FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jan Bodnár

Morphological Segmentation in Czech using Word-Formation Network

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	doc. Ing. Zdeněk Žabokrtský, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2020

I would like to thank my supervisor Zdeněk Žabokrtský for great support and guidance, and to my family for making all this possible.

Title: Morphological Segmentation in Czech using Word-Formation Network

Author: Jan Bodnár

Institute: Institute of Formal and Applied Linguistics

Supervisor: doc. Ing. Zdeněk Žabokrtský, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Morphological segmentation is segmentation of words into morphemes - smallest units carrying meaning. It is a low level Natural Language Processing task. Since morphological segmentation is sometimes used as method of preprocessing, achieving better results on this task may help NLP algorithms to better solve various problems, especially in scenarios involving small amount of data, and it may also also help the linguistic research. We propose a novel ensemble algorithm for morphological segmentation of Czech lemmas which makes use of the DeriNet derivation tree dataset. As a sideproduct we also created suggestions for improvements of the DeriNet dataset.

Keywords: morpheme morphology segmentation

Contents

Introduction

The morphological segmentation is a task of Natural Language Processing involving designing algorithms for decomposition of words into morphemes - the smallest units carrying meaning. For example, the word ”replacement” can be segmented as ”re-place-ment”. ”Re-” is a prefix meaning again, ”place” is the root of the word which carries the main meaning, and ”-ment” is a suffix which forms an action noun out of a verb. From this example we can see how morphological segmentation can help both people and computers to better understand words, especially if they haven’t seen them before. This help is especially important in languages with rich morphology, such as Czech.

In this thesis we propose a novel algorithm for morphological segmentation of Czech lemmas (basic forms of words), which makes use of both manually segmented dataset and derivational dataset DeriNet. (Vidra et al. [2019]). Our algorithm is an ensemble model consisting of a simple rule-based system, neural networks, and a tree propagation algorithm. We describe the ensemble, evaluate it, and we also create suggestions for improvements of the DeriNet dataset regarding which derivation trees should be merged together.

The main complication during the process was caused by the fact that the Czech language contains a big amount of allomorphy, which means that during the derivative and inflective processes Czech not only adds or removes morphemes, but it often also changes the appearance of the remaining morphemes. This results in one morpheme having multiple forms (morphs), which can sometimes be very different from one another.

Motivation

There are two main reasons why to improve the morphological segmentation algorithms. The first reason is that a good automatic morphological segmentation opens space for linguists who work with corpora to examine certain morphological phenomena in larger scale, and thus allows to broaden our knowledge of languages. Examples of such research can be exploration of the relation between morpheme boundaries and syllables, historical changes of morpheme use, or examining situations in which language prefers one variant of morpheme over another (allomorphy).

The second reason is that morphological segmentation may improve the behavior of other natural language processing algorithms, especially on morphologically rich languages and in scenarios involving small amount of data. It is known to help the statistical algorithms, and to improve the machine translation where we lack sufficient amount of parallel data. This is likely because the morphologically segmented text offers the information in a more structured way, which helps to tackle the sparsity problems in learning, and allows for inference of meaning of unknown words or word forms. Morphological segmentation is also closely related to lemmatization and stemming, which are traditional ways of text pre-processing.

The morphological segmentation or morpheme-aware approaches may be beneficial even for a modern deep learning methods, since even such methods with huge amount of data suffer from out of vocabulary words, or may exploit the knowledge of the word's structure during the training. As an example we may mention the successful subword-level word embedding (Bojanowski et al. [2016]), which forms the word embedding on the basis of all the sub-strings of a word. Such an approach may be considered rough in comparison with just operating with word's morphemes, but it seems likely that with sufficient amount of data and computational power, the model selects just certain meaningful sub-strings, and thus builds its own version of "morphological segmentation" - not necessarily based on non-overlapping strings, and not necessarily respecting our segmentation, since e.g. grouping frequently co-occurring suffixes into one group with a clear meaning may do more good than bad in practice.

This type of argumentation may be also used as an argument to support the application of unsupervised or minimally-supervised methods of morphological segmentation to under-resourced languages even in cases when they quite differ from the linguistic point of view, since even "wrong" segmentation may still be useful in practice.

Outline

In the thesis we first examine the related work, especially with regards to other general approaches used to tackle our problem and to the datasets we use, then we continue with discussion of the theoretical background of our approach both from the linguistic and AI perspective. After this we derive and describe the proposed solution, based on the ensemble model involving rules, derivation tree processing algorithm, and neural networks partially trained on the artificial data. Later we perform the experiments evaluating the model's performance, and justifying its structure. In the Appendix we show examples of the algorithm's output and examples of suggestions for improvements of the DeriNet dataset.

Prior publication disclaimer

Dislaimer:
Some of the original results developed while writing this thesis will also be published in

Jan Bodnár, Zdeněk Žabokrtský, Magda Ševčíková, Semi-supervised Induction of Morpheme Boundaries in Czech using a Word-formation Network, In: Proceedings of the 23rd International Conference on Text, Speech and Dialogue - TSD 2020.

The results which appear both in the thesis and in the paper are my sole work unless explicitly stated otherwise.

1. Related Work

1.1 Approaches towards the problem

We first examine the methods used by different authors.

1.1.1 Multilingual approaches

Computational linguists do not want to focus solely on the big common languages. They also want the small under-resourced languages to benefit from the advances advances of the field. Therefore there has been a big focus on techniques which require only a very small amount of annotated data and preferably just small corpora.

An example of this trend is a work Snyder and Barzilay [2008] where authors experiment with simultaneous unsupervised learning of morphological segmentation on multiple related languages in the same time. They create a Bayesian model which learns both morphemes specific to each of the languages, and morpheme correspondences to join two morphemes from different languages with similar meaning but a different form. They have also modified the model to prefer the correspondences among closely phonologically related morphemes. They reported that the model trained on Arabic and Hebrew shows significant improvements over the monolingual models.

1.1.2 Chinese sentence segmentation

The problem of segmentation of Chinese sentences into words is non-trivial since Chinese does not make spaces between words. This problem may seem unrelated with the morphological segmentation at first, but we should bear in mind that both tasks consist of taking a groups of characters, be it sentences or words, and segmenting each of them into subgroups which commonly co-occur in the language.

This is the main analogy which motivates us to look at the Chinese segmentation as a source of inspiration. Although we cannot hope to directly transfer the used algorithms, to morphological segmentation, since there are e.g. much more Chinese characters than letters in our alphabet, which means that a common co-occurrence of Chinese characters tells us much more. There are also much more sentences than words, which means that we see a single word many times, while we only see most of the sentences once.

An example of an algorithm used for this task is in Sproat et al. [1996] it is an iterative algorithm which relies on a combination of a word dictionary with statistical methods. The authors use an existing dictionary which they expanded by the addition of certain character combinations on the basis of their Mutual Information in the corpus. Then they run an iterative algorithm, which in one step estimates the frequencies of words in the corpus, and sets cost of each word's occurrence in a sentence as

$$Cost = -log(\frac{wordfrequency}{dictionarysize}) \tag{1.1}$$

then they segment a sentence in such a way that its total cost is minimal, and again re-estimate the frequencies.

1.1.3 Minimum Description Length and Maximum A Posteriori probability

The minimum description length is closely related to the compression. Imagine that you want to compress a dataset of words. Then it may be worth it, to simply memorize the most common sub-strings of words and always reference them, instead of repeatedly writing these sub-strings letter by letter.

More formally we can say that you first develop a class of encoding models, and then you try to find the encoding model which encodes your dataset into the smallest number of bits, while also counting the size of the encoding model's parameters. Then this trained model in a way represents the most natural representation of words, and the way of encoding of a certain word is expected to correspond with its morphological structure.
The optimized formula is

$$\Theta = min_\theta : (L(data\|\theta) + L(\theta))$$

. We sometimes see a correspondence with the Maximum A Posteriori probability method of training models, where we select parameters of the model in such a way that the likelihood of the parameters given the data is the highest:

$$\Theta = max_\theta : P(\theta\|data) = max_\theta : P(data\|\theta) * P(\theta) =$$
$$= min_\theta : -log_2(P(data\|\theta)) - log_2(P(\theta))$$

where $P(\theta)$ is the prior probability of our model having certain parameters. The correspondence can be looked for as soon as we realize that $-log_2(P_Y(X))$ is equal to the lowest possible number of bits "the best" compression method needs to encode a random sample X from a known distribution Y. With "the best" we mean a compression method which, with knowledge of Y, has the lowest expected value of bits necessary to encode a sample from Y. The "best" is in quotes because we speak about the theoretical lower-bound for compression established by Shannon (Shannon [1948]), rather than about the actual encoding, which may achieve worse results due to rounding.

We then know that two MDL and MAP models are equivalent if it holds that:

$$L(data\|\theta) = -log_2(P(data\|\theta))$$
$$L(\theta) = -log_2(P(\theta))$$

or rather we can use this relationship to attempt to derive the other form of a model, and get some deeper insight into the model's behavior, or even improve certain aspects of the model.

Please note that our description of the correspondence has certain mathematical limits. For instance, we did not sort out the question of correspondence of real valued parameters, because in such case $P(\theta = x) = 0$. In such case we may consider a limited precision representation, but then we may have the problems with analysis of the MAP model, etc.

Morfessor Baseline

The recursive algorithm of the Morfessor Baseline (Creutz and Lagus [2002]) is based on the minimum description length. The cost is the sum of dictionary length and the length required for encoding the input sequence of morphs with regards to a probabilistic model.

$$Cost = \sum_{i}(-log_2 p(morph_i)) + \sum_{j \in dictionary} k * L(morph_j)$$

where k is the number of bits necessary for encoding a character (in this case $k = 5$), and $l(m_j)$ is the length of j-th morpheme in the dictionary. This means that the underlying model of morphology is a probabilistic uni-gram model, which has likelihoods of each morpheme, but not e.g. likelihoods dependent on the context.

The model is then trained with a custom algorithm, which iteratively re-segments the words and updates the dictionary.

Linguistica

The algorithm Goldsmith [2006] also works on the minimum description length principle, although it employs much more complex model than Morfessor Baseline. It again describes the length of the representation of the model directly, while using the "best possible encoding" notion to describe the minimum length required to describe the dataset on the basis of the learned probabilistic model.

its "compression" model is based on a dictionary of "stem-groups" and "suffix-groups" (stem is simply the first part of the currently segmented part of the word, while suffix is the second part). It contains pairs consisting of a "stem-group" and a "suffix-group",such that each "stem" from the group of "stems" was seen with each "suffix" from the group of "suffixes". The algorithm first tries to fill this dictionary in such a way that it helps to compress the data, and then it attempts to clean the dictionary up, so that only linguistically plausible affixes remain.

