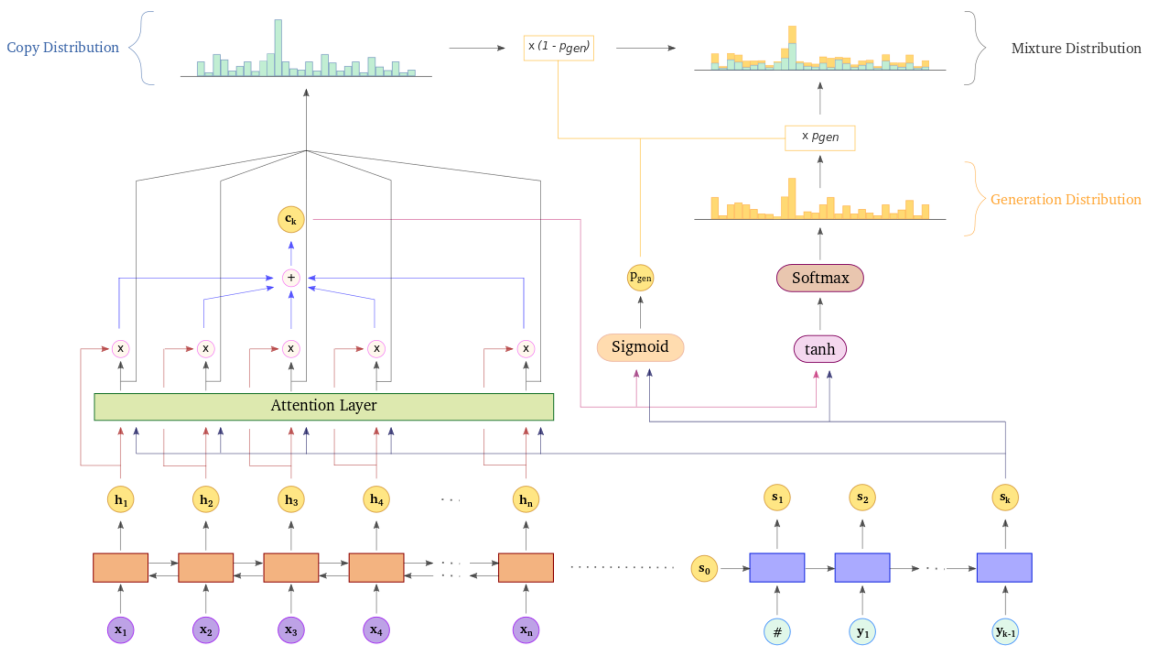

Figure 6: Encoder-Decoder with Soft Attention and Copy Mechanism

model generates a distribution over the vocabulary terms at each timestep and uses this distribution to decide the output for that timestep. The distribution generated by the model is called the *mixture distribution* [Makarov et al., 2017] since it relies on the *mixing* of information from two sources: a distribution which suggests the character to output if a new character is to be *generated*, and a distribution which suggests the character to output if a character from the input sequence is to *copied* over to the output. These two factors are weighed against each other by a mix-in parameter. The rest of this section deals individually with these three aspects of the copy mechanism - the generation distribution, the copy distribution, and the mix-in parameter.

1. **The generation distribution:** This is precisely the distribution which is generated by the attention mechanism, as described in section 2.2.2.

2. **The copy distribution:** This forms the core of the copy mechanism. The generation of the copy distribution can be seen, in a way, as converting the attention distribution, which is over the input sequence, to a distribution over the vocabulary, V.

   The procedure to perform this conversion differs slightly between HACM and

[2017]

SACM models, because they differ in the attention distributions they use.

- **HACM:** Recall from section 2.2.2.2 that in a hard attention model, attention is focused one element of the input sequence. The terminology from that section and equations 2.1 and 2.2 are reproduced here for convenience:

  The input character sequence is denoted by the vector $X = [x_1, x_2, ..., x_n]$, where n is the length of the input sequence. The hard attention distribution represented by the vector $A = [a_1, a_2, ..., a_n]$, where a_i is the attention on x_i,

  $$\exists i \in [1..n] \text{ such that } a_i = 1$$

  $$\forall j \in [1..n] - \{i\},\ a_j = 0$$

  The copy distribution over the vocabulary is defined such that it is 1 for the character with attention and 0 elsewhere.

  Concretely, let us denote the copy distribution by P_{copy}. Further, recall that the vocabulary is denoted by V. Then,

  $$P_{copy}(w) = 1 \text{ for } w = a_i \tag{2.8}$$

  $$P_{copy}(z) = 0 \text{ for } z \in V - \{w\} \tag{2.9}$$

- **SACM:** Recall from section 2.2.2.1 that in the case of an soft attention model, the attention can be distributed over multiple elements of the input sequence. The terminology from that section and equations 2.6 and 2.7 are reproduced here:

  For a soft attention distribution represented by the vector $A = [a_1, a_2, ..., a_n]$, where n is the length of the input sequence,

  $$\forall i \in [1..n],\ a_i \in [0, 1]$$

  $$\Sigma a_i = 1$$

  Getting the copy distribution from the attention distribution is slightly more involved in the case of soft attention. As done in the previous case, we denote the copy distribution by P_{copy} and the vocabulary by V.

$$P_{copy}(w) = \sum_{\substack{i=1 \\ w=a_i}}^{n} a_i \tag{2.10}$$

3. **The mix-in parameter:** The mix-in parameter at a given timestep k, w_k^{gen}, determines the relative weights to be given to the generation and distributions, therefore acting as a *soft switch* between generating a character or copying a character from the input sequence to the output sequence.

   It is calculated from the context vector at the current timestep (c_k) and the current decoder state (s_k), according to the following equation:

$$w_k^{gen} = \sigma(\mathbf{W_m} \cdot [\mathbf{c_k}; \mathbf{s_k}] + b_m) \tag{2.11}$$

3 Reinflection

To inflect a word is to change its form to mark such distinctions as those of case, gender, number, tense, person, mood, or voice. For example, the English word *play* can be inflected to *played* to mark a distinction in tense, with the inflected form denoting past tense.

