UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Kirill Milintsevich

Lexicon-Enhanced Neural Lemmatization for Estonian

Master's Thesis (30 ECTS)

Supervisor: Kairit Sirts, PhD

Tartu 2020

Lexicon-Enhanced Neural Lemmatization for Estonian

Abstract:

The problem of lemmatization, i.e. recovering the normal, or dictionary form of a word from the text, is one of the crucial parts of the natural language processing applications. It is important for the text preprocessing which is the step of cleaning and preparing the data for the use in NLP models and algorithms. This step can greatly improve the performance of a model if done correctly or, on the other hand, drastically reduce the quality of the output if neglected.

Nowadays, neural networks dominate in the field of NLP as well as in the problem of lemmatization. Most of the recent papers boast to achieve 95-96% accuracy but there is still plenty of room for improvement. As with most of the neural network architectures, the lack of training data can be a huge drawback during the process of model creation. There exist many smaller languages that cannot afford to have large annotated datasets. The Estonian language, being somewhat in the middle in terms of its data size, can benefit from additional data.

In this thesis, we propose a novel approach for lemmatization. In addition to the regular input, the lemmatization model takes the predictions either from another, weaker rule-based lemmatizer or uses the lexicon build from the training data to enhance the lemma prediction. With the combination of several attention layers, the model manages to choose the best from two inputs and produce more accurate lemmas.

Keywords:
Natural language processing, lemmatization, deep learning

CERCS: P176 - Artificial Intelligence

Neurovõrkudel põhineva eestikeelse lemmatiseerija tõhustamine leksikoni abil

Lühikokkuvõte:

Lemmatiseerimisel ehk igale sõnale tekstis algvormi leidmisel on loomuliku keele töötluse rakendustes tähtis roll. See on eriti oluline teksti eeltöötluse etapis, mida kasutatakse andmete puhastamiseks ja ettevalmistamiseks loomuliku keele töötluse mudelite ja algoritmide kasutamiseks. Lemmatiseerimine, kui seda teha õigesti, võib oluliselt parandada mudeli toimivust või teisest küljest vähendada oluliselt tulemuste kvaliteeti, kui see jäetakse tähelepanuta.

Tänapäeval domineerivad tehisnärvivõrgud nii loomuliku keele töötluse valdkonnas kui ka lemmatiseerimise mudelites. Enamik hiljutisi uurimustöid on saavutanud neurovõrkude abil lemmatiseerimises 95 –96% täpsust, kuid arenguruumi on veel piisavalt. Nagu enamiku närvivõrguarhitektuuride puhul, võib treeningandmete puudumine olla probleemiks mudeli loomisel. Paljude väiksemate keelte jaoks ei ole piisavalt suuri treeningandmestikke veel olemas. Ka eesti keel, mis olemasolevate treeningandmestiku suuruse poolest on kusagil keskmisel tasemel, võib saada kasu lisaandmetest.

Käesolevas magistritöös pakume välja uudse lähenemise neurovõrgupõhisele lemmatiseerimisele, mis lisaks tavapärasele sisendile saab sisendiks ka ennustused kas teisest, nõrgemast reeglipõhisest lemmatiseerijast või võib kasutada ka leksikoni andmeid lemma ennustuse tõhustamiseks. Mitme tähelepanukihi kombinatsiooniga suudab mudel valida kahest sisendist parima ja ennustada täpsemaid lemmasid.

Võtmesõnad:

Loomuliku keele töötlus, lemmatiseerimine, süvaõppe algoritmid

CERCS: P176 - Tehisintellekt

Contents

1 Introduction

Natural language processing has many applications such as detecting the sentiment of a text, answering people's questions or opening access to tons of relevant information in seconds. However, before all of the things above are possible to be done, cleaning and transforming the data have to be performed. One of the preprocessing tasks is **lemmatization**.

The goal of lemmatization is to take the word as it appears in the sentence and map it to its normal form, i.e. the form that we usually find in a dictionary. For example, such words as *plays*, *played*, *playing* are mapped to their normal form which is *play*. This process helps to reduce the amount of unique words in a text corpus. Having only one word instead of dozens of different forms makes it easier for the model to learn the patterns in the data as well as allows to put more training data with less computational cost. This is also relevant for the languages with developed declension since the number of possible wordforms is large.

Estonian is a particularly difficult case for this task. Though being a predominantly agglutinative language in its structure, i.e. different endings and suffixes hold unique meaning and are "glued" to the word base, which makes the lemmatization task easier for a computational algorithm to solve, Estonian also shifts towards a more fusional paradigm, i.e. one suffix or ending can introduce multiple features to a word. Moreover, the changes in the word root can make different words have the same forms in the text. An effective lemmatization algorithm is capable of taking into account various parameters to make a distinction in the cases as above and produce a correct dictionary form, or **lemma**.

For Estonian, VABAMORF[1] morphological analyzer uses a combination of dictionary-based and handcrafted rules, and statistical heuristics. This approach allows to clearly see and control each step of morphological analysis, however, it lacks accuracy and precision for rare words that are not covered by the rules. With Estonian becoming a part of Universal Dependencies project[2], most of the modern neural pipelines, such as UDPipe[3], TurkuNLP[4] or Stanza (previously StanfordNLP)[5] support for morphological analysis, lemmatization and syntax parsing for this language. These systems show robust and high performance but can be difficult to understand due to the complex nature of deep neural networks.

To achieve better results in this task, we propose a neural architecture that takes into account an additional input from a third-party lemmatizer or a lexicon. We construct a sequence-to-sequence BiLSTM encoder-decoder model that uses multiple interconnected

[1]https://github.com/Filosoft/vabamorf

[2]https://universaldependencies.org/

[3]http://ufal.mff.cuni.cz/udpipe

[4]https://github.com/TurkuNLP/Turku-neural-parser-pipeline

[5]https://github.com/stanfordnlp/stanza

attention layers to choose the most important parts from each input.

In **Chapter 2**, we define the task of lemmatization and provide an historical background to the approaches used to solve this task. In addition, we outline modern solutions for lemmatization in general and for the Estonian language in particular. Next, **Chapter 3** is dedicated to the explanation of technical background which is used in our method which is described in **Chapter 4**. In **Chapter 5**, we describe the experimental setup for our method. It includes the description of the data, metrics, and statistical background used for the evaluation. Finally, **Chapter 6** contains the results and their discussion, such as detailed error analysis and potential improvements.

2 Lemmatization in NLP

2.1 Lemmatization

To understand the concept of the lemmatization, we can turn to the problem of finding a word in a dictionary. Imagine that we read a book and encounter the sentence *"I wore a beautiful dress yesterday."* Suppose that we do not know what *"wore"* means and we turn to a dictionary to find this out. However, the dictionary most likely does not contain the explanation for the word *"wore"* since it is a past form for the verb *"to wear"*. Thus, *"wore"* is a **surface form** and the dictionary entry *"to wear"* is its **lemma**. The process of conversion from a surface form to a lemma is called **lemmatization**.

Different languages have different models of word inflection. In linguistics, there normally exist the following morphological types of languages: analytic (or isolating) and synthetic which are later divided into agglutinating, (in)fusional and polysynthetic. Usually, analytic languages rely more on the word order rather then changing the word form itself [Sap49], while synthetic languages use different strategies for incorporating additional meaning into a word. This way, synthetic languages add various affixes to the stem, the difference is that in agglutinating languages each affix has only one meaning [Pla99]; in fusional languages affixes are usually polysemantic, i.e. can carry more than one meaning [And85, Wha96]; in polysynthetic languages the word changing paradigm is so developed that the boundary between a word and a sentence is practically non-existent [For94].

This means that the difficulty of the lemmatization task varies from language to language. Thus, purely isolating languages, like Chinese or Vietnamese, do not require lemmatization at all since the surface forms and lemmas are always the same. Agglutinating languages, like Turkish, are easier for lemmatizing since morphemes are "glued" (hence the name "agglutinating") to the stem without changing the form and can be easily detected by a simple rule-based system. Finally, fusional languages, like French or Czech, are challenging to lemmatize since both affixes and word stems are constantly changing [TSG$^+$10].