1.1.4 Expectation Maximization

The Expectation Maximization (EM, formalized in Bishop [2013]) is an iterative meta-algorithm used for training of probabilistic models in an unsupervised or semi-supervised fashion. It has broad applications all around the field of artificial intelligence. In morphological segmentation EM has been applied to the morphological segmentation of Czech (Vidra [2018]), as well as of several other languages (Creutz and Lagus [2004]) and it has been recently proposed as a novel training algorithm for the Morfessor Baseline (model: Creutz and Lagus [2002], new algorithm EM+Prune: Grönroos et al. [2020]). Even the aforementioned algorithm for segmentation of Chinese sentences to words may be considered a simple model trained with EM.
The EM training has the Initialization phase and then the Expectation and Maximisation phases which repeat until convergence.
1) Initialization:
Before we can run the Expectation phase for the first time, we need to initialize the parameters of the probabilistic model. This phase can be very tricky, since

the quality of the initialization can highly affect the quality of the outcome, as has been well documented e.g. in the mixture model training. At the same time it is not clear how to properly initialize the model, since the best possible initialization is solving the whole problem in advance, and just setting the parameters to the right values. The commonly used initialization strategies include use of small amount of annotated data (known to help in morphological segmentation even with a few hundred samples Ruokolainen et al. [2016]), using domain specific heuristics (common substrings as candidates for morphemes or K-Means initialization of GMM), setting certain distributions to uniform, and using randomly generated values or random sampling.
2) Expectation: In the expectation phase we use the model to predict latent discrete variables of the data samples. E.g. in the Gaussian Mixture Model (GMM) clustering we want to predict the likelihood that each data sample belongs to each cluster on the basis of the current parameters of the Gaussian components and their priors. In the morphemological segmentation problem we would ideally like to derive the likelihood of each morphological segmentation of each word on the basis of the current segmentation model.
3) Maximization: During the maximization phase we use the segmentation or the clustering of the dataset to re-estimate the numeric parameters of probabilistic the model - we use the maximum likelihood principle and set parameters to values which maximize the likelihood of the model generating the data (the clustering, the segmentation) we observe. In the GMM model we need to recompute means, (co)variances and priors of all the Gaussian mixtures to best fit the segmentation. If we think about it, the Maximum Likelihood (ML) estimation of GMM parameters is surprisingly simple. The ML estimate of the cluster prior is basically sum of likelihoods that each datapoint lies within that cluster, while mean and covariances can be simply estimated by closed form formulae from all the data points weighted by their likelihood of being generated by the given cluster.

This surprising simplicity of the GMM training is a common property of many algorithms trained via EM, because they were designed so that both estimations highly simplify.

Now we would like to demonstrate what a real-world EM-based solution looks like on a linguistic problem. We want to show its behavior on something similar to the morphological segmentation problem, but we do not want to get into all the details and peculiarities of the more complex models. Therefore we will once again look at the aforementioned Chinese segmentation algorithm and describe it in the terms of EM (Sproat et al. [1996]).

First we create our generative story: likelihood of our dataset being generated is equal to the product of likelihoods of individual sentences. Each sentence is composed of words contained in our fixed lexicon. Each word has its own, context independent, probability of appearance in a sentence. We further assume that the likelihood of a sentence having k words is equal to

$$P[sentencelength = k] = constx \frac{1}{dictionary\_size^k}$$

This is a an example of a uni-gram model. Such models are not very grammatically plausible, because they do not incorporate context, but they are simple to deal with, and sometimes can yield good results. The exponentially decaying probability of sentences being longer may seem arbitrary but it is here to prevent divergence towards segmenting everything what could be possibly segmented. The concrete value of coefficient forms a threshold - words w_1 and w_2 will be merged together into word w_3 if

$$freq(w_1) * freq(w_2) < freq(w_3) * average\_word\_frequency$$

, which seems reasonable.

In the expectation phase we segment the dataset with our uni-gram model. We look at each sentence, and find its Most Likely segmentation, which means finding the words $w_1, w_2, ..., w_k$ such that concatenation of these words is the whole sentence, and that the product of likelihoods

$$P[sentence] = P[sentence\_length = k] * \Pi_{i=1}^{k} model\_likelihood(w_i)$$

$$P[sentence] = const * \Pi_{i=1}^{k} (unigram\_likelihood(w_i) * \frac{1}{dictionary\_size})$$

is maximal. Since the positive constants do not matter in maximization, we can just ommit const, and after setting

$$cost = -log(unigram\_likelihood(w_i) * \frac{1}{dictionary\_size})$$

,

this problem may be transfered to the problem of finding shortest path within directed acyclic graph of characters forming the sentence. Please note, that in this phase the algorithm differs from the true EM, where we compute the probabilistic distribution over all the possible segmentation, and not just the most likely segmentation. This variant of EM may be called "winner-take-all EM (Neal and Hinton [1998]), and is used also in other algoritms, such as Morfessor CatML (Creutz and Lagus [2004])

The maximization phase then attempts to find the best model, namely the model, from the selected class of models, which has the maximum likelihood of generating the current segmentation (or, in the standard EM, the distribution of segmentations). Finding of such a model is surprisingly simple. The best unigram model is the model which assigns all the word likelihoods equal to number of their occurrences within all the sequences (unigram_likelihood = word frequency in the current segmentation of dataset). This means that the cost above changes to

$$cost = -log(\frac{word\_frequency}{dictionary\_size})$$

which is exactly the formula from the paper.

The initialization is then done by setting number of occurrences of each word to the total number of occurrences its string form has inside the corpus.

Of course, this is an idealized version omitting some details, such as the question of out-of-vocabulary words, and it we described a very simple model, but it still can give us a better idea of how a linguistic model can be structured in order to work with EM.

1.1.5 Bayesian Methods

The Bayesian methods have been broadly used in natural language processing, and they have been applied to related morphological segmentation (e.g. Snyder and Barzilay [2008]) and a somewhat related task of segmentation of English sentences to words (Goldwater et al. [2009]).

They may be considered overlapping with other statistical approaches, such as methods centered around Expectation-Maximization or the Maximum A Posteriori Probability approach, since all tree methods use statistical models, and therefore both previously mentioned methods are equivalent in terms of the expression power (given we are able to compute the models, which may not always be the case with EM). But we still think that Bayesian methods should have a group of their own due to slightly different approach and methods involved.

The general Bayesian approach is defining a complex generative story, centered around the Bayes theorem, conditional probability and prior distributions, for instance we may set prior distribution to number of morphemes, so that we prefer simpler models, or prior distribution to morpheme frequencies (a few are frequent, many are infrequent), which will result in the model trying to develop in a naturally looking direction and possibly in a further reduction of overfitting. Knight [2009].

The generative stories should ideally as linguistically plausible as possible, but on the other hand we still need to think about the computability, and possibly look for places where our model may be simplified without doing too much harm.

Bayesian approach is usually connected with its own set of computational methods, such as Gibbs sampling, but it can also use Monte-Carlo methods.

1.1.6 The Follower Surprisingness

Another approach to morphological segmentation is creation of a model, which tries to predict the following part of the word. The assumption is that the model will be the most unsure on the boundaries between morphemes. The approaches in this paradigm range from early experiments with n-gram models Harris [1955] to somewhat similar experiments done while testing early recurrent neural networks - in Elman [1990] the author trains a network to predict the following character in a sentence and measures its sureness in any given step.

1.1.7 Deep Learning

In the field of deep learning, there have been previous experiments with learning of morphological segmentation in the supervised fashion (e.g. Vidra [2018], Wang et al. [2016]), as well as with creation of the so called sub-word embeddings: in Bojanowski et al. [2016] authors create a word embedding on the basis of all the word's sub-strings without caring which of them are correct morphemes, and which are not, expecting that the model will learn to handle the situation. This embedding is successful since it leads to performance improvements and a better handling of out-of-dictionary words, and therefore we can conclude that the subword approach remains to be useful, eventhough there may be a paradigm shift in it.

1.2 KonText Tool

KonText is an advanced search tool developed by the Institute of the Czech National Corpus. It allows to run complex search queries in the Czech National Corpus. Each query may combine matching words with regular expressions with filtering on the basis of information contained within the word's tags (e.g. case, number, ...) and it can even search on the basis of the context in which the word appears in the corpus.

Thanks to Mgr. Michal Křen, PhD. who kindly integrated our experimental outputs of segmentation into the KonText tool we were able to use it for debugging of our segmentation. It allowed us to look for certain patterns which our segmentation algorithm may be doing wrong, and thanks to the corpora frequencies integrated into the tool it gave us a different perspective about the mistakes we make because we could also focus on the behavior on the most common words.

1.3 Datasets

1.3.1 Manually segmented dataset

In the thesis we also used a dataset containing 2100 manually segmented Czech lemmas sampled from the DeriNet (Vidra et al. [2019]) and manually segmented by Zdeněk Žabokrtský and Šárka Dohnalová.

While manually creating a segmentation dataset authors run into an issue involving the question which words to actually choose. Should they select each word with the same probability, or should they sample on the basis of the corpora frequencies? In the end the dataset consists of 1000 words sampled in the first way and 1100 sampled in the second way, on the basis of word frequencies in the SYN2015 corpus (Křen et al. [2015]).
As a demonstration, we show 20 samples from each group:
The first method of sampling:

ulehlina, vybíračův, Vokurovský, Pillerův, synovskost, zdající, učenlivost, dymník, achronisticky, zigar, vláhonosnost, usoustruhovávatelnost, vydrobenost, Obodrit, asymptoticky, odpřeložitelnost, Vosejpkův, Blažeková, troštovávající, debutovatelnost

The second method of sampling:
aspoň, germanistika, povinný, vyspělý, parkoviště, milostivý, povolaný, vkus, zapuštěný, ministerstvo, vzdělání, akcie, formát, vztek, důraz, kožešinový, schod, post, chuť, pouze
We may conclude that the first method of sampling results in sampling of less frequent words, often containing a high number of morphemes, while the second method resulted in sampling of much more common, and much shorter words.