This example illustrates the ingredients of the inflection task: the lemma (*play*), the inflected form (*played*), and a set of morphosyntactic features (*{TENSE=Past}*) which govern the conversion of the lemma to the inflected form.

The example also shows for one language and one lemma-inflection pair an observation which is true for a majority of cases in many languages: the high overlap and low edit distance between the lemma and the inflected form which is believed to be the reason for the performance of the copy mechanism (section 2.2.3) on the inflection generation task.

3.1 Data

Two datasets have been used for performing experiments on morphological reinflection:

1. The SIGMORPHON 2016 dataset
2. The ConLL - SIGMORPHON 2017 dataset
3. UniMorph Schema

3.1.1 SIGMORPHON 2016

The dataset released for the SIGMORPHON 2016 shared task [Cotterell et al., 2016a] consists of inflection data from 12 languages, with training examples in each language consisting of (lemma, features, inflected form) triples.

As noted in the task paper [Cotterell et al., 2016a], the lemma and inflected word forms are orthographic strings in the native scripts (with the exception of Arabic, which is present in romanized form).

For each example, features are represented as a list of *(feature type, feature value)* pairs, where examples of feature types can be person, number, part-of-speech, and tense; and examples of feature values can be noun, verb, and adjective for the part-of-speech feature type; past and present for tense, etc.

A typical training example from the SIGMORPHON 2016 data, therefore, looks like the following one (taken from the German training set):

abenteuerlich   pos=ADJ,case=NOM,gen=MASC,num=SG   abenteuerlicher

3.1.2 ConLL - SIGMORPHON 2017

The dataset released for the SIGMORPHON 2017 shared task [Cotterell et al., 2017] consists of inflection data from 52 languages, with training examples in each language consisting of (lemma, inflected form, features) triples. As is the case for the SIGMORPHON 2016 data, the SIGMORPHON 2017 data also presents each training example as a (lemma, features, inflected form) triplet.

The SIGMORPHON 2017 data differs from the SIGMORPHON 2017 data in its representation of features. Unlike the 2016 data, which collates feature bundles into feature type = feature value lists as described in the previous subsection, the 2017 data drops the feature types. A feature bundle in the 2017 data, therefore, is a list of feature types.

A typical training example from the SIGMORPHON 2017 data, therefore, looks like the following one (taken from the Portuguese training set):

aceder   acedemos   V;1;PL;IND;PST;PFV

3.1.3 UniMorph Schema

The Universal Morphological Feature Schema [Sylak-Glassman et al., 2015] is a set of morphological features bucketed into a set of dimensions (feature types) which aims to allow definition of inflected words in terms of their roots or stems and a set of inflectional morphemes which can be mapped to the Schema.

Both the SIGMORPHON datasets define their feature bundles in terms of the features of the UniMorph Schema. While SIGMORPHON 2016 defines feature bundles in terms of feature type - feature value pairs, SIGMORPHON 2017 does so only in terms of the feature types.

The UniMorph Schema has been used in this thesis to generate a mapping from the feature types used in the 2017 dataset to their corresponding feature types.

3.2 Experiments

The primary aim of my experiments with inflection generation was to familiarize myself with the vocabulary of the domain and with some of the architectures used to address problems in the domain through existing models and resources. I realized during this study that the two models I studied about, both of which share the same core architecture, differed in their representation of the feature bundles which guide reinflection. As a secondary aim, I wanted to evaluate how these differences in feature representation affect model performance.

In line with this distinction between two different but overlapping aims, this section is also organized into two subsections, with section 3.2.1 dealing with lower level experiments with feature representations, and section 3.2.2 dealing with higher level experiments with the copy mechanism and other architectural choices.

For my experiments with inflection generation, I relied on the source code for Aharoni and Goldberg [2017] by Roee Aharoni and that for Makarov et al. [2017] by Tatyana Ruzsics instead of setting up a framework from scratch.

3.2.1 Feature Representation for Inflection Generation

Aharoni and Goldberg [2017] use the SIGMORPHON 2016 data for their hard attention model whereas Makarov et al. [2017] use SIGMORPHON 2017. Though the latter build their model on top of that introduced by the former, their approaches differ in places partly due to the the differences in the structures of the data they use.

As described in section 3.1, the most evident difference between the 2016 and 2017 data is the way feature bundles are represented. In accordance with their respective datasets, therefore, Aharoni and Goldberg [2017] differ from Makarov et al. [2017] in the way they encode features. Aharoni and Goldberg [2017] encode their features in the form of feature type - feature value pairs; that is, in the form they are given in the 2016 dataset. Makarov et al. [2017], on the other hand, encode their features as lists of feature values, in accordance with the way features are given in the 2017 dataset, since information about feature types is absent from it. The following subsections describe in some detail the procedures used by the two to generate feature vectors.

3.2.1.1 Aharoni and Goldberg [2017]'s Feature Representation

In Aharoni and Goldberg [2017]'s hard attention model, there exists a unique embedding for each feature type - feature value pair. Further, the SIGMORPHON 2016 data is designed such that in any example, there does not exist more than one feature value for a given feature type (it is possible for there to be no value assigned to a given feature type in an example). Hence, to generate the feature vector for an example, the following procedure is followed:

1. Iterate over all feature types relevant to the grammar of the given language
2. During a specific iteration, check whether the current feature type has been assigned a value in the current example
3. If yes, append the embedding corresponding to this feature type - feature value pair to the feature vector; else, append an *unknown* feature embedding to the feature vector.

3.2.1.2 Makarov et al. [2017]'s Feature Representation

In Makarov et al. [2017]'s model, on the other hand, there exists a unique embedding for each feature value. This model groups all feature values together, irrespective of the types they belong to, and generates a feature vector on the basis of the presence or absence of each of the feature values present in this group. The procedure it uses to generate the feature vector for a specific example can be split into the following steps:

1. Iterate over all feature values relevant to the given language
2. During a specific iteration, check whether the current feature value is present in the list of feature values for that example
3. If yes, append the embedding corresponding to this feature value to the feature vector; else, append an *unknown* feature embedding to the feature vector.