In reality, however, the languages exist in a continuous space between isolating and polysynthetic [Pla99]. For example, English, being a highly analytic language, still shares some fusional properties like possessive suffix *'s* or irregular forms of verbs in the past tense (*run—ran*). Estonian is a highly inflected language that shares both fusional and agglutinating properties [Eha09]. This makes lemmatization particularly challenging and extremely necessary for processing this language.

One example of the benefits for lemmatization for Estonian can be in web search. If a person wants to search for a bicycle in English, there are only two possible forms of this word: *bicycle* and *bicycles*. In Estonian, with its developed case system, the word *jalgratas* can have more than 30 possible forms. What is more, when the fusional aspect of the language steps in, the changes of the stem also occur (e.g. jalgra**t**as_SING.NOM and jalgra**tt**as_SING.INE). By performing lemmatization, we greatly reduce the number of entries the search engine has to store while also increasing the relevance of the search outputs.

Machine translation can be also improved with the lemmatization, especially when translating from a morphologically rich to a morphologically poor language (e.g. from Estonian to English) [TC09]. Another NLP task where lemmatization can improve the performance is syntactic parsing.

2.2 Approaches to Lemmatization

Early systems relied on carefully hand-crafted rules to find the dictionary form of the word. First, **stemming** was a popular approach to text normalization. Probably, the most popular stemming algorithm is Porter Stemmer [P+80]. Stemming, however, is in many places inferior to lemmatization since it does not take morphological properties into account and just strips away the suffixes from the words. In English, a stem is most of the times equal to a lemma (e.g. *"reads"*—*"read"*), however it is not usually true for morphologically rich languages [KKJ05]. For example, in Estonian the stem for *"ta luges"* (he read) is *"luge"* while the correct infinitive is constructed by adding "-ma" suffix—*"lugema"* (to read).

Rule-based approaches to lemmatization were popular for a long time. Ripple Down Rule approach [PLM+04] has been popular for crafting lemmatization rules. It is based on the if-then rules specifying which suffix should be removed or added to achieve the normal form of the word. Moreover, the recent research shows that rule-based lemmatizers were still developed for Hindi [PTJM13], Urdu [GJM16], and Kannada [PP15] until five years ago. Another example is Apertium, an open-source statistical machine translation tool [FGRN+11] which is still in active development. It contains morphological analyzers based on finite-state transducers and covers 80 languages. These analyzers are capable of producing lemmas and thus can be used for lemmatization. For Estonian, VABAMORF tool is based on two-level rules for morphological analysis and a lexicon check-up to find a lemma [KV01].

In 2014, [SVL14] proposed a sequence-to-sequence model, which was later improved with the attention mechanism by [BCB14], that made a crucial change not only in the machine translation but also in the lemmatization. By regarding lemmatization as a machine translation task, where a surface form is seen as a phrase in the source language and the lemma as a phrase in the target language, the researchers managed to achieve the results that significantly outperformed the traditional rule-based approach [GEM+19].

2.3 Overview of Modern Lemmatization Architectures

As mentioned earlier, almost all of the modern lemmatization systems are neural based. However, even though their architectures can be very similar, they differ in approaches for data preprocessing and usage of external data. Two big competitions—CoNLL 2018 Shared Task [ZHP$^+$18] and SIGMORPHON 2019 Shared Task [MVW$^+$19]—provide us with a detailed overview of existing systems for morphological analysis and lemmatization.

The best performing system for lemmatization in CoNLL 2018 Shared Task is a neural pipeline for TurkuNLP. It uses existing a deep attentional encoder-decoder network from OpenNMT project[6]. As an input, they use a character representation of the surface form concatenated with the corresponding part-of-speech tag and morphological features. The reason behind adding morphology to the input is that the lemmatizer does not see the whole sentence where a word appears, so this contextual information is provided by the morphological features [KGM$^+$18]. After the competition, the TurkuNLP lemmatizer was further improved by using the augmented data. The augmentation was performed by generating new word-lemma pairs with Apertium finite-state morphological analyzer or Unimoprh lexicon[7] [KGS19].

A year after, at SIGMORPHON 2019 Shared Task, UDPipe-Future system[8] scored the best in the lemmatization task. This system was first presented at the CoNLL 2018 Shared Task where it shored the second in the lemmatization task. The UDPipe-Future lemmatization model is a set of classifiers that produce so called *edit rules* that decribe the process of transformation from a word to a lemma. These rules may include such editing operations as deleting, moving or copying a character. The inputs to the model are presented as a concatenation of pretrained word embeddings, trainable word embeddings, and trainable character-level embeddings [Str18]. The main reason for improvement of this system that put it on the first position at SIGMORPHON 2019 Shared Task is adding BERT contextual embeddings [DCLT18] to the input [SSH19].

Another system that is worth mentioning is Stanza (formerly StanfordNLP). It scored the 6th in the lemmatization task at the CoNLL 2018, however it is a well-thought piece of software which is easy to use and modify, it is still constantly updated and has an active community of both researchers and end-users. For lemmatization, their approach is similar to the system from TurkuNLP—Stanza's lemmatizer is an encoder-decoder neural network. The input consists of the character representation of the word without any additional features. The authors argue that including part-of-speech tags to the input made their system perform worse for several languages [QDZM19]. Even though inferior in terms of raw performance compared to the other top systems, Stanza provides a perfect foundation for developing a custom pipeline.

[6]https://opennmt.net/

[7]http://unimorph.org/

[8]https://github.com/CoNLL-UD-2018/UDPipe-Future

2.4 Lemmatization Scene for Estonian

For Estonian, VABAMORF morphological analyzer has been developed since 1995 (first under the name ESTMORF) [MMP17]. VABAMORF is a combination of a rule-based and statistical models. For the lemmatization, it mainly relies on a dictionary-based approach, where the lemma is find by a simple look-up. It also features a guesser module, which can predict a lemma for a word not found in the dictionary using the set of rules. The ambiguous output can be also later disambiguated using the Hidden Markov Model which incorporates the local context, i.e. a sentence, to eliminate the least probable predictions [KV01]. Additionally, a broader context, such as the whole text or even a set of connected texts, can be used to disambiguate between lemmas [KKM12]. VABAMORF is incorporated into the ESTNLTK toolbox—a Python package for analyzing Estonian texts—which makes it easy to use and incorporate into another systems.

Concerning the recent neural approaches, all the top systems from the two competitions, described in the previous Section, show high results for Estonian. This, at CoNLL 2018 Shared Task, TurkuNLP system is ranked the first for Estonian, showing 96.57% accuracy in lemma prediction; the second place is taken by the UDPipe-Future System with 94.88% in accuracy; Stanza is place the third with 94.52% accuracy for Estonian. The improved version of the UDPipe-Future that was presented at SIGMORPHON 2019 Shared Task shows 96.59% accuracy. However, the results models in the two shared tasks were tested on a slightly different data, thus the results are not directly comparable and should be used only for the reference.

3 Technical Background

This chapter aims to give an overview sequence-to-sequence (seq2seq) networks. Seq2seq encoder-decoder networks provide us with the state-of-the-art solutions for lemmatization nowadays. That is why it is important to understand how they work. What is more, the neural architecture that we propose in **Chapter 4** also has a seq2seq encoder-decoder network as its core.

3.1 Encoder-Decoder Architecture

In general, a seq2seq model consists of two main components: **encoder** and **decoder**. Encoder takes the input sequence, transformed with an embedding layer, and produces its hidden representation. Later, this representation is fed into decoder that outputs the probabilities of the output sequence. Figure 1 shows the overall architecture of the model.