We think that it is likely better to sample the words in the second way, since the first approach sometimes selects foreign, or strange words. On the other hand, the first method also selects many regular, long words consisting of many morphemes, which is exactly what we need for the morphemic segmentation.

Our general approach to sampling also resulted in the imbalance of part of speech groups (nouns, adjectives, verbs, adverbs), with nouns being more than half of the dataset, and adjectives being big portion of the rest. Up to an extent this imbalance is a positive thing, since it corresponds with the number of each part of speech in the target texts and thus allows us to focus on the most important groups, but it would have been better to keep more control about the part-of-speech ratio, since the current imbalance complicates certain types of analysis.

1.3.2 Retrograde Morphemic Dictionary of Czech

We have also used the verbs from Retrograde Morphemic Dictionary of Czech (Slavíčková [1975]). We used the data digitized via OCR with manual cleanup. Since the verbs in this dataset end with the old Czech infinitive form ending ”-ti” (e.g. dělati, přijíti), we have replaced it with the standard ending ”-t” (dělat, přijít).

The Retrograde Dictionary is a dictionary of Czech where the words are ordered in alphabetic order, but on the basis of the last letter, then second but last letter, etc. This dictionary was created by its author to serve the morphological research, since the words with similar endings, and similar suffix morphs, are grouped together. The dictionary also contains the list of Czech word roots with all the allomorph variants, list of the most frequent Czech morphemes, list of the most common triplets of prefixes, etc.
We show a sample of data as an example (first read the first column, then the second, then the third):

o-lup-ova-ti	po-stup-ova-ti	roz-čar-ova-ti
s-lup-ova-ti	pro-stup-ova-ti	dar-ova-ti
vy-lup-ova-ti	u-stup-ova-ti	ob-dar-ova-ti
vy-lup-ova-ti	v-stup-ova-ti	po-dar-ova-ti
roz-lup-ova-ti	vy-stup-ova-ti	de-klar-ova-ti
od-šup-ova-ti	roz-stup-ova-ti	na-par-ova-ti
se-šup-ova-ti	štup-ova-ti	od-par-ova-ti
po-po-šup-ova-ti	za-štup-ova-ti	/re/par-ova-ti
o-tup-ova-ti	pře-kyp-ova-ti	/pre/par-ova-ti
na-stup-ova-ti	od-ryp-ova-ti	/se/par-ova-ti
za-stup-ova-ti	pře-syp-ova-ti	roz-par-ova-ti
před-stup-ova-ti	čar-ova-ti	var-ova-ti
pod-stup-ova-ti	za-čar-ova-ti	havar-ova-ti
pře-stup-ova-ti	od-čar-ova-ti	tvar-ova-ti
se-stup-ova-ti	při-čar-ova-ti	vy-tvar-ova-ti
roze-stup-ova-ti	o-čar-ova-ti	u-var-ova-ti
při-stup-ova-ti	u-čar-ova-ti	vy-var-ova-ti
do-stup-ova-ti	vy-čar-ova-ti	pár-ova-ti

Table 1.1: Data sample from the Retrograde dictionary Slavíčková [1975].

1.3.3 DeriNet

The whole method highly relies on the DeriNet dataset (Vidra et al. [2019]) (namely on versions 2.0 and 2.1beta). DeriNet is a derivational network of Czech lemmas. It consists of groups such that each word in a group originated by a derivative process from a different word in the same group (with exception of one main word, from which all the other words originated). Such a group may for example contain words délka, dlouhý, prodloužit (length, long, to make longer in Czech). Each group is structured as a rooted tree where each child node originated via a derivative process from its parent. The dataset also contains information about part of speech for each word.

Since Czech is quite a productive language in terms of morphology, the DeriNet contains one million lemmas in around 200 thousand trees. The number of words in each derivation tree is highly variable, ranging from just a single word to thousands of words, such as in case of word ”číst” (to read), ”čtenář” (reader), ”přečíst” (to read through), ”předčítat” (to read to others)

1.3.4 Morpho Challenge datasets

Morpho Challenge was a semi-regularly held shared task organized with the goal to support the development of morphological analysis of words. It contains datasets for several languages, including English, Finnish, German, Turkish, and Arabic. It contains lists of words with numbers of occurrences within a corpora, as well as a small amount of manually annotated data to enable researcher's own evaluation. Unfortunately, the golden data are not precisely suitable for a morphological segmentation task, since they do not contain the precise morphological segmentation, but rather an analysis of the word - morphemes are in their basic forms, and instead of some morphemes there are tags such as +PL, meaning that a word is in the plural form. For example ”indoctrinated” is described as ”in_p doctrine_N ate_s +PAST”.

2. Theoretical Background

2.1 Linguistic background

2.1.1 Czech Language

From the systematic perspective, the Czech language belongs to a group of the so-called inflected languages. This means that the Czech language forms words by combining morphemes, but that certain morphemes carry multiple meanings, such as the inflective ending ”ův” in ”Davidův” (David’s thing of male gender in the first case of singular). This means that changing the number may result in a complete change of a morpheme or into one morpheme’s transformation into two different morphemes, instead of simple addition of the plural morpheme (as is usually the case in English).

Czech morphology is very rich, and the affixes are used to express many different meanings: number, gender, negation, aspect, they can be used for adverbisation, to produce agent nouns, or names of places (”čekat”-”čekárna” - ”to wait”→”waiting room”, ”lék”→”lékárna” - ”drug”→”drug store”), there are prefixes commonly used for the formation of language names (Finsko-Finština, Česko-Čeština, Čína-Čínština - Finland, Czech Republic, and China with the names of corresponding languages). Czech also has a big variety of prefixes that just shift the meaning of the word (such as prefixes re- or de- in English).

2.1.2 Morphemes

The single most important linguistic notion for this thesis is morpheme. According to the definition in Čermák [2011], morphemes are the smallest, further indivisible, semantic parts of a word. Morphemes can be either auto-semantic, carrying meaning on their own (e.g. root of the word), or syn-semantic, having meaning only in context (inflective endings). This distinction also shows two ways how the morphological segmentation can help the Natural Language Processing algorithms - it can help them to better understand a meaning of the word itself, or possibly even derive a meaning of an unknown word, and it can help them to better understand the meaning of a word within a sentence, especially in very inflective languages.

Yet the concept of morpheme by itself is not that simple. If we start to examine morphemes in Czech, we will soon realize that a single morpheme can take several more or less related forms. This leads to the distinction between morpheme and morph. While morpheme is an abstract concept usually described by a prototypical example, morph is its concrete realization in text. Dokulil [1962] For instance, the root morpheme ”roz” may take form ”roz” in word ”rozbít” (to break) and form ”roze” in word ”rozebrat” (to disassemble). All the morphs derived from the same morpheme are called allomorphs.

There are further ways of classification of morphemes:
1) There is a distinction to inflectional and derivational morphemes, which is directly related to the distinction between inflection vs derivation as well as with all the problems connected with it.

2) We can distinguish between free and bound morphemes - free morphemes can stand alone as a single word, while bound morphemes only exist jointly with other morphemes. Please note that this distinction does not completely correspond with the distinction between prefixes/suffixes and roots since there are roots which can only exist with other morphemes.
3) classification on the basis of position in the word:
The morphemes in front of the root are called prefixes, the morphemes behind it are called suffixes. In the case of word compounds, there can also be an interfix, which connects the two words. A special type of morpheme is circumfix, which has its parts on two positions, e.g. the first prefix, and last derivational suffix. Examples of circumfixes are e.g. po-...-í in pobřeží (seaside) and poříčí (riverside), ná-...-ík in nárameník (shoulder board), and pa-...-ek in pahorek (hillock). (Dokulil [1962], Čermák [2008]). In some cases, it may not be perfectly clear whether a word was formed by circumfixation or just as by independent addition of two morphemes - word namodralý (blueish) could have been formed either via circumfixation as na-modra-lý or via consequent addition of prefixes as modrat→namodrat→namodralý. In this case, Čermák (Čermák [2008]) argues that we should not speak about the pure possibility of the grammatical derivation but that we should also consider the actual usage in language: since the word namodrat is almost never used, the derivation via namodrat is not a plausible explanation and therefore the word namodralý should be considered a case of circumfixation.

There are also certain special types of morphemes, such as the so-called cranberry morphemes, which are bound morphemes without any meaning or grammatical function, but which still affect the meaning of the word. This type of morphemes was named after the word cranberry where -berry is yclearly a morpheme since there exist words such as strawberry or blackberry with related meaning, and therefore the cran- also has to be a morpheme. But cran- has no meaning of its own, because it only appears in the word cranberry. Czech example of this phenomenon is e.g. "-kňub-" in "nekňuba". Ševčíková et al. [2019]

2.1.3 Allomorphs

In Czech, as well as in other inflected languages, allomorphy is frequent. Sometimes, the changes of the morphs may be regular. For example vowels in prefixes na-, vy-, při- often take short form in verbs but take long form in nouns: "nahradit"-"náhrada (to replace-replacement), vystoupit-výstup (to disembark-disembarking), připravit-příprava (to prepare - preparation), although, as always in the linguistics, this rule has its exceptions, as e.g. in nastavit-nastavení (to set up - setting up). Another common change is alternation of length of vowel in roots, as in hlas-hlásek, žába-žabka (voice and frog with their diminutive forms), or řidič-řídit (driver-to drive). Dokulil [1962] But there are also words which do not alternate, such as pes-pejsek (dog-doggie) or učit-učitel (to teach-teacher). The changes are not limited to the diacritics but can be much more variable and broader, such as in dům-domek (house-small house), kouřit-kuřák (to smoke-smoker), or den-dny (day-days). Dokulil [1962]

Allomorphy of roots of words may sometimes take extreme forms which even brings a question whether it still is a single word. An example could be e.g. jít-šel (to go - he went, which shows the same behavior in English) (Čermák [2011]), dobrý-lepší (good-better), je-byl (he is - he was) or psát, dopsat, píše, piš, písař, nápis (words derived from psát - to write).

After mentioning that a single morpheme may take many, often very varying, forms we should also mention that a single string of letters may be morph of several morphemes, which sometimes worsens the interpretability of word even after the segmentation. An example is a Czech flective ending of pronouns -í which is used with many different combinations of number, case and gender and hence gives us almost no information about the word form.