3.2.1.3 The Task - Capturing Feature Types

Given this difference in the feature representations of the two models I studied and the fact that the 2017 dataset, which is the more comprehensive of the two datasets, omits feature types, I tried to generate feature type information for the 2017 dataset, and then to run Aharoni and Goldberg [2017]'s model on it to see if capturing this information improved on a model which used only the feature values. The remainder of this section describes the procedure used to capture feature type information for the 2017 data.

3.2.1.4 The Procedure - Capturing Feature Types

The ConLL - SIGMORPHON 2017 Shared Task paper Cotterell et al. [2017] mentions that the features in the 2017 shared task dataset are encoded using the UniMorph Schema. However, where the UniMorph Schema lists both feature types and their feature values, the 2017 dataset does away with the former. Therefore, to incorporate feature types into the 2017 dataset, I used the UniMorph schema to create a dictionary mapping feature values back to feature types. Ideally, this should have been a fairly straightforward process, but it was complicated by the fact that the current public list of feature types (called dimensions in the Schema) and feature values (called features in the Schema) differ in places from the feature value lists of the 2017 dataset. Much of this section of the thesis deals with the reconciliation of these differences.

3.2.1.5 The Problem - Differences between the Schema and the 2017 data

I identified 3 types of feature values present in the ConLL 2017 dataset which could not be mapped directly to their corresponding feature types using the UniMorph Schema. These are listed below:

1. Feature values present in the 2017 data which are not present in the UniMorph dictionary.

   These are the feature values present in the 2017 dataset which are absent from the UniMorph Schema list.

   Examples: HYP, FREQ, AGT, CONN, STRONG, etc.

2. Pairs of feature values, where both elements of the pair belong to the same feature type, and are both present in a single example.

Implicit in the structure of the 2016 dataset is the fact that in any given example, a feature type can be assigned at most one feature value. There exist examples in the 2017 dataset where this assumption has been relaxed.

Example: V.PTCP (Participle) and V (Verb) both come under the part-of-speech feature type in the UniMorph Schema, and there exist examples in the 2017 dataset where both these values are present in the same feature bundle.

3. Features values listed in the form of combinations

   In the ConLL 2017 dataset, there exist feature values which are listed as a combination of 2 or more feature values separated by one of the three symbols: +, :, /

   Examples: AT+ESS, 2:PL, MASC/FEM

3.2.1.6 The Original Solution

This section describes the proposed solutions to each of the three problems listed in the previous section, in that order.

1. Feature values present in the 2017 data which are not present in the UniMorph dictionary.

   The feature values not present in the UniMorph dictionary have been split into the following three categories and each missing feature value has been handled according to the category it falls in:

   a) Features values which appeared to belong to classes of feature values existing in the Schema or which appeared to be alternate spellings of existing feature values.

      In such cases, the missing feature values have been assigned to the feature type of the feature values which are judged similar to them and are present in the Schema.

      Example - 1: The feature values LGSPEC3 to LGSPEC9 are present in the 2017 data while they are absent from the UniMorph Schema list. However, LGSPEC1 and LGSPEC2 are mentioned in the Schema. Clearly, LGSPEC3 to LGSPEC9 also belong to the same class of features (language specific) as the latter. Therefore, they have all been assigned to the feature type of LGSPEC1 and LGSPEC2, i.e., LanguageSpecificFeatures.

Example - 2: The feature value CAUSV is present in the 2017 dataset while it is absent from the Schema list. However, the feature value CAUS (Causative Valency) is present in the schema. Judging it to be a alternate spelling, CAUSV has been assigned the feature type of CAUS, i.e., Valency.

b) Feature values which appear to be misspellings of feature values existing in the Schema.

   In such cases, the misspelling is identified, but while creating the feature dictionary for that example, the correct spelling, along with the feature type of the correct feature value, is entered.

   Example: LgSPEC8 is present as a feature value and is identified as a feature different from LGSPEC8 while clearly both refer to the same feature value. Here, the feature value is assigned the feature type of LGSPEC8, i.e., LanguageSpecificFeatures and also, the feature value LGSPEC8 is fed in the dictionary instead of LgSPEC8.

c) Feature values not falling into either of the two aforementioned categories.

   In such cases, an attempt has been made to assign a feature type to each value using the following method, illustrated with the help of a hypothetical example:

   - Given an unidentified feature value *f*, it is located in the data, i.e., in the language, and the specific example (line number) within that language's file, where it occurs.
   - Feature values to the immediate left and right, say *f1* and *f2*, are noted, along with their corresponding feature types, *F1* and *F2*.
   - The same file is searched for another example with a single identified feature sandwiched between features of type *F1* and *F2*.
   - If such an identified feature is present in the file, then the feature type of the identified feature is assigned to the unidentified feature, *f*.
   - If no such identified feature is present, the unidentified feature (*f*) is ignored.

2. Pairs of feature values, where both elements of the pair belong to the same feature type, and are both present in a single example.

If the feature bundle of an example contains two feature values *f1* and *f2* which both belong to the feature type *F*, then both the pairs *F* = *f1* and *F*=*f2* are added to the feature dictionary, so that both these pairs get their individual embedding. However, while appending the embedding corresponding to *F* to the complete feature vector, both these embeddings are added.

3. Features values listed in the form of combinations.

   As mentioned in section 3.2.1.5, there three symbols are used in the 2017 data to combine feature values: +, :, /. These have been handled in two different ways on the basis of the relationship between the feature values they combine:

   - Colon (:)

     It is observed that the instances of feature values combined by the colon are such that the feature values belong to different feature types. Such combinations are treated like two different feature values belonging to two different feature types. In other words, any two feature values separated by a colon are treated as they would have been had they been separated by a semicolon.

     Example: In the 2017 dataset, one of the combinations which uses a colon is 1:SG, where the value 1 belongs to the feature *Person*, and the value SG belongs to the feature *Number*. Hence, any example with 1:SG in the feature bundle gets *Person* = 1 and *Number* = SG assingned to its feature set.