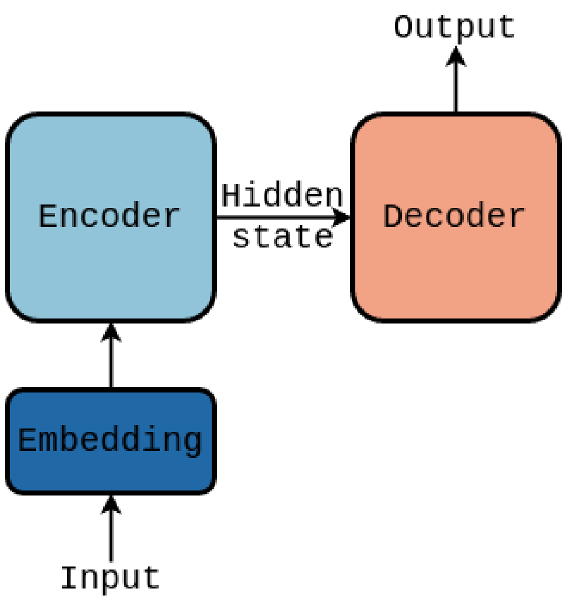


Figure 1. General architecture of an encoder-decoder model

3.1.1 Encoder

Encoders can be built of various types of hidden layers. The most popular way to construct an encoder is to use a recurrent neural network (RNN). RNNs are undoubtedly good at processing sequential variable-length inputs and can be successfully used for a language modelling task [MKB+10]. Furthermore, with the introduction of long short-term memory (LSTM) RNNs [HS97] the quality of the language modelling improved even further [SSN12].

When using an RNN encoder, the hidden state h_t^e is propagated through the RNN cells for each element in the sequence at a timestep $t = \{0, \ldots, n\}$. The formula to compute h_t^e is the following:

$$h_t^e = f(W^{(hh)} h_{t-1}^e + W^{(hx)} x_t)$$

where $W^{(hh)}$ is a weight matrix for hidden states h^e, $W^{(hx)}$ is a weight matrix for inputs x [SVL14].

3.1.2 Decoder

Decoder is structurally similar to the encoder. It also consists for several recurrent units which predict an output y_t at each timestep t. However, while the initial hidden state for an encoder is random, a decoder is initiated with the final hidden state from the encoder. The hidden state for the decoder h_t^d is computed with the following formula:

$$\begin{aligned} c &= h_n^e \\ h_0^d &= c \\ h_t^d &= g(\hat{y}_{t-1}, h_{t-1}^d, c) \end{aligned}$$

where c is a context vector, g is any RNN, and $\hat{y}_{t-1}$ is the embedding for the output sampled from the softmax at the previous step which is calculated as:

$$\begin{aligned} z_t &= f(h_t^d) \\ y_t &= \text{softmax}(z_t) \end{aligned}$$

To restrict the length of a decoded output, usually a special end-of-sequence symbol is used. The output is generated until the end-of-sequence symbol was produced or the maximum sequence length was reached.

3.2 Long Short-Term Memory Network

In this work, we build a neural model based on recurrent layers. In particular, we use a long short-term memory (LSTM) network as a core of both encoder and decoder.

As mentioned in 3.1, recurrent neural networks are good at learning from sequences. To better understand the functionality behind them, we review the simplest RNN proposed by [Elm90]. **Figure 2** shows a simple RNN encoder that consists of three RNN cells. It combines the previous hidden state and an input, later passing them through the TanH activation function. Next, the output of the previous cell is passed as a hidden state to the next cell. This way, the model manages to learn from the previous context.

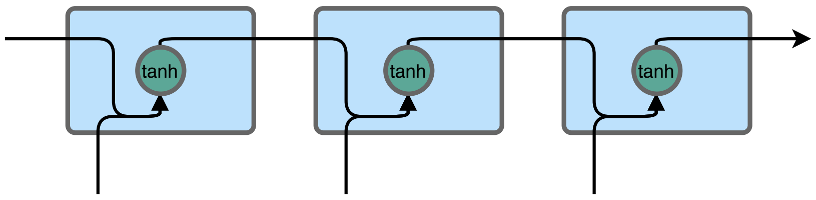


Figure 2. Simple RNN encoder with TanH activation function

This simple architecture may seem well adapted for capturing the context, however, in practice simple RNNs suffer from being unable to learn long-term dependencies [BSF94]. It happens because usually RNN architectures are very deep, and, during the backpropagation, the gradient tends to vanish or explode.

This problem was tackled with several modifications of an RNN, one of them being a LSTM network. To overcome the problem of vanishing or exploding gradients, [HS97] introduced, in addition to the hidden state h_t, a new cell state c_t which carries additional information between the cells. To determine which information will be forgotten or not, all the inputs are passed through a forget gate f_t, an input gate i_t, and an output gate o_t. While **Figure 3** gives a visual overview of the transformations within a LSTM cell, we will formulate and explain each operation in more detail.

Forget gate First, to determine which information to discard, the inputs are passed through the forget gate. (2) shows how the output of the forget gate is calculated. Here and henceforth, W_q and U_q are, respectively, the input and recurrent weights for the gate $_q$ which can be $_f$ for the forget gate, $_i$ for the input gate, $_o$ for the output gate, and $_c$ for the memory cell. σ is the sigmoid activation function (1).

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

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2}$$

Input gate To decide which values to update, the inputs are passed through the input gate. (3) shows how the output of the input gate is calculated. In addition, new candidate cell values $\hat{C}_t$ are calculated in (4).

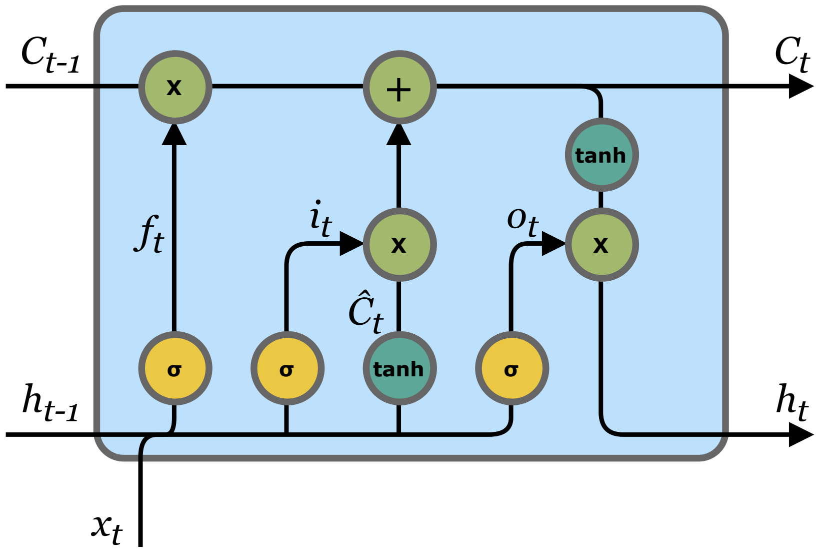


Figure 3. Detailed view of an LSTM cell

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{3}$$

$$\hat{C}_t = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right) \tag{4}$$

To update the previous cell state C_{t-1} we use the outputs of the forget gate f_t, input gate i_t, and the candidate cell state values $\hat{C}_t$ are combined as shown in (5). The new cell state C_t will now carry all the updated information for the next LSTM cell.

$$C_t = f_t \circ C_{t-1} + i_t \circ \hat{C}_t \tag{5}$$

Output gate To finish the calculations, the previous inputs are passed through the output gate. This gate produces an updated hidden state h_t which will serve and an input for the next LSTM cell as well as a final output of a current cell at a timestep t. The output gate result o_t is calculated similarly to the previous gates, as seen in (6), and the new hidden state h_t is constructed using the updated cell state C_t as shown in (7).

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{6}$$

$$h_t = o_t \circ \tanh\left(C_t\right) \tag{7}$$

What we described above is a basic LSTM architecture. There are several modifications above this architecture. One notable example is adding "peephole connections" inside an LSTM cell that allow some of or all the gates to have access to the cell state [GS00]. Another major modification of an LSTM is a Gated Recurrent Unit (GRU) that combines input and forget gates as well as cell and hidden states [CVMG$^+$14].

Finally, a traditional LSTM treats an input sequence from left to right, thus taking only past context into consideration. However, it often happens that future context contains useful information. To tackle this problem, bidirectional recurrent neural networks (BRNN or BiRNN) were proposed [SP97]. These networks have a second recurrent layer that passes hidden states from right to left. The layers share the same inputs and combine both outputs in the end by stacking them together.

3.3 Attention

As we saw in the previous section, RNNs are powerful in capturing the sequence information. However, there is still a room for improvement and one serious bottleneck to be dealt with. When building a RNN encoder-decoder model, only the last hidden state from an encoder goes into a decoder, as seen on **Figure 2**. It can cause problems when encoding long sequences since the encoder has a task to encapsulate the whole sequence into a vector of a restricted size. This may result in the information contained in the beginning of a long sequence being lost. The solution was found in the attention mechanism proposed by [BCB14].