2.1.4 Word formation processes

There are multiple definitions of the word formation processes, often differing in the number of considered linguistic phenomena. According to Štekauer et al. [2012] we may distinguish between the following groups of word formation:

1. Affixation
   Addition, removal or change of affixes
   e.g. skok→skokan (jump→jumper)
2. Combination of word roots
   (a) Compounding
       Joining two words (roots) together
       e.g. poločas (Half-life)
   (b) Reduplication
       Duplication of the word e.g. černočerný (literally "blackly-black")
   (c) Blending
       Similar to compounding, but we only join part of the first word with part of the second word.
3. Without addition of derivational material
   (a) Conversion
       Change of part of speech without addition of morphemes
   (b) Stress, Tone
       Not common in Czech. Possibly proudit vs. proudit (to flow vs. to smoke meat thoroughly)

The affixation could be further separated into derivation and inflection. Derivation is a word forming process during which we modify the meaning of a word itself, while with derivation we only change the grammatic meaning of the word. For example, the following transformations are cases of derivation: "běh"→"běžec" (a run → a runner), "hladký" → "hladce" (smooth → smoothly), while the following are examples of inflection: "pes" → "psi" (dog-dogs), "hraje" → "hrál" (he is playing → he played), v autě → do auta,z auta,k autu (inside a car -¿ into a car, from a car, to a car). To problematize this distinction a little, we have to point out that even though the difference between derivation and inflection is clear most of the time, it may sometimes be blurry, such as in case of negation. The set of all inflectional forms of a word is called lexeme, and it is usually represented by a word called lemma, which belongs to the lexeme and is in a standardized form chosen by convention. E.g. lemma of a verb is its infinite form both in Czech and English.

2.1.5 Derivational graphs

By opening the topic of derivation, we shifted from discussion of internal structure of a single word to the discussion of relationship between two words. This opens a natural question which word was derived from which. For doing this decision Ševčíková et al. [2019] suggests the following criteria: - the original word should have simpler morphological structure
- the original word should have broader meaning

Although in certain cases the direction of derivation still may not be clear, such as while comparing a noun and a verb which differ just by the part-of-speech change. This becomes an issue when we try to organize the derivations into a derivational graph. Either we need to accept that the graphs may contain cycles, or we have to create a rule about how to sort such cases out. If we want to build derivational trees, as is the case in DeriNet (Vidra et al. [2019]), the problem increases, because we have to look for a single word from which was our word derived. This may be problematic for words with multiple derivation parents, such as word povyskočit (to jump), which can be derived both from vyskočit and poskočit which were both derived from skočit (to jump).

2.2 Neural Networks

2.2.1 Basic Model

Neural networks are a class of machine learning methods, nowadays mostly based on the models consisting of the Perceptron (Rosenblatt [1958]) neurons.

Perceptron is one of the simplest neuron architectures. For inputs
$x_1, x_2, ..x_{n-1}, x_n$
the output of neuron is
$f(w_1 * x_1 + w_2 * x_2 + .. + w_n * x_n + b)$,
where $w_1, ..., w_n$ and b are trainable parameters of each neuron and f is a so called "activation function". Examples of such functions are e.g. sigmoid $f(x) = \frac{1}{1-e^{-x}}$, Hyperbolic tangent $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$, or ReLU $f(x) = max(0, x)$. (Goodfellow et al. [2016]) Since the training algorithm requires the activation functions to have the first derivatives in all points, which ReLU does not fulfill, we artificially define its derivative in zero as either 0 or 1, depending on implementation.

The most common way how to organize perceptrons in a neural network is the form of a multilayer perceptron (Rosenblatt [1963]) - a layered model with fully connected layers: each layer $l[i]$ consists of one or more neurons. Each neuron of layer $l[i]$ takes all the outputs of the layer $l[i-1]$ as its inputs. The neurons from the layer $l[1]$ take the input data as its inputs. The output of the model is then the output of the last layer.

The following model is then trained via gradient descent - we define a differentiable loss function, such as the Mean Square Error loss function defined as $loss(data, labels) = \frac{1}{n}\sum_{i=1}^{n}(net(data_i) - labels_i)^2$
where $net(d)$ is the operation performed by the neural network on the data sample d - it is basically output of the last layer of the network, after processing the input d.

The training is then performed in such a way that we compute a gradient (with regards to all the weights and biases of individual neurons) of the loss function in point $(weights, biases, data, labels)$. We then shift all the weights and biases by a little step in the direction of the minus gradient. This process is repeated as long as the loss of the neural network keeps decreasing on a separate data sample, which is not used for direct training. The size of the step is called learning rate and is usually decreased during the training.

The gradient descent optimization can be further improved by adding certain amount of inertia to the gradient to stabilize it among iterations, or by allowing it to adapt its step size separately in each dimension. Such modified algorithms (e.g. Adam Kingma and Ba [2014] or RMSprop) usually achieve faster convergence towards an optimum than the standard Stochastic Gradient Descent.

Here we write "an optimum" and mean a local optimum, since the surface of the neural network contains many local optima and we therefore cannot guarantee that we reach the global one.

Fortunately, we do not need to reach the global optimum. Contrarily, if we reached it it could cause our network to over-fit - to adapt and focus too much on the specifics of the training dataset instead of extracting the general relation we care about. The model could, for instance, learn to identify all the images on the basis of the pixels in the top left corner instead of attempting to recognize the objects depicted.

2.2.2 Neural networks and language

Unfortunately, the standard architecture has a disadvantage which limits its use in Natural Language Processing - it accepts only constantly long input data, we are unable to transform words to constantly long vectors in any meaningful way. Padding is a bad idea in this case. The whole word would shift by a few characters when we e.g. add another prefix and the network would not be able to handle it. Therefore we have to use different architectures for learning of the word segmentation. The two architectures commonly used for this purpose are Recurrent Neural Networks (RNN), and Convolutional Neural Networks (CNN).

As the first approximation, the recurrent neural network could be seen as a box consisting of fully connected networks, input of which are two vectors - the previous state, and the current input (e.g. the current word, or the current letter), and its output is the next state and the current output (e.g. information, whether there should be a boundary behind the current letter).

Yet in reality it is not this simple. In past there have been experiments with recurrent neural networks that only had one part - a big feed forward network which got concatenation of input and the previous state and outputted the current input and the next state. Unfortunately these networks had problems with the size of the gradient - either it vanished (its size went to zero obscuring the way where the weight should be moved), or it exploded (went to high, and thus caused too large updates of the network). In order to tackle the gradient size problems a novel architecture appeared - LSTM (Hochreiter and Schmidhuber [1997]). Later, GRU (Cho et al. [2014]) appeared as its simplified variant.Goodfellow et al. [2016]

Convolutional neural networks work on a different principle. They consist of several filters. Each filter is applied to all positions in a sequence, always taking several neighbor members of a sequence as its inputs. E.g. on sequence $x_1, x_2, ..., x_n$ with filter of length 4 we first process $x_1..x_4$ and produce an output, then $x_2..x_5$, then $x_3..x_6$, etc. The output of k filters on sequence of length N has dimension $N * k$, or also $(N - const) * k$, depending on how we handle the shifting of filters to the ends of sequence. If we process a sequence $x_1..x_n$ we may either start with processing $x_1...x_4$ and finish with $x_{n-3}...x_n$ or start with x_{-2}, x_{-1}, x_0, x_1 and end with $x_n..x_{n+3}$, with additional elements of the sequence being padding, often zeros. Both approaches may be useful depending on the application. In segmentation we need the second one since it allows to segment even the first letter alone.

On more dimensional data, filters basically slide along one or more dimensions, and handle the remaining dimensions as constantly long. Otherwise the convolutional filter works as a standard perceptron - for inputs x1..x4 it produces output $f(w_1 * x_1 + .. + w_4 * x_4 + b)$.

We need to further elaborate on the way how to feed a character into the network. The first idea - to translate characters into numbers (a=1, d=4....) and feed them into network like this does not work. The main problem is that the network would have to learn that number 6 (f) is not "something between" 5(e) and 7(g). That would highly extend the training if the network would ever be able to learn it. Because we cannot feed the letters into the network as numbers, we either need to use the one-hot encoding (the letters are encoded as constantly long vectors consisting of all zeros and one one - e.g. the fifth letter is encoded as a vector with one on the fifth position and zeros everywhere else), or we may use the more efficient Character Level Embeddings (each letter is assigned a trainable vector) which requires network to have much less input parameters on the first layer and allows the network to train letter level features used among the letters. Unfortunately, this encoding has one big disadvantage - the network cannot handle unknown letters (e.g. ö). This is not generally considered a big issue, since the letter can either be replaced with a closely related letter (o) or a pair of letters (oe), or, if the letter cannot be easily replaced, it is so different from anything else, that the network would not be able to work with it anyway. But if it is really necessary, we could also use a trainable vector meaning "unknown letter". It may also be useful to add two special characters - beginning of word and end of word to help the algorithm learn that something only happens in the beginning.

2.2.3 Triplet loss

Until now we spoke about using neural networks in the context of classification - they receive input and compute a single number or numbers, which are then interpreted as classification verdict(s). Yet there are also more advanced ways how a neural network can be used. There is a notion of the representation training, where our goal is to train a neural network to output a vector of numbers with a specific meaning.

We may for instance create an embedding network, which maps input samples into a special n-dimensional space where two input samples lie close to one another if and only if they are in some sense similar in the original space. The meaning of the word ”similar” is decided while training. It may be e.g. mapping faces of the same person close together, and faces of different persons far apart, which can be later used for the face recognition; or it may be mapping of the words with the same root together; or mapping of similar songs together; or anything else, depending on our interest. The only requirement for the training is that we have enough of these similar and dissimilar examples.

The triplet loss model (Schroff et al. [2015]) used for face recognition is trained on batches of triplets, each of which consists of two positive samples a, b (two samples which we want to be close together) and one negative sample c, which should be far apart.

The triplet loss then optimizes function
$loss = max(||a - b||_2^2 - ||b - c||_2^2, \alpha)$

Please notice the maximum within the loss function, it means that if the positive samples are sufficiently close and the negative sample is sufficiently far apart, the network will no longer use the triplet for training. This is there to improve generalization and prevent model from just increasing the distances further and further apart.

For good training of the triplet loss model it seems important to use a good sampling of the negative samples. It is better to incorporate some sort of an adversarial training, rather than just relying on random sampling. We can pick our positive samples and then look for a negative sample which should be in a different group but is too close to our positive sample.