   - Plus (+) and Forward slash(/)

     It is observed that the instances of feature values combined by plus and forward slash are such that the feature values belong to the same feature type. Both of these types of combinations have been treated in the same way: for each feature value present in the combination, a feature type = feature value pair has been added in the feature alphabet, but at the time of encoding that feature type in the feature vector, the embeddings of all feature type = feature value pairs are added.

     Example: In the dataset, one of the combinations separated with a forward slash is SG/DU/PL, where each of the three feature values corresponds to the feature type *Number*. When an example with this combination is encountered, then all three of *Number* = SG, *Number* = DU, *Number* = PL, are added to the feature set of this example. Also, while the feature alphabet is being constructed using all *Feature type* = *Feature* *value* pairs from all the examples of a given language, each of these three pairs are added to the alphabet. However, in the feature encoding for any one example, there can be just one feature embedding corresponding to each feature type. Hence, in this case, the three feature embeddings - one from each of the three pairs listed above, are added to obtain an embedding of the combination, which has dimensions equal to the dimensions of a standard feature embedding.

3.2.1.7 The Modified Solution

The solution described in section 3.2.1.6 aims to minimize the number of feature values which remain unassigned to some feature type. Due to this fact, it leaves some room for error in the assignment of feature types to feature values. In other words, it is possible for some of the techniques suggested in section 3.2.1.6 to misclassify a feature value to the incorrect feature type.

To ensure that this does not have unintended consequences, another set of more rigid criteria has also been used to perform the assignments, and the performances of both the resulting models have been recorded independently.

This section describes the changes that the modified solution undergoes as compared to the original one.

A change was made to the third category of missing features in section 3.2.1.6, i.e., those which did not fall into the first two categories (1(c)). Instead of attempting to indirectly determine the feature type of the values in the third category, all such values have been assigned to an *Unknown* feature.

A change was also made to the features falling in the first category of section 3.2.1.6 - apparent alternate spellings (1(a)), in that they have also been assigned to an *Unknown* feature.

Example: Previously, the feature value $CAUSV$ (which is not present in the UniMorph Schema), was judged to be an alternative spelling of CAUS (Causative Valency) and had been assigned to the feature type *Valency* on the basis of this apparent relation. But since it cannot be claimed with complete certainty that $CAUSV$ cannot represent some other unidentified feature value, it has also been placed within the *Unknown* feature type.

3.2.2 Experiental Settings for Inflection Generation

The inflection generation experiment has been conducted under four settings. The first three use a hard attention mechanism and differ in their feature representations. The fourth uses a copy mechanism. Among the first three settings (which do not use a copy mechanism), the first two incorporate feature types in the feature representation using the procedures described in sections 3.2.1.6 and 3.2.1.7 respectively. The third setting does not use feature types, hence following Makarov et al. [2017] in their representation. Finally, the fourth setting employs the copy mechanism on top of the third setting.

For convenience, the four settings described here are labelled as follows:

- Model 1.1: With feature types (Original)
- Model 1.2: With feature types (Modified)
- Model 2.0: Without feature types
- Model 3.0: With Copy Mechanism

For all four settings, experiments are run on the 52 languages of the ConLL - SIGMORPHON 2017 dataset and results have been reported for a majority voting ensemble of 5 models for each setting-language pair. All experiments have been run on the medium setting of the dataset, i.e., with 1000 training examples.

3.3 Results

Table 3.1: Inflection Generation Accuracy Scores

Model / Lang	Model 1.1		Model 1.2		Model 2.0		Model 3.0	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test
albanian	82.40	81.40	81.90	82.30	81.40	81.00	85.20	86.40
arabic	72.00	72.40	72.60	74.10	72.90	73.30	80.40	77.20
armenian	90.10	88.60	89.90	89.30	89.90	88.40	91.70	89.70
basque	71.00	74.00	70.00	76.00	74.00	80.00	72.00	75.00
bengali	93.00	94.00	94.00	94.00	94.00	94.00	97.00	99.00
bulgarian	78.20	77.70	78.20	77.70	78.80	76.70	83.30	81.00
catalan	88.60	88.20	88.60	88.20	87.90	87.40	92.00	92.00
czech	82.30	81.90	82.30	81.90	82.00	81.40	86.10	85.20

danish	82.70	80.80	82.70	80.80	82.70	80.50	83.70	82.50
dutch	78.60	77.50	78.60	77.50	80.00	78.20	80.50	82.70
english	89.40	90.00	89.40	90.00	90.80	89.90	92.30	92.70
estonian	74.80	75.30	74.80	75.30	76.30	75.30	77.10	79.40
faroese	61.00	61.80	61.00	61.80	62.50	63.10	66.10	65.90
finnish	66.20	65.60	65.70	65.10	66.20	66.80	73.90	73.70
french	76.50	72.50	76.50	72.50	77.40	74.40	81.60	78.50
georgian	91.80	92.90	92.30	93.00	92.30	92.30	93.60	93.40
german	76.80	75.60	76.80	75.60	76.30	74.90	80.50	79.30
haida	88.00	90.00	88.00	90.00	87.00	87.00	88.00	87.00
hebrew	80.10	78.80	79.10	79.90	79.90	77.90	81.20	81.40
hindi	93.30	93.30	93.30	93.30	93.90	91.60	97.30	96.80
hungarian	70.60	69.60	71.50	68.70	72.20	70.00	71.50	70.90
icelandic	66.00	67.20	66.00	67.20	66.20	66.90	71.40	72.50
irish	68.40	66.80	67.20	65.50	67.20	66.40	71.50	71.90
italian	91.20	91.10	91.20	91.10	92.30	90.40	93.10	91.50
khaling	78.80	75.10	78.80	75.10	78.40	74.70	76.90	74.70
kurmanji	89.40	91.50	89.40	91.50	90.10	90.60	91.30	91.60
latin	43.70	42.60	43.70	42.60	41.80	42.10	47.90	47.60
latvian	81.20	83.20	85.80	87.30	86.10	86.50	88.80	88.20
lithuanian	52.10	51.20	57.20	56.30	57.00	56.60	58.90	59.20
lower-sorbian	79.00	80.70	79.00	80.70	79.20	80.20	82.20	82.30
macedonian	87.30	87.00	87.30	87.00	87.80	88.10	89.90	90.60
navajo	45.30	45.40	45.30	45.40	43.40	45.70	47.20	47.80
northern-sami	58.60	57.50	58.60	57.50	59.60	56.40	67.90	67.30
norwegian-bokmal	82.50	82.90	82.20	83.00	83.00	83.10	83.10	83.30
norwegian-nynorsk	65.20	61.50	66.80	60.20	67.20	61.40	68.30	63.60
persian	86.50	84.10	86.50	84.10	85.60	83.30	87.00	84.60
polish	77.40	77.70	77.40	77.70	76.70	75.40	81.00	80.60
portuguese	92.80	93.90	92.80	93.90	93.00	93.30	94.40	94.90
quechua	96.60	94.80	96.60	94.90	95.30	93.30	95.90	95.30
romanian	75.40	73.50	75.40	73.50	76.40	73.40	79.10	77.20
russian	80.20	80.50	80.20	80.50	80.20	80.20	84.90	83.80
scottish-gaelic	94.00	82.00	94.00	82.00	94.00	82.00	96.00	90.00
serbo-croatian	79.70	76.50	79.70	76.50	80.00	76.30	83.60	82.10
slovak	74.80	74.60	74.80	74.60	71.60	72.00	79.70	79.50
slovene	87.60	87.20	87.60	87.20	86.40	86.90	87.70	88.10