To formulate the attention mechanism as in [BCB14], we introduce the following variables. A source sequence x of length n and an output sequence y of length m.

$$\begin{aligned} x &= [x_1, x_2, \dots, x_{n-1}, x_n] \\ y &= [y_1, y_2, \dots, y_{m-1}, y_m] \end{aligned}$$

The decoder produces an output $s_t = f(s_{t-1}, y_{t-1}, c_t)$ for a timestamp t, where the vector c_t is a weighted sum of input hidden states and h_i is the final hidden state produced by the encoder.

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i$$

Here, $\alpha_{t,i}$ is an alignment, or attention score.

$$\begin{aligned} \alpha_{t,i} &= \text{align}(y_t, x_i) \\ &= \frac{e^{\text{score}(s_{t-1}, h_i)}}{\sum_{i'=1}^{n} e^{\text{score}(s_{t-1}, h_i')}} \\ &= \text{softmax}(\text{score}(s_{t-1}, h_i)) \end{aligned}$$

In [BCB14], the authors parametrize α "*as a feedforward neural network which is jointly trained with all the other components of the proposed system*". They also define the score function as follows:

$$\text{score}(s_t, h_i) = v_a^T \tanh\left(W_a[s_t, h_i]\right)$$

where v_a and W_a are the weight matrices learned by the network. This score function is also called *additive*.

Later, the researchers proposed many other ways of calculating and applying the attention score. In this work, we use a soft dot attention. *Dot attention* was introduced by [LPM15]; they redefine the score function as:

$$\text{score}(s_t, h_i) = s_t^T h_i$$

In addition, there is another modification of dot attention which is called *scaled dot attention* [VSP+17]. It is defined as follows:

$$\text{score}(s_t, h_i) = \frac{s_t^T h_i}{\sqrt{n}}$$

where n is the dimension of the source hidden state. This helps to avoid a small gradient produced by softmax function when the input is large.

Finally, [XBK+15] introduce *soft* and *hard* attention mechanisms. In soft attention, the alignment weights are learned and applied over the whole sequence, behaving in the same way as proposed in the original paper by [BCB14]. This allows the attention to be differentiable yet computationally expensive if the input sequence is large. Hard attention, on the other hand, only applies to parts of a sequence at a given timestamp. This makes it less computationally expensive but at the same time non-differentiable, thus requiring more complicated training techniques.

4 Lexicon-Enhanced Lemmatizer

In this thesis, we propose the **LEX**icon-**EN**hanced **LEM**matization (LEXENLEM) model. LEXENLEM is a seq2seq encoder-decoder model which benefits from using a third-party lemmatizer as a second input. A third-party lemmatizer can be either a rule-based morphological analyzer or a lexicon. Hereafter, we refer to the output produced by a third-party lemmatizer as a **candidate**.

As a core, we use Stanza pipeline for the lemmatization [QZZ+20]. Stanza's lemmatization module is a seq2seq neural network. The encoder is represented by a single BiLSTM layer. The decoder is a single-way LSTM with one soft dot attention layer and one linear output layer that maps the output of the LSTM layer to the size of the vocabulary. The inputs to the Stanza's lemmatizer consist of the characters of the source word framed by <SOS> (start-of-sequence) and <EOS> (end-of-sequence) symbols. Before getting into the encoder, each character is encoded with a trainable embedding layer and the dropout is applied to each encoded sequence during the training phase.

We chose Stanza as the base for LEXENLEM since it has a generic seq2seq encoder-decoder architecture, various utilities for loading and processing the data, well-written documentation, and live and responsive community. In addition, Stanza is implemented in PyTorch[9] which is a well developed machine learning framework with an extensive API and documentation. Finally, Stanza is still in active development; the authors regularly publish up-to-date models that makes it easier to compare our system with Stanza. Even through Stanza has been recently published, it is an heir of StanfordNLP system which was presented in CONLL 2018 shared task [QDZM19] and ranked 6th for the lemmatization task[10].

The architecture of the LEXENLEM model is shown on **Figure 4**. Compared to Stanza, we add the second BiLSTM encoder that takes candidates from a third-party lemmatizer. Both encoders are structurally the same as the original encoder. Another difference is that the input for the first encoder, or $input_1$ on the Figure 4, also contains a universal part-of-speech tag (UPOS) and a set of morphological features (FEATS). UPOS and FEATS are represented as a single character and embedded together with the rest of the input. **Figure 5** shows an example of $input_1$. The candidate is transformed in the same manner and passed into the second encoder as $input_2$. In case when several candidates are produced for the same $input_1$, they are concatenated without any separators and embedded as a single string. The Encoder produces four outputs: h^1 and h^2 are the last hidden states of the first and second encoder respectively; h^n is a combination of h^1 and h^2; c^n is a combination of the last cell states produced by the both encoders. Since the encoder is a bidirectional LSTM, both final hidden and cell states are a concatenation of final states from two directions of the network. Cell states are a product of using

[9]PyTorch is a machine learning framework for Python. URL: https://pytorch.org/

[10]https://universaldependencies.org/conll18/results-lemmas.html

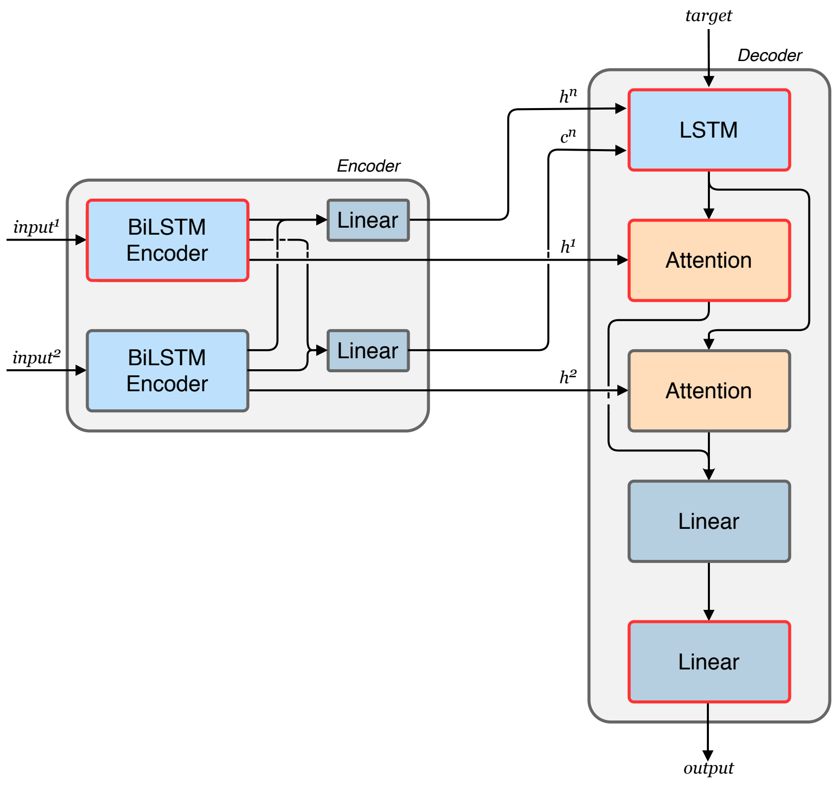


Figure 4. Model architecture with double encoder and double attention decoder. Layers that comprise the original Stanza lemmatizer are shown in red.

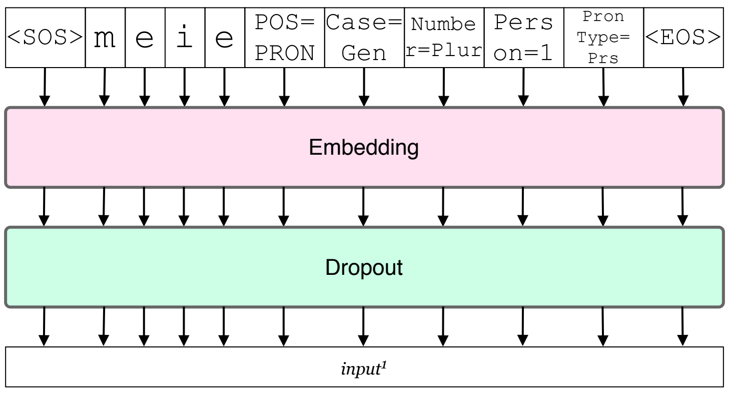


Figure 5. Transformation process for an source input. <SOS> and <EOS> are start-of-sequence and end-of-sequence tokens correspondingly. Morphological features are stored as strings.