Yet, while doing this we need to be careful because we may run into trouble with noise in the data. Therefore we should not select the single worst data-point from all the data, but rather k-th worst sample from a mini-batch sampled from the dataset.

3. The Proposed Solution

The goal of the thesis was to create a system for morphological segmentation of Czech lemmas. For this purpose we used two datasets: the DeriNet dataset (Vidra et al. [2019]) consisting of derivation trees of lemmas and a dataset of manually segmented lemmas as described in the Datasets chapter. The main question was how to combine these diverse resources in a meaningful way. The use of the manually segmented dataset was quite straightforward - we used it to train a standard classifier. To make use of the DeriNet data was a little bit more challenging, but the most difficult challenge was to actually design a complete solution making use of both of these approaches simultaneously.

In the following sections we will briefly discuss the path that led us to the development of our current solution, then we will look at the complete description of the solution, and later we will examine the details of the used parts.

3.1 Origin of the solution

Our process of development of the solution began with making use of the DeriNet derivation trees dataset. We developed a part called Segmenter (see Section 3.5), which processes a derivation tree and uses direct comparison of the neighbor words to discover new morpheme boundaries or to spread the boundaries, we already know about, from one word to another.

Later, we trained a classifier on the manually segmented dataset and used it to add the detected boundaries into a tree before processing it with the Segmenter part. We are now effectively using the DeriNet to also spread the learned boundaries trough the tree.

While looking at the output of Segmenter we discovered that it makes many mistakes which could be easy to detect, since they look wrong on the very first sight. Therefore we decided to also use the classifier in the opposite way - we also run it after the Segmenter, but this time we remove boundaries which were created by the Segmenter and are too unlikely. Therefore the classifier now has 2 thresholds, separating 3 options: ”Is not a boundary” (remove it) / ”I do not know” (preserve if present) / ”Is a boundary” (add it).

We tried to improve the classification algorithms by neighbor voting about boundaries - when we decide, whether on one place should be a morpheme boundary, we could make use of not just the word itself, but also of its parent and children in the derivation tree if they contain a sub-word containing this potential boundary. Unfortunately this approach did not lead to any noticeable improvements. It seems like the larger context inside the neighbor words does not help, and that the immediate surroundings of the place with the boundary is always the same. The classifiers likely just decide the same, because they get input, which is, in some sense, almost the same.

We also discovered that it may be useful to add a certain sort of smoothing of the outputs of Segmenter. In larger trees Segmenter easily detects boundaries which the classifier does not see. We would like to extract this knowledge and transfer it also to the smaller trees. We also know, that the Segmenter sometimes makes mistakes, but that those mistakes always look different.

Therefore our hope is that if we use the whole ensemble to generate us artificial data for training of the second classifier, then the second classifier will learn what Segmenter is doing right and use it also in the smaller trees, but that it will not learn what Segmenter does wrong.

The thinking about classifiers helped us to actually separate the ensemble into 3 parts:
1. addition phase
2. propagation phase
3. clean-up phase
later also: 4. post-processing phase, when we realized that we need to fix certain regular wrong behavior of the ensemble.

This perspective guided our future design. We looked for all the datasets which could add some information and we also created a set of rules to add some regularly formed boundaries. Each rule could also add the information that certain part surely does not contain a boundary, which was then used in the cleanup phase.

3.2 The final solution

Now we will proceed to describe the final structure of the ensemble.

The ensemble receives a DeriNet tree containing the word to be segmented as its input. Then it applies all the following methods one by one separately to each word. The only exceptions are Segmenter and propagation of roots, which are done on the whole tree at once.

1. Dataset Addition - The boundaries marked in DeriNet (Vidra et al. [2019]) and the Retrograde dictionary (Slavíčková [1975]) are added.

2. Rules - The manually created rules are used to add boundaries, and to create a blacklist of non-boundaries.

3. Dataset Cleanup - If DeriNet marks a root inside the word, then we remove all the boundaries which would split it further.

4. Neural classifier 1 - Addition of boundaries

5. Neural classifier 2 (trained on artificial data) - Addition of boundaries

6. Segmenter - Propagation of the known boundaries trough the tree and detection of the new ones

7. Rules - Removal of the boundaries from the blacklist created during 2)

8. Dataset Cleanup - If DeriNet marks a root inside the word, then remove all the boundaries which would split it

9. Classifier 1 - Prunning. If classifier considers a boundary too unlikely, we remove it

10. Classifier 2 - Prunning

11. Post-processing rules - Rules designed to fix the systematic errors done by the ensemble.

12. Dataset Addition - Addition of boundaries which are marked in DeriNet or Retrograde dictionary

13. Root Detection - We detect the root in the root word of the tree and propagate it trough the tree, while also removing boundaries which would further segment it.

Now we will proceed to describe the internal working of the main parts of the algorithm.

3.3 Classifiers

3.3.1 Architecture

We wanted to use the manually segmented data to train a neural network classifier. For this we had to decide about the general architecture of the classifier.

We decided to use the network to predict boundaries between morphemes, and not to e.g. predict whether certain group of letters forms a morpheme. We then tested both standard ways of feeding a word into a network - Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN).

After making this decision we found out that it is somewhat unclear whether the existence of morpheme boundary between letters i and i+1 should be predicted by network after e.g. after i-th letter, after i+1-th letter, or after i+2-th letter.

In convolutional networks this shift affects the scope of vision of the CNN. For example, if we want to use a convolutional network to predict morphological boundaries within a word nastoupit (to get in), then the question is, when the boundary between na-stoupit should be predicted. Should it be predicted when the convolutional window covers [ˆnas]toutpit$ or when it covers ˆ[nast]outpit$ or should it be ˆn[asto]utpit$? For a two letter suffix this question is quite clear, but bear in mind that we also want to segment roots or affixes with three or more letters.

In case of RNN's the situation is a little bit more complex, since the shift can give the network more information about the context of the boundary, but it could make the network forget the previously seen parts of word.

We also wanted to incorporate the information about the part of speech of the processed word, because it was expected to improve the performance of the network by allowing it to create specific rules for each word category. While it is straightforward to feed this information into the recurrent network, it was not very clear how to make use of this information in convolutional networks. Just adding it to the information about every letter was not considered the best approach, and due to the limited data we wanted the network to be able to use the same filters for multiple parts of speech instead of having to relearn them. For this reason we created a special variant of a CNN layer. It is a standard CNN layer, but instead of having just one bias, as it normally does, it has four biases - one for nouns, one for verbs, etc. and we select the right one depending on the processed word. This approach allows the network to separate learning of general morphemes from learning in which parts of speech they can be used.

To feed the letters into the networks we used standard trainable character embeddings trained altogether with the model, so that the model can learn similarities among the letters on its own. While feeding the letters into the network we found out that there are certain rare foreign letters in the dataset (e.g. German ö). We decided not to feed such letters into the network, since for one, the German words lie behind the boundaries of the system, and for two, we do not have sufficient data to actually learn how such letters work in words. Another option was to simply replace ö with the standard o, which still can be done by just replacing the letters before feeding them into the ensemble, if we want to.

3.3.2 Usage

In the end we trained two different classifiers - one on the manually annotated data and the other on artificial data generated by the whole ensemble, as described earlier. We then use each of them them in two ways - for adding boundaries which have sufficiently high likelihood, and for filtering out the boundaries which have too little a likelihood.

3.4 Rules

The ensemble also contains rules. We did not want to run into the common issues of the rule based systems, such as handling the conflicts of multiple rules, or designing rules to fix bugs of other rules, and therefore we decided to keep the rules simple, and only use them to pick the low-hanging fruit. The rules are directly based on the Czech morphology and they look for concrete patters.

The rules look e.g. for prefixes which rarely occur inside the stem, and also for combinations of suffixes with post-fixes specific to a given part of speech. The rules decide both where the boundaries are, and where they aren't. This should prevent the following layers of the algorithm from making certain obvious mistakes.

We later added another group of rules to the very end of the ensemble to fix the common systematic mistakes discovered during the error analysis.

3.5 Tree Propagation (Segmenter)

3.5.1 Description

The Segmenter is the central part of the algorithm. It receives the DeriNet derivation tree containing a word which we want to segment, and uses the tree to find the boundaries. The derivation trees are rooted trees of words. The edges in them represent that a given word was derived from its parent.

How could we use such a tree for prediction of the morpheme boundaries? We will look at pairs of words connected with an edge. Each such pair represents a concrete instantiation of a certain linguistic derivational phenomenon. In Czech, there are many such phenomena of very distinct semantic meaning (diminutive, adjectivisation, agent nouns, ...), yet it usually holds that the derivational changes take place along the morphematic boundaries.

Therefore if we look at a pair jít-přejít (to go - to cross), we can conclude, that it is most likely that the word "přejít" originated by addition of prefix "pře-" to the word "jít" (further we will represent this as ["+pře","jít"]). Therefore, on the basis of this pair we can conclude that there is likely a morphemic boundary: "pře/jít".

In the algorithm we do this alignment via a standard edit distance base algorithm, which tries to find how to transform one word into another while using the lowest possible number of additions and deletions of letters. The edit distance algorithms usually work on the basis of dynamic programming with a matrix from which it can be extracted not only what is the distance, but also the precise way of transformation of one word to another. It e.g. finds that the edit distance between the words "jít" and "přejít" is 3, and that the best way how to transform one word to another is to simply add the tree initial letters.

This method of using the alignment to predict the morpheme boundaries is the key principle on which the whole Segmenter relies, yet it takes further refinement. So far we have only spoken about pairs of words, yet we can extend this approach to the whole trees. If we have a simple tree:

```
učit (to teach)
- učitel (male teacher)
- - učitelka (female teacher)
```

We first use the pair učit-učitel to discover the boundary "učit/el". Then we discover, that "učitelka" was derived as ["učitel","+ka"]. From this we can conclude, that there is a boundary "učitel/ka", but we are not done yet. We also know about one boundary in the word učitel, and since we know how the word učitelka was derived from it and that the boundary remained in place during this derivation, we may also transfer this boundary: "učit/el" -¿ učitelka = ["učit/el", "+ka"]. And we so achieve the segmentation "učit/el/ka". In practice, this transfer is responsible for most of the boundaries the algorithm discovers. (See 3.1 for another demonstration.)