sorani	67.90	65.90	67.90	65.90	68.20	66.20	68.00	67.90
spanish	89.20	89.00	89.20	89.00	88.30	88.80	90.10	89.40
swedish	77.40	76.00	77.40	76.00	77.20	77.70	77.50	78.50
turkish	79.50	79.00	84.10	83.20	81.30	81.80	85.10	84.50
ukrainian	75.40	77.60	75.40	77.60	74.90	76.70	80.80	81.00
urdu	92.80	93.70	92.80	93.70	92.80	93.40	96.70	96.70
welsh	83.00	86.00	83.00	86.00	78.00	90.00	84.00	88.00
Average	78.58	78.03	78.86	78.34	78.80	78.15	81.63	81.23

Header Legend: Model - 1.1: With Feature Types (Original); Model - 1.2: With Feature Types (Modified); Model - 2.0: Without Feature Types; Model - 3.0: With Copy Mechanism

---

Table 3.1 shows that the copy mechanism achieves significant improvement of ~3% from the simple soft-attention models, which do not vary greatly among themselves in terms of performance. Model 1.2, which is the better performing of the two settings which take feature types into account, performs only slightly better than the Model 2.0, which does not use feature types, with a difference which might be explained by experimental error.

The explanation for this might lie in the observation that it might be possible for Model 2.0 to capture feature type information implicitly. Recall from sections 3.2.1.1 and 3.2.1.2 that Aharoni and Goldberg [2017]'s model, which takes feature types into account, generates a feature vector by concatenating embeddings for feature=value pairs. Makarov et al. [2017]'s model, which does not take feature types into account, generates the feature vector by going over all possible feature values, and appending the embedding for a specific feature value if it is present in the example, and the embedding for an *unknown* value otherwise.

Consider how the embeddings corresponding to a specific feature type are represented in both these cases. Concretely, we consider a feature type F with possible feature values f_1, f_2, ..., f_n.

In case of Models 1.1 and 1.2, there exists an embedding corresponding to each pair $F = f_i$, and there is a section of the feature vector for the feature type F where one of these n embeddings is inserted. In case of Model 2.0, there does not explicitly exist a section in the feature vector corresponding to F. There are, however, sections of the feature vector corresponding to each of the feature values (f_i). Without loss of generalization, we can assume that all these sections occur in the feature vector in a continuous sequence. This allows us to define a section corresponding to F. In this section, in most cases, there will be $n-1$ embeddings corresponding to the *unknown* value, while there will be 1 embedding corresponding to the f_i value which is present in that example. There are cases in the 2017 dataset where more than one of the f_i values might be present in the feature vector, but these cases are exceptions.

Considering the sections to feature types as they have now been defined for Model 2.0, it can be seen how it might become possible for the model to learn about feature types not by means of explicitly defined feature type=feature value embeddings, but by implicitly noting that only one of n feature values gets a non *unknown* embedding per example. The model would therefore learn a relation between these n feature values, which would essentially allow it to capture information about feature types which we explicitly fed in the preceding two models. This can be a possible explanation for the similar performance of Models 1.1, 1.2, and 2.0.

4 Segmentation

Morphological segmentation decomposes words into morphemes. Recall from section 2.1.2 the distinction between surface and canonical segmentation. To reiterate, surface segmentation partitions the input string into a sequence of substrings each of which corresponds to a morpheme, while canonical segmentation [Cotterell et al., 2016b] further converts the substrings obtained as a result of surface segmentation into their canonical forms. For instance, the surface segmentation of the word *aggregation* would be $aggregat|ion$, whereas its canonical segmentation would be $aggregate|ion$, with the letter *e* appended to the first morpheme output by surface segmentation to bring the morpheme *aggregate* to its canonical form.

Since the aim of this part of the thesis is to apply to segmentation the copy mechanism, which resulted in improved performance on a model for inflection generation, we delve into some of the similarities and difference between these two tasks in an attempt to justify the motivation behind applying the copy mechanism to segmentation.

First, the example of segmenting the word *aggregation* serves to highlight the difference that unlike in the case of inflection generation, there is no set of morphosyntactic features forming a bridge between the source and the target words in the case of segmentation.

Among similarities, the example illustrates that like in the case of inflection generation, segmentation also deals with significant overlap between the source and the target sequences with an edit distance of 2 between the two strings. Finally, since we are dealing only with canonical segmentation during the course of this thesis, another similarity between the two tasks is that like inflection, segmentation also requires new characters (which are not in the input sequence) to be generated.

These two observations of similarity give an intuition on why a mechanism which has the capability to generate new characters and to copy existing characters from the input sequence to the output sequence may work well for segmentation too.