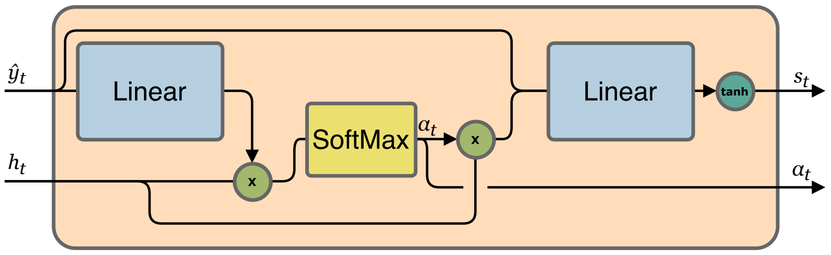


Figure 6. Attention layer used in the decoder

LSTM architecture. Finally, we use a linear layer to combine the hidden and cell states.

To accommodate additional outputs produced by the Encoder, we modify the Decoder as well. As shown on the **Figure 4**, originally, the output states h^n and c^n from the Encoder were use as the initial hidden and cell states of the LSTM layer in the Decoder together with the *target* sequence as an input. During the training phase, the *target* is the correct lemma and during inference *target* starts with the <SOS> symbol with the new character predictions appended after each decoding step. In the end, the weighted output is mapped to the vocabulary with the linear layer. Since there are two encoders now, we add a second attention layer which shares the same output h^y from the first LSTM layer and takes the final hidden state from the second encoder h^2 as inputs. The outputs of the both attention layers are concatenated with the linear layer before mapping to the vocabulary.

Similar dual-encoder approach was used in [SL18] for neural machine translation task. Even though the authors failed to achieve any significant improvements, we saw potential in this approach and revised it for LexEnLem. Furthermore, in recent years, multi-encoder models were successfully used for conversational models [RCL+19], speech recognition [LWM+18], and image processing [BCÇŞ20].

5 Experiments

To test our hypothesis that a second encoder can improve the prediction accuracy, we carry out four experiments:

1. LEXENLEM model that always has an empty input to the second encoder, i.e. uses only source input, POS tag, and morphological features (**DEFAULT**);
2. Second encoder takes a predicted lemma from the lexicon, constructed from the training set, as an input to the second encoder (**LEXICON**);
3. Second encoder takes a predicted lemma from the VABAMORF morphological analyzer with the disambiguation and guessing modules enabled (**VABAMORFALL**);
4. Second encoder takes a predicted lemma from the VABAMORF morphological analyzer with the disambiguation and guessing modules disabled (**VABAMORFNOGUESS**).

5.1 Data

5.1.1 Universal Dependencies

To train and test our model, we use the data from the Universal Dependencies (UD) project[11]. UD gathers researches from all over the world who work on annotating data for their language. Each language has a treebank, a set of annotated text data in CoNLL-U format. As for now, UD has more than 150 treebanks in 90 languages.

All the treebanks are annotated using the same CoNLL-U schema that makes creating multilingual pipelines very easy. Almost every treebank, with the exception of very small ones, has a train, development, and test sets. CoNLL-U format is a tab separated document with each sentence split into words. Each word is placed on a new line and has 10 fields containing the following information: ID (word index), FORM (surface form of a word), LEMMA, UPOS (universal POS tag), XPOS (language-specific POS tag), FEATS (list of morphological features), HEAD, DEPREL (universal dependency relation to the HEAD), DEPS, MISC. For our experiments, we are interested in FORM, LEMMA, UPOS, and FEATS fields.

For the Estonian language, we use Estonian Dependency Treebank (EDT) v2.5[12] [MMP+14]. The treebank contains 437,769 tokens and features texts from fiction, newspapers, and scientific texts. In addition, lemmas are annotated with word formation labelling, specifically for compounding and morphological derivation. The compounding of several roots is marked with "_" while morphological derivation is marked with "=" and "+". In fact, only "=" stands for pure derivation while "+" denotes inflectional changes. However, it is

[11] https://universaldependencies.org/

[12] https://github.com/UniversalDependencies/UD_Estonian-EDT/tree/master

difficult to draw the boundary between the two even for linguists [And82], so we group both of them under morphological derivation for the sake of simplicity. For example, the lemma for the word *ostusedelisse* (in the shopping list) is *ostu_sedel*. The underscore in the lemma shows that the word is compound and in fact consists of two words: *ostu* (shopping) and *sedel* (a list). An example of morphological derivation is *improviseeri=mine* (improvisation) since this noun is derived from the verb *improviseerima* (to improvise) with the help of the derivational suffix "-mine". In addition, several types for word formation can be combined in one word, e.g. *teguri+te_rühm* (group of factors) where "-te" is an inflectional suffix of plural genitive case.

During the training phase, we use the correct POS tags and morhpological features from the treebank. During the testing phase, we predict them with Stanza pipeline using the model trained on the same EDT v2.5 treebank. The model shows 97.19 accuracy for predicting UPOS tags and 95.77 accuracy for predicting FEATS.

5.1.2 Lexicon

In cases when a third-party lemmatizer is not available, it can be simulated by building a lexicon from the training data. In this case, we build the Lexicon as consisting of two look-up dictionaries: the first one contains the pair (FORM, UPOS) as a key and LEMMA as a value; the second one has FORM as a key and LEMMA as a value. During training and prediction, the model first compares if the first dictionary has a (FORM, POS) key for the input word and takes the corresponding LEMMA if it is true. If not, the model checks if the second dictionary has a FORM key and takes the corresponding LEMMA if it is true. Finally, if all the previous checks failed, the Lexicon returns an empty list. If there exist several lemmas for the same key, they are concatenated into one string.

During training, we noticed that the model's performance dropped significantly when using the Lexicon. Upon further investigation, we found out that the model tends to "overtrust" the lexicon and thus the model did not learn properly to predict on unknown vocabulary. This happens because the Lexicon contains all the forms from the training set, which the model trains on, and the simplest way to predict a lemma is now take it from the Lexicon. During the prediction phase, however, the Lexicon returns empty predictions for unknown words. To overcome this, we introduce the Lexicon dropout which is a probability of discarding a lexicon candidate during the forward pass of the model. To decide which candidates are discarded, a pseudo-random real number in the range $(0, 1)$ is generated for each candidate in the input batch. If this number is less that the Lexicon dropout value, the candidate is replaced with an empty string.

5.1.3 Vabamorf

As a third-party lemmatizer, we use VABAMORF[13] which is a rule-based and statistical analyzer for Estonian. It predicts both morphological categories as well as lemmas. In its core, is analyzes the input by looking it up in the dictionary and applying various morphological rules. The authors claim that approximately 98% of the words can be analyzed this way [KV01].

VABAMORF also features guessing and disambiguation modules. The guesser uses various heuristics to predict the lemmas for words that are not found in the dictionary. The disambiguator is present as a probabilistic Hidden Markov Model (HMM). It aims to reduce the ambiguity when a certain word form has multiple possible morphological analyses. To choose the appropriate prediction, context is taken into account. Usually, the context is restricted by the sentence boundaries, but the authors also claim that better disambiguation can be achieved if using the whole paragraph or text as the context [KV01].

To incorporate VABAMORF into our model, we use it from ESTNLTK[14] tool since it provides Python mappings and is easier to install and use in general.

5.2 Experimental Setup

All the models were trained using the Rocket Cluster of the High Performance Computing (HPC) Center at the University of Tartu. The hardware used for training is the following: 2 x Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz (48 cores total), 512 GB RAM, NVIDIA Tesla P100 GPU with 12GB of VRAM.

The model hyperparameters are shown in **Table 1**. Most of the hyperparameters are taken from the original Stanza model. These hyperparameters were found as a result of an extensive research [QZZ$^+$20]. We additionally introduce the Lexicon dropout of 0.8 since this is an approximate amount of words from the training set that are not present in the test set.