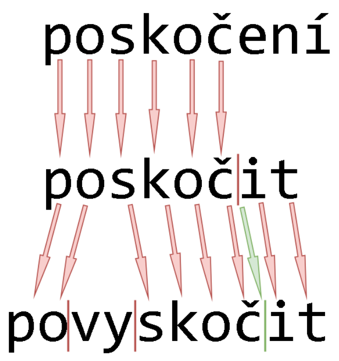

Figure 3.1: An example of segmenter's operation on words. First we compare poskočení with poskočit, and derive a boundary poskoč/it (here we also derive boundary poskoč/ení, but we do not show it for clarity). Then we compare poskočit with povyskočit, by which we derive boundaries po/vy/skočit (and po/skočit, which is again not shown), and transfer the boundary poskoč/it.

We can also use this method to transfer the boundaries received from third parties (e.g. from the classifiers run before Segmenter) and to spread them trough the tree.

Now we know how to spread boundaries from one word to any of its neighbors (parrent, children) but we still have to find an efficient way how to discover all the possible boundaries and spread them to all the possible places within the whole tree. Here we have to bear in mind that even though most of the DeriNet trees are small, a few of them contain more than a thousand of nodes.

At first we used to run this method in an iterative manner: whenever we discovered a new boundary in a word, we added it into a queue and when it was its turn we spread the boundary to all its neighbors, repeating the process.

Yet, later we discovered that it is sufficient to have two passes of word comparisons - first we go in the inverse Depth First Search order (the order in which the Depth First Search closes vertices - first children, and then the parrent), comparing the words when we step from one vertex another, and then we go in the standard DFS order. In the first pass, this algorithm discovers all the possible boundaries, and spreads them up to the lowest possible node (root=low,leaves=high), where the part of the word containing the boundary still exists: e.g. boundary in učitel/ka remains in the node učitelka, since we cannot transfer it to the word učitel. Then in the second pass, all the boundaries are spread upwards to all the possible sub-trees and leaves. This gives us a nice, linear, complexity which ensures that even the biggest trees will be processed quickly.

3.5.2 Problems

Even though this algorithm can discover many boundaries correctly it also makes many mistakes. The main problem is that the derivations in Czech are not always regular. We can frequently observe changes of the root. It may be just a change of diacritics such as in dar-dárek (gift), or omitting letters as in pes-psík (dog - doggie) -¿ compare with les-lesík (forest - little forest) - once the e disappears, once it remains, or even more complex changes, as in chůze-chodit (a walk - to walk). So far, Segmenter can only handle the changes of the first type, which was achieved by simply ignoring the diacritical signs. We expect that the remaining mistakes will be handled by the following cleanup classifiers.

For further discussion of the errors please refer to the Experiments chapter.

3.6 Root detection

We also wanted to mark the root inside each word and consequently do a simple morpheme classification to prefix/root/suffix. (We are aware that this approach to morphological segmentation is oversimplified, but it should mainly serve as the first proof of concept)

For this we came up with two algorithms: the first one is relatively simple - we morphologically segment a derivation tree, look at its root word (which likely contains the lowest amount of morphemes), detect the root morpheme of the root-word on the basis of morpheme frequencies, and then propagate the information about the root via the Segmenter algorithm. The second algorithm is described in the following section.

3.6.1 Triplet Loss

We also experimented with a little bit more advanced algorithm which unfortunately did not work well enough, but we at least used it to generate some improvements suggestions for the DeriNet dataset, by identifying pairs of derivational trees which should likely be joined.

The principle of the algorithm relies on creating a neural network $f()$, which gets a word as its input and produces an embedding vector such, that two words have a similar embedding if and only if they have a similar root. We then planed to use a trick, e.g. to run this network on all the sub-strings of a word, while comparing the embeddings with the embedding of the complete word. Then the root would be a sub-string of the word which is:

- close to the original word,
- all of its sub-strings are far from the original string.

We also thought that the distance of a sub-string from the original word might correspond with the correctness of segmentation, e.g. that sub-strings "vyskočit"* and "skočit"* would be much closer to "povyskočit" than e.g. strings "ovyskočit"** or "yskočit"**, which could be also exploited. But this property does not seem to hold. [1]

Such a network could be trained via the Triplet loss Schroff et al. [2015] (see the Theoretical Background Chapter), by sampling words a,b from a single derivation tree, and sampling word c from another tree. We would then train the network to minimize:
$loss = max(||a - b||_2^2 - ||b - c||_2^2, \alpha)$,
putting words from the same tree together, while putting the words from different trees apart. During the training we run into big trouble involving the sampling of the samples, especially of the negative ones, because further improvements required combining many sampling strategies in different parts of the training, including adversarial negative sample selection.

Unfortunately, even after we finished the training, the network still relied too much on the overall word similarity, which disallowed us to actually use it to extract the roots from the words.

Yet, comparison of the representations of root-words of various DeriNet trees at least suggested trees of words which may have been derived from the same parent word, and therefore should be joined together, such as trees beginning with brambor and biobrambora (potato, biopotato). On the other hand, its output definitely needs manual cleanup, since it produces many mistakes, such as Litva-litec (Lithuania, caster), král-Králík (king, surname meaning Rabbit), pít-pět (to drink, five/to sing).

[1] "Povyskočit" is the original word meaning something like "to jump". The strings marked with * are correctly formed words with stripped prefixes, while the strings marked with ** are non-words which originated by removal of the first letter of a prefix, instead of removing it completely.

4. Experiments

In the previous chapter we have explained the whole architecture and the general process leading to it being created the way it is. We will first discuss the process of training or developing the individual parts, then we will show how we created and debugged the whole ensemble and what are its results.

4.1 Classifiers

First we will discuss the development of the separate components.

Classifier 1

The classifier architecture was designed on the basis of grid search among many possible architectures with various hyper-parameters, namely we tested GRU + dense layer, convolution+GRU+dense layer, pure convolutional networks, and networks which combine convolution with deconvolution. We have also experimented with version of convolutional network which has filters of multiple kernel sizes in each layer. The words were fed into the network character by character, with each character being encoded with a trainable embedding with 10 digits (increase of its length did not seem to improve the results). We have also added markings of beginning and end of each word to allow the network to easier detect prefixes and suffixes at the word boundaries. Because of the need to preserve the word length we use the convolutional filters with the same size padding and stride 1. The only exception was the evaluation of the deconvolution networks where we tried to first ”compress” the words and then to ”decompress” them. We have also experimented with the mutual position of the input word and the labels (”shift”).

As an example we show results of two architectures. The method we used for the comparison of models was computing precision for several values of recall (10%,25%,50%,75%,95%). Different methods such as Area Under Curve (integral of the space bellow the PR curve) could have also been used, and would have allowed us to represent the results a single number, but we rather opted for the precision-at-N%-recall measure, since it is in closer relation with what we need and tells us more directly what shall we expect.

The architectures based on GRU (GRU×dense layer, CNN×GRU×dense layer) seemed to diverge, no matter the shifts and hyper-parameters.

	-1	0	1	2
110	88,3%	87,3%	80,2%	63,9%
220	90,8%	90,3%	83,5%	66,1%
330	91,6%	91,5%	85,9%	66,9%
440	91,9%	91,8%	85,3%	67,0%

Table 4.1: Precision for 75% recall for architecture CNN × CNN × dense which uses simultaneously multiple kernel sizes. Dependence of precision on number of filters and shift. Filters have sizes 1 to 4. 5x cross-validated.

	-1	0	1	2
ReLu	91,1%	89,8%	84,3%	66,3%
Sigmoid	69,5%	68,8%	59,3%	45,9%
Tanh	77,7%	77,6%	71,6%	59,5%

Table 4.2: Precision for 75% recall for architecture CNN × CNN × dense with 200 filters and size 4 in dependence on shift and the activation function.

While thinking about the possible ways to improve the classifier, we thought that the limit may be that the information insufficiently spreads across the word, and therefore we may be deciding about a boundary only on the basis of its most direct surroundings. Therefore we tried using the deconvolutional layers to fix this issue, but they did not seem to help.

We have also experimented with addition of part of speech tags to the network's input, but they did not lead to noticeable improvements. This may be caused by the large imbalance of our dataset, which mostly consists of nouns and adjectives.

In Figure 4.1 we also present the Precission-Recall plots for classifiers trained on various amounts of the training data. This plot inspired us to acquire more data by further annotation. The results were 10 times cross-validated and computed on the total of 2100 samples.

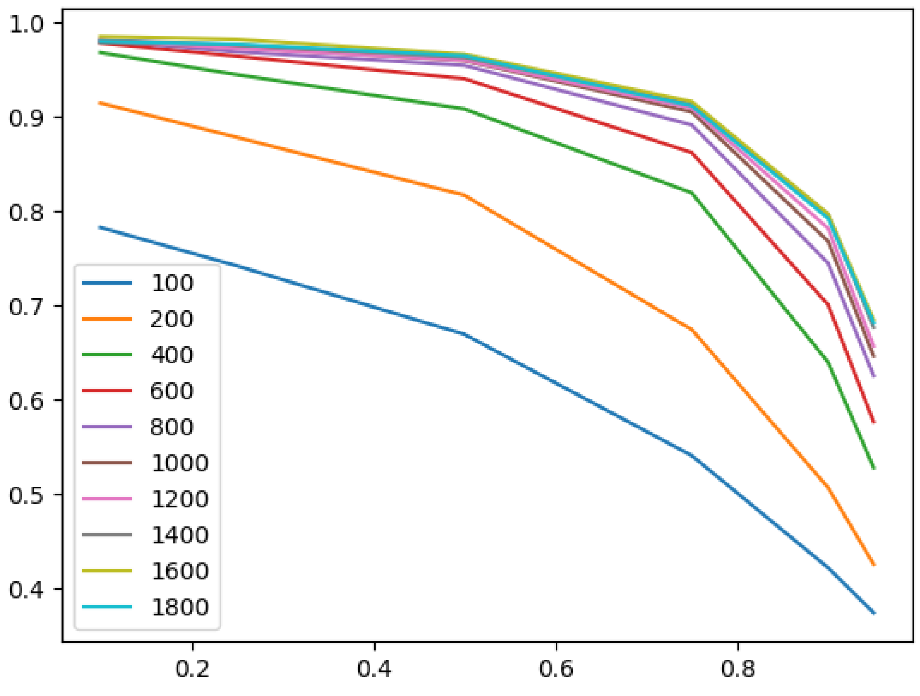

Figure 4.1: Precision-Recall plots for the Classifier 1 trained with various numbers of samples.