4.1 Data

The dataset for segmentation comes from Cotterell et al. [2016c], and consists of data from three languages: English, German, and Indonesian.

The data for each language consists of 10,000 example words with their canonical segmentation, and experiments have been performed on 10 splits each with 8,000 training, 1,000 development, and 1,000 test examples each.

4.2 Experiments

The primary motive of my experiments with segmentation was to incorporate the copy mechanism into an existing character level encoder-decoder model with soft attention. To do this, I relied on the following models to understand how the mechanism worked in different contexts, since the problem I aimed to address shared some similarities with each of them:

1. **The pointer-generator network of See et al. [2017]**
   This model apples a copying mechanism to text summarization. It builds a word-level encoder-decoder model with soft attention, and uses this soft attention distribution to build the copy distribution.

   This was relevant for me since I integrated the copy mechanism into a soft-attention model for segmentation. Therefore, I employ the procedure used by See et al to generate the copy distribution from the attention distribution.

2. **The copy mechanism of Makarov et al. [2017]**
   This model applies a copying mechanism to morphological inflection generation. It builds a character-level encoder-decoder model with hard monotonic attention (section 2.2.2.2), and uses this hard attention distribution to build the copy distribution.

   This was relevant for me since I integrated the copy mechanism into a character level model for segmentation. Therefore, all aspects of my implementation, apart from that which deals with the part of the model between the attention layer and the copy distribution, borrow from this model.

I reused the implementation of a character-level encoder-decoder with soft attention developed by Tatyana Ruzsics to perform my experiment with the copy mechanism. Following the code design paradigm used by Ruzsics in her code base, I extended a class corresponding to the soft attention model to create a class corresponding to a soft attention model with a copy mechanism.

Recall from section 2.2.3 that an ACM model differs from a traditional attention model in the former's use of a copy distribution and a mix-in parameter. The copy distribution is obtained by simply redirecting information generated by the attention mechanism along new paths, i.e., using the attention distribution calculated by the attention mechanism to generate a copy distribution over the vocabulary, as described in section 2.2.3. Since I combine the copy mechanism with a soft attention model, I use 2.10 to obtain the copy distribution from the attention distribution. The mix-in parameter is calculated using equation 2.11. Note that the only new model parameters introduced by the copy mechanism come from the calculation of the mix-in parameter. These are W_m and b_m from equation 2.11.

We have 3 languages in the dataset and 10 splits for each language. For each language-split pair, 5 models have been trained with different seeds. All models have been trained with input (embedding) dimension 100 and hidden layer dimension 200 for 30 epochs. The optimization method used is stochastic gradient descent (SGD).

4.3 Results

Table 4.1 shows the results obtained for experiments with segmentation. There are 4 parts to the table:

1. Results obtained for language-split pairs for one random seed without copy
2. Results obtained for language-split pairs for ensembles of 5 random seeds without copy.
3. Results obtained for language-split pairs for one random seed with copy
4. Results obtained for language-split pairs for ensembles of 5 random seeds with copy.

Table 4.1: Morphological Segmentation Accuracy Scores

Language	English		German		Indonesian	
Without copy						
Split Number	Dev	Test	Dev	Test	Dev	Test
0	78.30	76.20	72.40	73.00	93.70	92.70
1	77.50	75.20	74.90	74.90	94.50	94.00

2	77.90	75.60	76.30	74.60	94.80	92.60
3	75.80	73.80	75.20	75.90	95.00	92.60
4	77.20	76.20	73.90	73.70	93.40	94.50
5	75.80	77.80	76.60	76.10	94.60	93.50
6	78.00	76.90	77.50	75.70	94.30	94.20
7	76.10	78.00	77.20	77.10	93.30	94.20
8	78.20	71.60	74.60	75.40	95.00	93.40
9	75.10	74.10	78.00	74.80	94.30	92.90
Average	76.99	75.54	75.66	75.12	94.29	93.46
Without copy (Ensemble)						
Split Number	Dev	Test	Dev	Test	Dev	Test
0	81.30	79.30	77.30	79.10	94.20	93.80
1	81.60	78.90	78.50	80.50	94.90	94.90
2	80.90	78.50	79.80	79.60	95.60	94.40
3	80.10	79.60	78.40	79.30	95.70	93.40
4	80.50	81.10	79.70	81.30	93.50	95.10
5	78.50	79.70	79.20	80.70	95.20	94.20
6	80.80	80.00	80.60	79.50	95.10	94.60
7	79.90	82.00	81.40	82.00	94.30	95.10
8	81.50	76.70	78.80	78.30	95.50	94.60
9	80.10	77.80	82.30	79.00	94.30	94.40
Average	80.52	**79.36**	79.60	**79.93**	94.83	**94.45**
With copy						
Split Number	Dev	Test	Dev	Test	Dev	Test
0	77.20	74.30	74.50	75.40	93.30	93.10
1	78.30	74.30	75.20	78.20	94.20	94.60
2	78.50	75.80	77.00	73.50	95.10	94.30
3	77.50	75.60	73.30	72.90	94.90	93.30
4	78.30	75.90	73.30	72.50	92.90	93.90
5	76.70	78.10	76.10	76.20	94.00	93.50
6	78.80	75.70	75.20	69.30	93.80	94.40
7	77.70	78.20	77.20	77.20	94.10	94.60
8	78.40	74.00	74.10	71.30	94.80	93.60
9	78.40	77.40	78.00	75.80	93.90	93.50
Average	77.98	75.93	75.39	74.23	94.10	93.88
With copy (Ensemble)						

Split Number	Dev	Test	Dev	Test	Dev	Test
0	81.40	79.20	78.80	78.10	94.30	93.80
1	81.20	77.50	78.50	80.30	94.60	95.20
2	81.40	78.70	79.50	78.40	96.10	94.30
3	80.80	79.10	79.80	80.10	95.40	93.50
4	80.50	80.40	78.70	78.90	93.30	95.30
5	78.10	80.00	79.60	80.30	94.50	94.40
6	81.20	80.30	81.70	78.80	94.50	95.30
7	80.30	81.80	80.60	80.90	94.40	95.80
8	81.80	76.90	79.80	78.80	95.80	95.20
9	80.30	78.80	81.70	80.20	94.60	93.90
Average	80.70	**79.27**	79.87	**79.48**	94.75	**94.67**

As table 4.1 shows, the copy mechanism does not achieve notable improvement over a soft-attention model without the copy mechanism. Among the ensemble models, there is a drop of ~0.1% and 0.45% in the accuracy for English and German respectively and a rise of ~0.2% in the accuracy score for Indonesian.