Each model was trained for 60 epochs with an early stopping if no improvement was observed in the validation score during 10 epochs. The models are trained using the negative log likelihood loss and Adam optimizer. In the end, the model with the best validation score is saved as used for testing.

Finally, as a decoding strategy to produce the output lemma, we choose a greedy search. We conducted experiments with beam search decoder with beam sizes $\{3, 5\}$ and found out that it does not give any improvement in the accuracy of produced lemma. Moreover, beam search proves to be much slower than a simple greedy search.

[13]https://github.com/Filosoft/vabamorf
[14]https://github.com/estnltk/estnltk

Hyperparameter	**Value**
Encoder hidden dimension	100
Decoder hidden dimension	200
Embedding dimension	50
# encoder layers	1
# decoder layers	1
Embedding dropout	0.5
Lexicon dropout	0.8
Maximum decoded length	50
Learning rate	1e-3
Learning rate decay	0.9
Batch size	50

Table 1. Model hyperparameters

5.3 Evaluation

5.3.1 Metrics

We use accuracy to measure the performance of the models. Accuracy is defined as:

$$\text{accuracy} = \frac{n_c}{N}$$

where n_c is the number of correct predictions and N is the total number of lemmas. The prediction is considered to be correct if all the characters match exactly the gold lemma. In addition, we report the results as percentage, i.e. accuracy of 96.87 means that 96.87% of all the predictions were correct.

5.3.2 Paired Bootstrap Resampling Test

If the difference in scores of two systems is too small, it is difficult to state that one system really outperforms another and it is not a random fluctuation. PBRT is one of the ways to do it. To perform a PBRT, the sentences of each system's gold and predicted sets are resampled with replacement n_r times (usually $n_r > 1000$), and the score of each resample is taken. Then, it is counted how many times one system perfomed better or worse than another, thus the number of wins n_w is calculated. Later, the scores for each system are sorted by the middle score. Finally, the p-value is calculated as

$$p\text{-value} = \frac{n_w}{n_r}$$

and 95% confidence interval is taken to calculate the score deviation from the average. If the p-value < 0.05, we can state with 95% confidence that one system outperforms another.

This method can be used for evaluating machine translation systems [Koe04], as well as end-to-end NLP pipelines. It was used to rank the systems in CoNLL-U 2018 shared task [ZHP$^+$18]. Since we work with the same data, we revised their implementation[15] of PBRT to compare the systems in this research.

[15]https://github.com/udapi/udapi-python/blob/master/udapi/block/eval/conll18.py

Rank	Model	Accuracy		
		Average (p-value)	OOV	Known
1-4	VabamorfNoGuess	96.87 ± 0.17 (0.081)	88.64	98.58
1-4	VabamorfAll	96.70 ± 0.17 (0.056)	87.69	98.57
1-4	Lexicon	96.49 ± 0.18 (0.170)	85.49	98.78
1-4	Default	96.36 ± 0.20 (0.038)	86.16	98.48
5	Stanza	96.11 ± 0.19	83.86	98.66

Table 2. Experimental results for our four models and the Stanza baseline.

6 Results and Discussion

6.1 Results

The results of all experiments are given in **Table 2**. The line between the systems indicates that the difference between them is statistically significant. To test the statistical significance between the model scores, we use paired bootstrap resampling test with 1000 resamples and 95% confidence interval.

Models with Vabamorf candidates as a second input outperform all the other models. Particularly, the model which takes the input from Vabamorf with the guesser and disambiguator module disabled (**VabamorfNoGuess**) performs slightly better than the model where the candidate inputs to the second encoder were disambiguated by Vabamorf (**VabamorfAll**). Another multi-encoder model that takes the candidates from the generated Lexicon (**Lexicon**) is placed right after. However, showing the highest p-value when compared to the model without the second encoder (**Default**), the Lexicon model shows the least evidence of any significant improvement.

Compared to the Baseline (**Stanza**), the best model VabamorfNoGuess achieves 19.54% error reduction. Moreover, when compared directly, VabamorfNoGuess shows a statistically significant improvement over the Default model with the p-value of 0.002.

In addition, we separately test the accuracy of Vabamorf. All the texts use the tokenization from the UD data. It was run in three different settings: the default setting, with the disambiguator and guesser enabled; the setting without using the disambiguator and guesser; the setting with global disambiguator. The difference between the default and global diambiguation is that the first one uses only the local context, i.e. the sentence, to discard incorrect predictions while the global disambiguation uses the whole text as a context. The concept of global disambiguation is based on the assumption that ambiguous words reoccur in other parts of the text, so it is possible to choose the final

Model	Missing		Misplaced		Misc	Total
	COM	DER	COM	DER		
VABAMORFNOGUESS	82	225	197	124	890	1518
VABAMORFALL	65	193	203	154	985	1600
LEXICON	95	203	194	89	1121	1702
DEFAULT	107	211	166	120	1163	1767
Stanza	146	192	150	82	1316	1886

Table 3. Error analysis for the models. **COM** signifies the symbol of a compound word "_", **DER** is the derivational symbol "+" or "=".

prediction based on the most frequent lemma for such a word [KKM12]. Since the Estonian UD corpus contains the metadata for each sentence, it is possible to group all the sentences back into the texts.

As result, VABAMORF with local disambiguator and guesser shows 81.06% accuracy on the test set, 57.00% accuracy without the disambiguator and guesser, and 81.77% accuracy using global disambiguator. In case whem VABAMORF produced several predictions, only the first one was compared to the gold lemma.

The low accuracy for VABAMORF without any disambiguator and guesser is due to many missing predictions. Since the guesser module is disabled, about 19% of all words in the training set were not lemmatized and thus counted as erroneous. Moreover, 10% of words had two predictions and 3% had three predictions, compared to 3% and 0.1% respectively when local disambiguator and guesser were enabled. Finally, global disambiguation slightly improved the accuracy of VABAMORF.

6.2 Discussion

6.2.1 Error Analysis

Upon closer examination of the errors, we found out that most of them are connected with word formation labelling. **Table 3** shows that all the models have almost equal amount of errors connected with either compound word symbol "_" (**DER**) or derivational symbols "+" and "=" (**DER**). Specifically, **Missing COM** and **Missing DER** stand for the errors when the respective symbol is present in the gold lemma but not present in the predicted lemma and if removed, the prediction is correct (e.g. correct: *"laua_naaber"*; predicted: *"lauanaaber"*). **Misplaced COM** and **Misplaced DER** stand for the error when the respective symbol is present in both gold and predicted lemmas but is misplaced (e.g. correct: *"ostu_sedel"*; predicted: *"ostus_edel"*) or it is not present in gold but present in predicted and if removed, the prediction is correct (e.g. correct: *"seostamine"*;

predicted: *"seosta=mine"*). **Misc** stands for all the other errors.

We investigated these errors further and discovered that the annotation in the EDT corpus is inconsistent. For example, for the word "*maailm*" (the world) and its forms, the test set contains 21 lemma where the two parts—"*maa*" (the land) and "*ilm*" (the weather)—are separated by the compounding symbol ("*maa_ilm*") and 24 lemmas as one atomic word ("*maailm*"). Morphological derivation—marked by "=" for the suffixes and "+" for the endings—also shows inconsistency. For example, suffixes "*-mine*" and "*-sus*" (variation of "*-us*") are clearly derivational and highly productive in Estonian [Vik94], however, they are not always marked with "=" in the data cf. "*improviseeri=mine*" and "*tegemine*".

When analyzing other mistakes, the system mostly fails to predict on long words (e.g. "*pophitti-raadiomulli*" predicted as "*pophitti-raadill*") and tokens that include several words separated by "/" (e.g. "*taglierinid/tagliatelled*") or "()" (e.g. *"(kuri)tarvitamise"*). After analyzing the inputs for the VABAMORFNOGUESS model for these cases, we discovered that VABAMORF predicts only the first word in the first case and completely breaks in the second case by predicting only "(" symbol. However, the annotation of these cases is also questionable. In the EDT corpus, "*taglierinid/tagliatelled*" is lemmatized as "*taglierinid/tagliatelle*" but "*noodile/sõnale*" is lemmatized as "*noot/sõna*". In first example, only the second word is lemmatized while in the second one they lemmatize both words.