Classifier 2

Classifier 2 was trained on the training set with 5000 samples segmented by the final architecture (just without the classifier 2 and final post-processing rules, which were added later). The purpose of this classifier is connected with the relatively small amount of the training data we have, since there are many regularly formed morpheme boundaries which the first classifier is not able to discover, but the tree segmentation notices them. Consequently, the second classifier may learn what from the tree segmentation algorithm, and transfer the patterns it discovers also to the other trees.

The selection of architecture was partially based on the general experience from experiments with classifier 1, thus we did not further experiment with RNNs and deconvolutions, and we also had to select a model which does not tend to learn also the systematic mistakes done by the tree sementation algorithm.

The comparison of Figures 4.2, 4.3 (which show results after each layer) shows that the addition of the Classifier 2 has improved the overall results, and they also show that the Classifier 2 learned to predict part of the borders usually predicted only by the Segmenter, which was the original reason for our experiment's with it. On the other hand, we have to conclude, that the added value of the second classifier slightly decreased with the doubling of the training dataset for the classifier 1.

Thresholds

Since both classifiers are used in the ensemble and since they are used both for the addition and cleanup, the question arises regarding how did we set the thresholds - we have set the addition thresholds independently for each classifier, purely on the basis of the network's performance on the particular development set. We wanted it to add as many boundaries as it could, while preserving a fixed precision.

On the other hand we were setting both pruning coefficients at the same time and on the complete ensemble, on the basis of grid search over pairs of thresholds. Our goal was to find the best combination with regards to the precision and recall.

4.2 Triplet Loss

Initially we wanted to use the triplet loss for the root detection, but we found out that it relies too much on the overall structure of the word and therefore we are unable to use it as intended. We expect, that these issues were partially caused by the hardness of training. The main issues with training was the sampling of triplets and the distance issue.

4.2.1 Training problems

The positive sampling strategy

Each training step requires triplets consisting of two morphologically related words (positive samples), and one unrelated word (negative sample), and it is not clear how to sample them. We always want to select the positive samples from one tree, and the negative sample from another, but there are more factors to consider.

We may sample the positive pairs either as neighbors from the tree (words will be quite similar), or as words selected uniformly from the whole tree (words may be quite different, especially in large trees). We also have to decide whether to sample from all the trees with the same probability, or whether to prefer the big trees with many lemmas, or rather the small trees.

The bigger trees usually tend to contain many regularly formed words with various combinations of affixes. Their words tend to be quite distinct from one another, but most of the differences seen in big trees are regularly formed. This may help the algorithm to learn the regularities in the language, but may be as well too much for it, especially in the initial phases.

On the other hand, the smaller trees contain words which are generally closer to one another, but they may contain much higher percentage of irregularly formed or foreign words, and it is not clear whether such irregularities improve the output or worsen it. Yet, we need to sample at least part of such words, since we do not want to over-fit to just a few word roots seen in the large trees.

From the aforementioned arguments we can conclude, that sampling positive samples uniformly is not as neutral an option as it may seem in the beginning. If we uniformly sample a tree, we will almost always select a small tree, because there are many of them, and if we uniformly sample a positive pair from a big tree, we will most likely end up with very different words, within distinct branches of the tree, and almost never with quite similar words which are close apart. The consequence of this fact is, that we cannot just sample uniformly, we always have to identify the important factors, and then sample on the basis of them. In our case, such important factors are the distance between words within a tree and the tree size, and our decision is to make use of all the groups of data in a meaningful way - e.g. first training on easier samples and then on the harder ones.

Negative sampling strategy

Each triplet also needs to contain a negative sample. Here the sampling is even more important, since it is much harder to learn without good negative samples. In the beginning we use both uniform sampling from all the trees and uniform sampling from the large trees. Later we shift to adversarial sample selection - for each batch we compute the representations of its central positive samples, then we compute representations of much higher amount of negative sample candidates, and then we select k-th worst negative sample for each central positive sample. We do not select the worst sample, because we want to avoid the impact of noise, such as similar word in a different tree, words from another tree, which actually should be part of the tree with positive samples, etc.

The distance limit

The loss for Triplet loss is defined as:
loss = max(distance(a,b)-distance(b,c),alpha)
for a,b being positive and c being negative sample, and alpha being the threshold.

This design of loss is crucial. Without the alpha, the model would completely diverge, pushing the negative samples further and further apart, even if they are already are sufficiently far from one another. Yet, this design of loss causes also certain issues, namely vanishing samples. We may realize, that the percentage of samples from batch that are bellow the threshold quickly declines, even when the network is not trained yet. This means, that even if we put hundreds of samples into the network, the network may just use 20 of them for training in the given step. This creates both instability in gradient (which can be partially fixed by the momentum of the Adam optimizer we used), but also causes big issues with regards to the efficiency of the algorithm - the neural network processes many samples, just to drop most of them afterwards.

This issue is especially big in the case of the adversarially sampled negative samples, because their generation is even more demanding.

4.2.2 Our Experiments

Due to extreme computational demands (days on a gaming notebook with CUDA), we have decided to train the network in a manually supervised fashion in Pyzo (MATLAB-like IDE for Python).

We have decided to use multiple sampling strategies, and to change them as soon as the percentage of used samples in each batch drops too low. Usually we have observed that the percentage of used samples quickly dropped to relatively small values and continued to slowly decrease further. When the percentage fell below 5-10%, we continued to train the network with different sampling.

The used sampling strategies were:

1) Positive samples selected as two random samples from a randomly selected tree. Negative sample chosen at random from a different tree.
This method of sampling is expected to be the easiest for the network, because the uniformity of sampling of trees means that most of the positive samples are from the small trees and therefore there should not be a big difference between the positive samples.

2) Positive samples selected as two random samples from a randomly selected big tree. Negative sample chosen at random from a different tree.
This method of sampling is more challenging, since in the big trees the words tend to differ more.

3) Adversarial sampling: While the positive samples were selected randomly from the big trees, the selection of the negative samples was more complex. We randomly sampled N words (N = thousands, N was gradually increased, as necessary), computed their representations, measured distance from the central positive sample, and selected the k-th (initially 5-th) closest sample as the negative

This method of sampling consumed the most of the computational power. At the moment when we decided to stop training, our network was training only on 2.7% of samples from an adversarial batch consisting of 4096 samples, while ignoring the rest due to the sufficient distance, and the hard samples were selected as 10-th hardest sample from 20 000 candidates, because we had to increase the number from 5 to 10, due to observation of too many correct words on the first positions. This unfortunately led to the selection of poorer negative samples and therefore it was a factor which forced us to further increase the number of candidates to preserve the percentage of useful (sufficiently similar) negative samples.

The aforementioned numbers mean that in order to make a single training step and update the network on the basis of 110 triplets, we had to evaluate the network on tens of thousands samples. We believe that further experiments may lead to the improvements of the network, but they pose challenges regarding the computational power and regarding the manual supervision of higher number of models in the same time.

4.2.3 Outcomes

We have attempted to use the network to extract the word roots, by measuring the distance of word substrings from their parent words, but as we have mentioned earlier, these experiments were not successful, and so we at least tested whether the network could be used to create improvements suggestion for the DeriNet dataset. We measured the distances between roots of different DeriNet trees to see whether there are related words within different trees. With this we were able to create 2300 suggestions (or 3000 depending on the threshold) regarding which DeriNet trees should be merged. Sample of suggestions can be observed in appendix. After finishing the thesis we will further work to integrate the suggestions into the DeriNet.

We expect that our method can be used to generate even more improvement suggestions for DeriNet, because after a careful examination of the current suggestions we may decide to further increase the threshold. Furthermore, we were not able to use our method to compare all the possible pairs of trees.

DeriNet contains around 200 hundred thousands trees, and so checking distances between all pairs of trees becomes computationally demanding. Therefore we decided to split the DeriNet to the big trees (at least 5 nodes) and the small trees (less than 5 nodes). We only computed distances between big trees, and between big trees and small trees. Therefore we expect further suggestions regarding tree merges when we focus on the comparison of small trees with one another, and make the computation tractable. We also expect further suggestions when we integrate the already discovered tree merge suggestions, and e.g. start comparing the words which have trees of their own with all words inside the big trees (and not just with the roots of the big trees as we are doing now).

4.2.4 Error analysis

In the suggestions for the DeriNet improvements (for examples see appendix), we have observed the following systematic errors:

1) Many suggested pairs seem to show certain similarity, which resembles derivations normally seen in language:

Gent-agent, kladný-Kladno, konat-konec, brunet-Brunej, fikce-fík, kára-kárat, konstanta-Konstantin.

Such mistakes are hard to avoid, since they are not entirely wrong, they just are not right this time.

2) For certain pairs it seems as if they were caused by learning certain regular changes of letters inside words, seen in the dataset, such as: jít-jet, krk-Krč,

Kon-kůň.

3) Triplet loss often correctly discovers that certain surname or other proper noun originated from a different word and suggests merging their trees: e.g. Dlabač-dlabat, Chalupník-chalupa. Addition of such derivation edges is the matter of a linguistic decision, and so far DeriNet prefers to handle each of those names separately from the original word.

We have also examined pairs of words which are connected with derivation edges, but seem to be too far appart. Among those words there do not seem to be many useful suggestions, which is likely due to the DeriNet's preference of precision over recall. On the other hand, they are an interesting source of linguistic phenomena which the algorithms working with derivations need to handle.

It also does not seem to be the case that Triplet Loss behaves just as a pure edit distance-like metric. The following words seem to be just too far to be matched by an linguistically uninformed algorithm, unless such algorithm makes too many mistakes:

píárko-píár (public relations), duše-Duša, žďářený-žďářit, biobrambora-brambor, bipolarizovat-bipolární.

On the other hand, we have also observed a systematic error in DeriNet, because it did not recognize that word pairs such as zpikantnit-zpikantnět, zpovrchnět-zpovrchnit,.. should belong to the same tree. Although, having those words separated could as well be a linguistic decision of the authors, as it was with the surnames. If we want, in future we may look for pairs of this type automatically, since the word-alignment / word-difference method we used for tree segmenter may be also used for a convenient comparison of word pairs, and so we may e.g. attempt to look for all pairs which show the change pattern [SAME, "-i", "+ě", SAME] (replacement of "i" with "ě").