5 Normalization

Text normalization is the task of mapping non-canonical language, typical of speech transcription and computer-mediated communication, to a standardized writing [Lusetti et al., 2018]. This thesis focuses on the task of normalization of Swiss German, which does not have an orthographic standard but is used for informal written communication such as through messaging. The lack of a written standard and spelling conventions, along with regional variation, ensures variability in written communication through Swiss German. For example, the normalized form viel ('much') is associated to many source words, which all have the same meaning but are spelled differently, such as viel, viil, vill and viu. It is in this context that this work addresses normalization, following Lusetti et al. [2018] in their use of encoder-decoder architectures for the task.

From a machine learning perspective, the differences and similarities between normalization and inflection are similar to the differences and similarities between segmentation and inflection described in chapter 4. Normalization differs from inflection in that the former does not employ morphosyntactic tags to generate the output sequence (the normalized form) from the input sequence (the original word found in the wild). It is similar to inflection in that the input and output strings have low edit distances, which is our motivation for applying the copy mechanism to normalization.

5.1 Data

Two corpora have been used for performing experiments on normalization:

1. The ArchiMob Corpus [Samardžić et al., 2016]
2. The WUS (What's Up, Switzerland?) Corpus [Überwasser and Stark, 2017]

5.1.1 ArchiMob

The ArchiMob corpus consists of transcriptions of video recordings in Swiss German collected by the ArchiMob association, which was normalized in a semi-automatic fashion. The dataset consists of 65132 training, 8967 development, and 8177 test examples.

5.1.2 WUS

The WUS corpus consists of WhatsApp messages written in the languages of Switzerland. Out of the messages written in Swiss German, a small portion was used for manual normalization. It is this manually normalized part of the corpus which has been used during this thesis. It consists of 43370 training, 5418 development, and 5422 test examples.

5.2 Experiments

The experiments conducted for normalization are almost a replication for those done on segmentation as described in section 4.2. To recapitulate, the aim was to incorporate the copy mechanism in an existing implementation of a character level encoder-decoder model with soft attention. To do so, the copy distribution (section 2.2.3) has been generated from the attention distribution generated by the soft attention mechanism using the method described by See et al. [2017].

The experiment is repeated for each of the two corpora: ArchiMob and WUS. For both the corpora, 5 models have been trained with different random seeds. All models have been trained with input (embedding) dimension 100 and hidden layer dimension 200 for 30 epochs. The optimization method used is stochastic gradient descent (SGD).

5.3 Results

Table 5.1: Normalization Accuracy Scores

	Soft Attention ED		Soft Attention ED Ensemble	
Without copy				
Dataset	Dev	Test	Dev	Test

Archimob	86.31	86.11	87.69	87.91
WUS	85.29	85.30	86.60	86.51
With copy				
Dataset	Dev	Test	Dev	Test
Archimob	86.21	86.46	87.86	87.89
WUS	84.80	84.78	85.43	85.79

As table 5.1 shows, the copy mechanism does not achieve notable improvement over a soft-attention model. In fact, there is a notable decline in performance for one of the two corpora used: the WUS corpus, for which accuracy for an ensemble of 5 models goes down from 86.51% to 85.79%.

6 Discussion and Conclusion

The research question this thesis began with asking whether it was possible to obtain performance improvements on the tasks of morphological segmentation and normalization through the use of a copy mechanism. The motivation for this question had come from the significant performance improvement achieved by the copy mechanism on morphological reinflection, as shown by Makarov et al. [2017]. The argument in favour of the possibility of similar results on the other two tasks came from the observation that when modeled as sequence-to-sequence learning tasks, all three of the tasks were similar in consistently having low edit distances between input and output strings which, theoretically, plays in favour of the strengths of the copy mechanism.

As shown by the results obtained on applying it to segmentation and normalization, the copy mechanism appears not to add much value to simple attention based models for both these tasks. Soft attention models both with and without the copy mechanism have similar performances. In the case of the WUS corpus for normalization of Swiss German, the copy mechanism is, in fact, detrimental to the performance of the attention model.

Why it is that performance of the copy mechanism varies between reinflection and segmentation or between reinflection and normalization. There are at least two possible lines of thought here, the first one leading to the second.

The first question arises from the fact that during the current thesis, the copy mechanism was used with a hard attention model to address reinflection and with a soft attention model to address segmentation and normalization. The question, then, is to ask whether the copy mechanism performed well because of the hard attention setting and consequently, whether it would be useful to try a hard attention model for segmentation and normalization. While it may be argued that this might be the cause of the difference, there is a possibility that this might not be the most productive direction. The reason for why this might be so can be broken into two steps. First, consider the case that a hard attention model does capture the problems better. Here, we revisit the observation that while the models for reinflection had a maximum of 1000 training examples, the models for segmentation and normalization had access to much higher a number of training examples, with the number being 8000 for segmentation and greater than 65000 for normalization. A closer look at hard and soft attention suggests that hard attention is, in fact, a special case of soft attention. The argument, then, is that given the higher number of training examples, if hard attention does capture the problem better, then soft attention would *also learn* a harder attention distribution. This would reduce the soft attention model to a hard attention one and the copy mechanism would not help. Next, if the hard attention model does *not* capture the problem better, then its performance would be worse than that of a soft attention model. Appending the copy mechanism on top of such a worse performing hard attention model may or may not improve the performance of that model, but it might not be able to exceed the performance of the soft attention model.

The second question asks whether it might be possible that the copy mechanism itself is rendered less useful as the size of the dataset increases. There does not seem to be a way to reason about this possibility theoretically. It can be checked by training models for segmentation and normalization on a subset of the current corpora and seeing whether the copy mechanism's performance is affected.