Another common error is lemmatizing verb past participles ending on "-tud" as adjectives. This error mostly occurs because of the error in POS tagging. For example, if the POS tag for a "-tud" word is incorrectly predicted as a verb instead of an adjective, the model will predict a "-ma" infinitive. In addition, it is worth noting that "-tud" forms are always left ambiguous in VABAMORF because "it is impossible to say whether they are verbs or adjectives unless the immediate context is provided." [KV01].

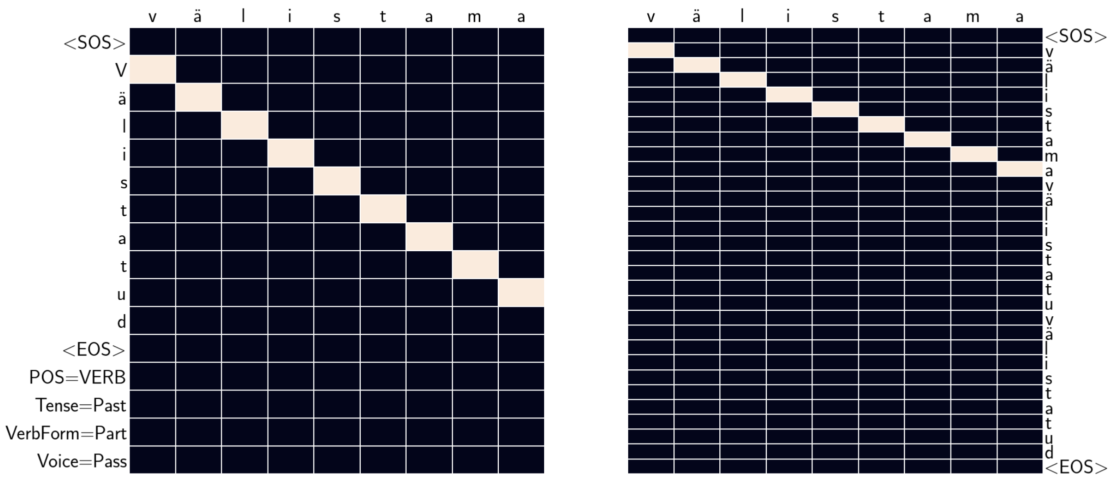
Pronouns "*see*" and "*tema*" are often confused by the models. These two pronouns share the same forms in their plural form "*nemad*" thus creating confusion for the models.

Finally, a many mistakes include wrong capitalization. It mostly occurs in proper nouns but sometimes random capitalization is present in the middle of the word.

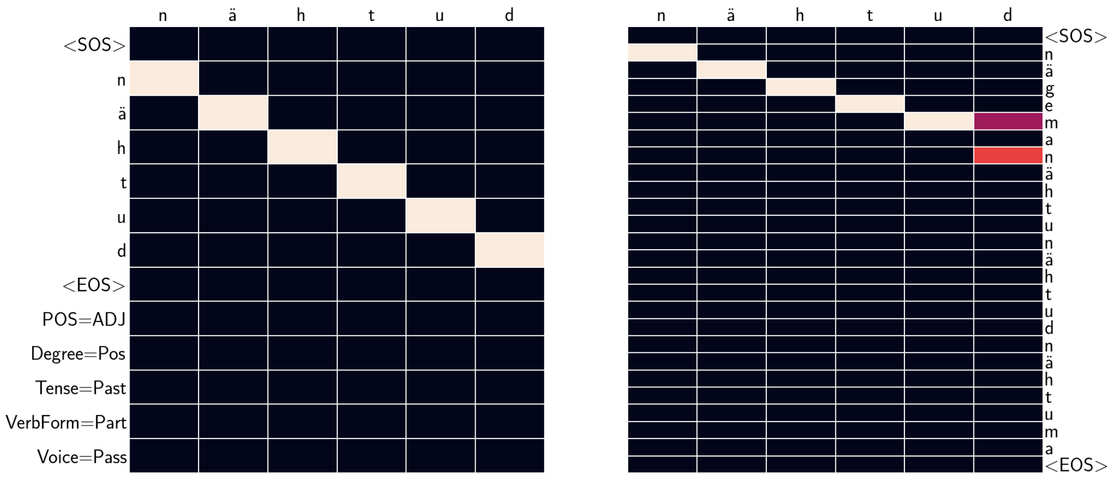
6.2.2 Analysis of Attention Weights

Visualizing attention weights is a good way to look into the model's reasoning. **Figure 7** shows the attention weights during prediction of ambiguous words with VABAMORFNOGUESS model. The graph on the left shows the attention scores for the first encoder which takes the source token and morphological features as input. The graph on the right shows the scores for the second encoder which takes the predictions from VABAMORF as input. In the bottom of each graph is the predicted lemma and on the sides are the inputs for the corresponding encoders.

It can be seen how on **Figure 7a** the model fully attends to the first prediction from



(a) Attention points to the first prediction from VABAMORF during decoding



(b) Attention points to the second prediction from VABAMORF during decoding

Figure 7. Attention visualization for VBFNOGUESSER model.

⬜ 1.0
🟥 0.6
🟪 0.4
⬛ 0.0

VABAMORF since it is the correct one for this case: the source word *Välistatud* is a verb and should be brought to its -ma infinitive form *välistama*. On **Figure 7b**, we can see that the model's attention switches from *nägema* to *nähtu* at the right graph. The second prediction resemble more the correct lemma *nähtud* since the source word *nähtud* is an adjective in this case.

It seems, however, that attention does not point at POS tags and morphological features in these examples but [KGS19] shows the same pattern. This can happen because the morphological information is already encoded in the corresponding characters, however this hypothesis is yet to be tested.

6.2.3 Future Work

There are several steps can be made to potentially improve the quality of the Estonian lemmatization. These steps include not only technical modifications of the model but also several conventional questions.

First, it is possible to improve the quality of the candidates for the model by better preprocessing the predictions from VABAMORF. As we saw above, tokens with different separating symbols are mistreated by VABAMORF. It is possible to preserve the original tokenization by introducing a custom tokenizer in ESTNLTK. In addition, VABAMORF supports the same word formation labelling ("_", "=", and "+" symbols) as in the Data used for training the models. This may, or may not, improve the performance of our models that use candidates from Vabamorf (VABAMORFALL and VABAMORFNOGUESS) depending on how accurate Vabamorf is with the word formation annotation.

Next, the data can be reworked to improve the quality of lemmatization. One way is to augment the training data using VABAMORF. Similar approach is used in [KGS19] by adding new training samples from unambiguous predictions of a third-party lemmatizer. Another approach is reducing the inconsistency with labelling of compound and derived words. This can be achieved by consulting the team responsible for the corpus annotation.

Finally, several modifications can be introduced to LEXENLEM model. On of them is to introduce a linear classifier to predict if the word is compound or derived. This can possibly reduce the number of errors where the word formation labelling is missing in the prediction. The similar approach is implemented in Stanza for classifying if the lemma and the surface form are same or not [QZZ+20]. Another potential improvement is constraining the capitalization in the prediction. It can be achieve by adding a *casing classifier* as it is done in UDPipe-Future [SSH19]. This will help to reduce the errors where wrong capitalization occurs in the middle of the word.

These steps can potentially improve the quality of our models for Estonian lemmatization but they come with the trade-off of shifting the models from being language-independent to language-specific.

7 Conclusion

LEXicon-**EN**hanced **LEM**matization (LEXENLEM) is a novel approach to lemmatization. It is an encoder-decoder neural model that incorporates a third-party statistical lemmatizer by adding a second encoder to the model and an additional attention layer to the decoder. The key difference of our method from other systems which use external sources for lemmatization is that both data and a third-party system can be easily and independently plugged into LEXENLEM. Moreover, if a third-party lemmatizer is not available, it can be substituted by a lexicon constructed from the training data. This makes LEXENLEM easily adaptable for other languages as well.

We showed that using LEXENLEM together with VABAMORF statistical morphological analyzer improved the lemmatization accuracy for Estonian by 0.76%, reaching $(96.87 \pm 0.17)\%$ accuracy, compared to a relatively strong baseline. We also demonstrated that the improvement is statistically significant. Additionally, we conducted a thorough error analysis of all the proposed systems which made the roadmap for the future improvements clearer.

Additionally, we performed a quantitative and linguistic analyses of the results. This allowed us to raise the questions about the data available for Estonian, mainly about the inconsistency of annotation. Tackling this issue can potentially improve the lemmatization for existing systems that are trained on the same data.

It is still possible to further improve the lemmatization accuracy of LEXENLEM. First, we can perform additional preprocessing of VABAMORF candidates, like ensuring the conservation of the original tokenization and replacing an empty candidate with random noise. We can also introduce addition layers to the model for better resolving the word formation labelling and learning the part-of-speech and morphological tags jointly thus removing the necessity of a third-party morphological analyzer. Finally, introducing pretrained contextual embeddings as input features can also improve the model's performance.

References

[And82] Stephen R Anderson. Where's morphology? *Linguistic inquiry*, 13(4):571–612, 1982.

[And85] Stephen R Anderson. Typological distinctions in word formation. *Language typology and syntactic description*, 3:3–56, 1985.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BCÇŞ20] Ümit Budak, Zafer Cömert, Musa Çıbuk, and Abdulkadir Şengür. Dccmed-net: Densely connected and concatenated multi encoder-decoder cnns for retinal vessel extraction from fundus images. *Medical Hypotheses*, 134:109426, 2020.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[CVMG^{+}14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Eha09] Martin Ehala. Linguistic strategies and markedness in estonian morphology. *STUF-Language Typology and Universals Sprachtypologie und Universalienforschung*, 62(1-2):29–48, 2009.

[Elm90] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[FGRN^{+}11] Mikel L Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. Apertium: a free/open-source platform for rule-based machine translation. *Machine translation*, 25(2):127–144, 2011.

[For94] Michael Fortescue. Morphology, polysynthetic. *The encyclopedia of language and linguistics*, 5:2600–2602, 1994.

[GEM$^+$19] Rüdiger Gleim, Steffen Eger, Alexander Mehler, Tolga Uslu, Wahed Hemati, Andy Lücking, Alexander Henlein, Sven Kahlsdorf, and Armin Hoenen. Practitioner's view: A comparison and a survey of lemmatization and morphological tagging in german and latin. *Journal of Language Modelling*, 7(1):1–52, 2019.

[GJM16] Vaishali Gupta, Nisheeth Joshi, and Iti Mathur. Design and development of a rule-based urdu lemmatizer. In *Proceedings of International Conference on ICT for Sustainable Development*, pages 161–169. Springer, 2016.

[GS00] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194. IEEE, 2000.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[KGM$^+$18] Jenna Kanerva, Filip Ginter, Niko Miekka, Akseli Leino, and Tapio Salakoski. Turku neural parser pipeline: An end-to-end system for the conll 2018 shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual parsing from raw text to universal dependencies*, pages 133–142, 2018.

[KGS19] Jenna Kanerva, Filip Ginter, and Tapio Salakoski. Universal lemmatizer: A sequence to sequence model for lemmatizing universal dependencies treebanks. *arXiv preprint arXiv:1902.00972*, 2019.

[KKJ05] Kimmo Kettunen, Tuomas Kunttu, and Kalervo Järvelin. To stem or lemmatize a highly inflectional language in a probabilistic ir environment? *Journal of Documentation*, 2005.

[KKM12] Heiki-Jaan Kaalep, Riin Kirt, and Kadri Muischnek. A trivial method for choosing the right lemma. In *Baltic HLT*, pages 82–89, 2012.

[Koe04] Philipp Koehn. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 388–395, 2004.

[KV01] Heiki-Jaan Kaalep and Tarmo Vaino. Complete morphological analysis in the linguist's toolbox. *Congressus Nonus Internationalis Fenno-Ugristarum Pars V*, pages 9–16, 2001.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[LWM$^+$18] Ruizhi Li, Xiaofei Wang, Sri Harish Mallidi, Takaaki Hori, Shinji Watanabe, and Hynek Hermansky. Multi-encoder multi-resolution framework for end-to-end speech recognition. *arXiv preprint arXiv:1811.04897*, 2018.

[MKB$^+$10] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[MMP$^+$14] Kadri Muischnek, Kaili Müürisep, Tiina Puolakainen, Eleri Aedmaa, Riin Kirt, and Dage Särg. Estonian dependency treebank and its annotation scheme. In *Proceedings of 13th workshop on treebanks and linguistic theories (tlt13)*, pages 285–291, 2014.

[MMP17] Kadri Muischnek, Kaili Müürisep, and Tiina Puolakainen. Parsing and beyond: Tools and resources for estonian. *Acta Linguistica Academica*, 64(3):347–367, 2017.

[MVW$^+$19] Arya D McCarthy, Ekaterina Vylomova, Shijie Wu, Chaitanya Malaviya, Lawrence Wolf-Sonkin, Garrett Nicolai, Christo Kirov, Miikka Silfverberg, Sebastian J Mielke, Jeffrey Heinz, et al. The sigmorphon 2019 shared task: Morphological analysis in context and cross-lingual transfer for inflection. *arXiv preprint arXiv:1910.11493*, 2019.

[P$^+$80] Martin F Porter et al. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[Pla99] Frans Plank. Split morphology: How agglutination and flexion mix. *Linguistic Typology*, 3(3):279–340, 1999.

[PLM$^+$04] Joël Plisson, Nada Lavrac, Dunja Mladenic, et al. A rule based approach to word lemmatization. In *Proceedings of IS*, volume 3, pages 83–86, 2004.

[PP15] RJ Prathibha and MC Padma. Design of rule based lemmatizer for kannada inflectional words. In *2015 International Conference on Emerging Research in Electronics, Computer Science and Technology (ICERECT)*, pages 264–269. IEEE, 2015.

[PTJM13] Snigdha Paul, Mini Tandon, Nisheeth Joshi, and Iti Mathur. Design of a rule based hindi lemmatizer. In *Proceedings of Third International Workshop on Artificial Intelligence, Soft Computing and Applications, Chennai, India*, pages 67–74. Citeseer, 2013.

[QDZM19] Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D Manning. Universal dependency parsing from scratch. *arXiv preprint arXiv:1901.10457*, 2019.

[QZZ$^+$20] Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 2020.

[RCL$^+$19] Da Ren, Yi Cai, Xue Lei, Jingyun Xu, Qing Li, and Ho-fung Leung. A multi-encoder neural conversation model. *Neurocomputing*, 358:344–354, 2019.

[Sap49] Edward Sapir. Language: An introduction to the study of speech. 1921. *New York: Harvester paperback, nd*, 1949.

[SL18] Jaehun Shin and Jong-hyeok Lee. Multi-encoder transformer network for automatic post-editing. In *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*, pages 840–845, 2018.

[SP97] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[SSH19] Milan Straka, Jana Straková, and Jan Hajič. Udpipe at sigmorphon 2019: Contextualized embeddings, regularization with morphological categories, corpora merging. *arXiv preprint arXiv:1908.06931*, 2019.

[SSN12] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[Str18] Milan Straka. Udpipe 2.0 prototype at conll 2018 ud shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, 2018.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[TC09] Kristina Toutanova and Colin Cherry. A global model for joint lemmatization and part-of-speech prediction. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 486–494. Association for Computational Linguistics, 2009.

[TSG$^+$10] Reut Tsarfaty, Djamé Seddah, Yoav Goldberg, Sandra Kübler, Marie Candito, Jennifer Foster, Yannick Versley, Ines Rehbein, and Lamia Tounsi. Statistical parsing of morphologically rich languages (spmrl): what, how and whither. In *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 1–12. Association for Computational Linguistics, 2010.

[Vik94] Ülle Viks. A morphological analyzer for the estonian language: The possibilities and impossibilities of automatic analysis. *Automatic morphology of Estonian*, 1:7–28, 1994.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[Wha96] Lindsay J Whaley. *Introduction to typology: the unity and diversity of language*. SAGE publications, 1996.

[XBK$^+$15] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[ZHP$^+$18] Daniel Zeman, Jan Hajic, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. Conll 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual parsing from raw text to universal dependencies*, pages 1–21, 2018.

Appendix

I. Code

The source code for the LexEnLem is available on GitHub by the link bellow:

`https://github.com/501Good/lexicon-enhanced-lemmatization`

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public