Rather than looking for differences in the performance of the copy mechanism in the contexts of hard and soft attention, a more interesting line of inquiry is suggested by a closer look at the predictions made by the models with and without the copy mechanism. Although the copy mechanism does not make significant strides in terms of performance, the model it learns differs from the model learnt by soft attention without the copy mechanism. Concretely, there are some cases where the copy mechanism performs well and others where it performs worse than an attention mechanism without copy.

In case of segmentation, for instance, it is observed that the copy mechanism has a higher tendency to split words - a tendency which plays to its advantage in some examples and to its detriment in others. For example, on one of the test runs for segmentation, the model with the copy mechanism segments the words *dislocate* and *bigotry* to $dis|locate$ and $bigot|ry$ respectively whereas the model without the copy mechanism does not segment either of those two words and outputs them in the original form. In other instances, however, the copy mechanism takes this a little too far. This is especially evident in the case of words which start or end with a common prefix or suffix like $-er$ or $-ic$. For example, the copy mechanism, when given the word *blister*, segments it into $blist|er$. In a more extreme case, when given the word *panic*, the copy mechanism segments this into $pania|ic$. The model without the copy mechanism is able to avoid these missteps and outputs these words in their original form, which is the desired result.

For one of the train - test splits used during segmentation, the predictions made by the models with and without the copy mechanism on the test set were manually inspected. Such inspection showed that out of all the test examples where the two models differed in their predictions, there were a few cases where both models got the predictions wrong. In all of the other cases (33 examples observed), one of the models made the correct prediction, and each of the models was correct in about half of the cases (17 for the copy mechanism and 16 for the other model) which also explains the similar accuracy scores obtained by them. Interestingly, however, in many of the cases where the copy mechanism made the correct prediction (7/17), it was because it split the word where the other model did not, and in more than half of the cases where the model without copy made the correct prediction (10/16), it was because the copy mechanism split the word where it should not have.

This suggests that the copy mechanism overgeneralizes and often splits the word as soon as it encounters the first letter of common suffix. This is further suggested by another set of examples where both the models split but the copy mechanism splits earlier than it should have and ends up committing an error. For example, the copy mechanism splits *theistically* to $theistic|ally$ where the correct segmentation, as made by the soft attention model, is $theistical|ly$. Here also, the copy mechanism splits on encountering the *a*, which might be an overgeneralization.

In view of these observations, a possible direction of future research can be to reconcile these two opposing factors: to attempt to capture the strengths of the copy mechanism which allow it to handle cases which the soft attention model misses, while at the same time suppressing the copy mechanism's tendency to overgeneralize, as a means of combining the strengths of the two models to obtain a more robust solution.

References

R. Aharoni and Y. Goldberg. Morphological Inflection Generationwith Hard Monotonic Attention. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 2004–2015, 2017. URL `https://www.aclweb.org/anthology/P17-1183`.

M. Aronoff and K. Fudeman. *What is Morphology?* Blackwell Publishing, 2011.

N. Chomsky. *Syntactic Structures*. Mouton Publishers, 1957.

R. Cotterell, C. Kirov, J. Sylak-Glassman, D. Yarowsky, J. Eisner, and M. Hulden. The SIGMORPHON 2016 Shared Task—Morphological Reinflection. In *Proceedings of the 14th Annual SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 10–22, 2016a. URL `https://www.aclweb.org/anthology/W16-2002`.

R. Cotterell, A. Kumar, and H. Schütze. Morphological Segmentation Inside-Out. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2325–2330, 2016b. URL `https://www.aclweb.org/anthology/D16-1256`.

R. Cotterell, T. Vieira, and H. Schütze. A Joint Model of Orthography and Morphological Segmentation. In *Proceedings of NAACL-HLT 2016*, pages 664–669, 2016c. URL `https://www.aclweb.org/anthology/N16-1080`.

R. Cotterell, C. Kirov, J. Sylak-Glassman, G. Walther, E. Vylomova, P. Xia, M. Faruqui, S. Kübler, D. Yarowsky, J. Eisner, and M. Hulden. CoNLL-SIGMORPHON 2017 Shared Task:Universal Morphological Reinflection in 52 Languages. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 1–30, 2017. URL `https://www.aclweb.org/anthology/K17-2001`.

M. Lusetti, T. Ruzsics, A. Göhring, T. Samardžić, and E. Stark. Encoder-Decoder Methods for Text Normalization. In *Proceedings of the Fifth Workshop on NLP* *for Similar Languages, Varieties and Dialects*, pages 18–28, 2018. URL `https://www.aclweb.org/anthology/W18-3902`.

P. Makarov, T. Ruzsics, and S. Clematide. Align and Copy: UZH at SIGMORPHON 2017 Shared Task forMorphological Reinflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 49–57, 2017. URL `https://www.aclweb.org/anthology/K17-2004`.

B. H. Partee, A. ter Meulen, and R. E. Wall. *Mathematical Methods in Linguistics*. Kluwer Academic Publishers, 1990.

T. Ruzsics and T. Samardžić. Neural Sequence-to-sequence Learning of Internal Word Structure. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 184–194, 2017. URL `https://www.aclweb.org/anthology/K17-1020`.

T. Samardžić, Y. Scherrer, and E. Glaser. ArchiMob — A corpus of Spoken Swiss German. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, 2016.

A. See, P. J. Liu, and C. D. Manning. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1073–1083, 2017. URL `https://www.aclweb.org/anthology/P17-1099`.

K. Sudoh, S. Mori, and M. Nagata. Noise-aware Character Alignment for Bootstrapping Statistical MachineTransliteration from Bilingual Corpora. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 204–209, 2013. URL `https://www.aclweb.org/anthology/D13-1021`.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

J. Sylak-Glassman, C. Kirov, D. Yarowsky, and R. Que. A Language-Independent Feature Schema for Inflectional Morphology. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pages 674–680, 2015. URL `https://www.aclweb.org/anthology/P15-2111`.

S. Überwasser and E. Stark. What's up, Switzerland? A corpus-based research project in a multilingual country. 2017.