4.3 Error analysis

We knew about certain limits of our algorithm (allomorphy, word compounds, but we wanted to look also for other systematic errors. For this we used mainly manual examination of segmented morphemes, the KonText search tool, and also logs which allowed us to see which component is responsible for a boundary not being present, so that we know, that e.g. the boundary was removed by both classifiers in the cleanup phase.

Another useful tool for debugging and understanding of the complete segmentation system were the plots showing how precision and recall change as the data passes through the architecture, such as 4.2. Such plots were our main guide regarding which components should be added or modified. With them we for instance debugged the rule sets or found out, that it does not make sense to add again the boundaries formerly created by the rules, if they have been removed by the classifiers.

Before proceeding further, we will have a look at the architecture, and try to analyze it a little on the basis of 4.2.

As we look at the graph we can notice that the algorithm can be grouped into four main phases:
1] The Addition Phase (steps 1-3, 5-6): We add all the information about the

morpheme boundaries we have on the word level. In this phase recall highly increases and precision drops to the level which is limited by the performance of our classifiers and therefore can hardly be exceeded in later phases.
2] The Propagation Phase (step 7): We propagate all the information about the boundaries through the derivation trees and while doing so we also discover new boundaries stemming from the derivative relations between pairs of words. In this phase we discover boundaries which would not be discovered otherwise, which causes another huge increase in recall, but we also make a lot of mistakes, nature of which will be subject to further analysis in 3.5.
3] The Cleanup Phase (steps 8-11): In this phase we are using all the available methods to remove as many mistakes created by the segmentation phase as possible, while not removing too many correctly found boundaries. We see that during this phase the recall drops from 87.1% to 80.7% and even after the final additions it only goes to 81.6%. This shows another space for possible improvements.
4] The Final Addition Phase (steps 12-15): We again add the boundaries we are sure about, and use the fix-up rules developed while analysing the false negatives of the phase 3].
This separation helps us with the error analysis.

4.3.1 Error analysis of the Addition Phase

This error analysis overlaps with the analysis and development of the classifiers and rules. Since committing a mistake in this phase may, in the worst case, lead to its spreading around the whole tree, we simply had to add only the boundaries we were really sure about. That forced us to change the rules several times, and to set the classifier thresholds high even though it costs us a lot of recall

4.3.2 Error analysis of Tree Propagation

In this phase we propagate the boundaries received from preceding components and look for the new ones, via the tree propagation algorithm Segmenter.

This part has one big disadvantage with regards to the errors - one wrong boundary in a single word, even added by the previous modules, can be potentially spread to all the words within the whole tree. That's why we are so cautious about adding boundaries in the preceding parts.

Otherwise, mistakes done by this part directly stem from tree main causes:
1) errors in word alignment
2) changes inside morphemes
3) not discovering a boundary due to lack of information

The first type of error can be observed e.g. in pair nožířský-¿nožířství, where the words are automatically aligned as:
nožířský-¿nožířství = [”nožířs”,”-ký”, ”+tví”]
since this alignment requires less editations than the correct alignment:
nožířský-¿nožířství = [”nožíř”,”-ský”, ”+ství”]
this leads to a discovery of a wrong morpheme boundary nožířs/ký, which splits morpheme sk in half. Even more serious instance of this problem is in matching of words zajíc-zajoch (rabbit), where we get zajíc-¿zajoch=[”zaj”,”-í”, ”+o”, ”c”,”+h”], instead of [”zaj”,”-íc”, ”+och”]. This misalignment leads to segmen-

tation zaj/o/c/h [zaj(o)c(h)], instead of zaj/och [zaj(och)] and zaj/í/c [zaj(í)c instead of zaj(íc)].

The second type of error is directly connected with various linguistic phenomena. While certain phenomena, such as shortening or lengthening of vowels (e.g. dar-¿dárek = gift) are easily tackled simply via ignoring the diacritics while comparing the words, different phenomena can be much more problematic. For instance bigger changes inside the root are both quite common and hard to handle, such as in délka -¿ dlouhý (in English lenght-¿long) or nůž -¿ nožík (knife-¿little knife). Even though there are certain groups of commonly occurring root changes, handling them would require us to build large rule based system, and to later incorporate it into the tree segmentation. This may be a useful approach in future, but for now we rather decided to try a different approach. We rely on fixing these errors via the cleanup classifiers - classifier does not need to know that the root ”nůž” can transform to ”nož”. It just needs to know that morpheme boundaries ”n/o/ž” are suspicious (= unlikely).

The filtering with classifiers may not be 100% accurate, but it is still much better than the original output of the pure Tree Segmentation, as we can se below.

The third group of errors - inability to discover certain boundaries - is caused by two main factors: either the information about the presence of a boundary is not in the tree, because we simply do not have any word pair which would induce a change along the given morpheme boundary, or such a change occurs somewhere in the tree, but we are not able to propagate it further. The example of such behavior is the boundary nožov/itý. We discover it in word nožov/ý, but since there is no way how to transfer it trough the common parent of both words (nůž), we cannot propagate it to the second branch. This could be partially fixed by also allowing the transfer from brother to brother, and not just between parent and son, but it would have large negative consequences, since brother words can be quite different from one another. E.g. if we compared words přejet (to run somebody over) and předjet (to overtake), which share a common parent jet (to drive), we would discover boundaries pře/d/jet. But this is just another case of misalignment - the real structure of words is pře/jet and před/jet, with pře- and před- being completely different morphemes. Therefore, we once again decide to rely on classifiers, this time for addition.

In table 4.3 we show the comparison of outputs of the whole ensemble with output of the Segmenter on the tree belonging to the word ”nůž” (knife). We can see that the classifiers manage to fix many instances of the n/o/ž error described above, although they leave a few of them. The classifiers also make several false negative errors when removing true boundaries detected by the Segmenter.

Removing of detected true boundaries is actually a common occurrence, and therefore a space for potential future improvements. If we compare the statistics of Tree Propagation and the Final Cleanup in figure 4.2 we can notice that after passing trough the tree propagation, we discover 87.1% of the true boundaries, which is much more than the final output of the algorithm - 81.6%.

4.3.3 Error analysis of Cleanup Phase

In analysis of Phase 3 we try to spot systematic False Negative errors, which we can fix by addition of special fix-up rules to the Phase 4. Part of the false

Segmenter	Ensemble	Correct
n/ů/ž	nů/ž	nůž
n/o/ž/ovit/ý	nožovit/ý	nož/ov/it/ý
n/o/ž/ovit/ě	nožovit/ě	nož/ov/it/ě
n/o/ž/ovit/ost	nožovit/ost	nož/ov/it/ost
n/o/ž/ov/ý	nož/ov/ý	nož/ov/ý
n/o/ž/ov/ě	nož/ov/ě	nož/ov/ě
n/o/ž/ov/ost	nož/ov/ost	nož/ov/ost
n/o/ž/í/k	nož/í/k	nož/ík
n/o/ž/í/če/k	nož/íč/e/k	nož/í/če/k
n/o/ž/íř	nož/íř	nož/íř
n/o/ž/íř/ka	no/žíř/k/a	nož/íř/ka
n/o/ž/íř/čin	no/žíř/č/in	nož/íř/čin
n/o/ž/íř/s/k/ý	no/žíř/sk/ý	nož/íř/sk/ý
n/o/ž/íř/s/k/ost	no/žíř/sk/ost	nož/íř/sk/ost
n/o/ž/íř/s/k/y	no/žíř/sk/y	no/žíř/sk/y
n/o/ž/íř/s/t/ví	no/žíř/stv/í	no/žíř/stv/í
n/o/ž/íř/ův	nožíř/ův	nož/íř/ův
n/ů/ž/k/y	nůž/k/y	nůž/k/y
n/ů/ž/k/ov/ý	nůž/k/ov/ý	nůž/k/ov/ý
n/ů/ž/k/ov/ě	nůž/k/ov/ě	nůž/k/ov/ě
n/ů/ž/k/ov/ost	nůž/k/ov/ost	nůž/k/ov/ost
n/ů/ž/tič/k/y	nůžtič/k/y	nůž/tič/k/y

Table 4.3: The comparison of segmentation done by the tree segmentation with the segmentation produced by the whole ensemble and with the correct answers.

negative errors present in this phase was caused by the classifier 2 being too sure that certain boundary should not be present. It stems from it being trained on the synthetic data, and thus accidentally overfitting to certain mistakes of the whole ensemble. An example of such behavior its systematic omitting of border between suffixes in tel/k, which was fixed by addition of a specialized rule to the set of final rules.

As a test, we have processed the development dataset[1] with the Classifier 1 in the cleanup mode, and observed, which boundaries will be removed even if some part discovers them. This approach may be in some sense redundant, since it is duplication of analysis we can do on the final output, but on the other hand it helps us to see, that if we improve one part, it may still not be enough.

This is the case for instance with word compounds, which often tend to be in different trees than the original words in DeriNet, and therefore the Segmenter cannot detect the boundaries, and even if it could detect them, the classifier would remove them anyway. In general, most of the errors seem to be in long words and on the root boundaries, both in prefix and suffix. Analysis also showed problems with the detection of single letter morphemes - s,z,v,... - the classifier suggests removal of 18 out of 58 single letter prefixes in the dataset.

4.3.4 Systematic errors of the whole architecture

During the analysis we discovered several systematic errors done by the whole architecture. We decided to integrate fixes for all of them into the very last module - FinalRulesPostprocessor, which just looks for them and fixes them. Such an approach may be considered dirty, yet there are no better approaches available. We are also aware of the fact that such rules also make mistakes, but we use them because they fix more things than they break.

The systematic mistakes we discovered:

1) We found out that the ensemble systematically does not split pair of morphemes tel/k, and leaves them together. Such a morpheme combination appears e.g. in word učitelka (female teacher). This lead to addition of a rule t-e-l/k, meaning ”remove boundaries between t,e,l if there are any, and add a boundary in front of k. This mistake originates from both classifiers which remove this boundary when they see it because they consider it suspicious. This was likely caused by the fact that the training dataset did not contain any word with this morpheme, while containing many morphemes ”teln”, where we do not split after ”tel”. This mistake was learned by classifier 1, which later transferred it to the artificial dataset, and thus to the classifier 2.

2) We also discovered that the algorithm has trouble handling doubled prefixes. Therefore we added the list of 12 common combinations, and later inspired by this we also added 4 common pairs of suffixes. Yet here we had to be careful because unlike with prefixes we cannot tell our rule to only match on the beginning of the word, and therefore we had to pick only suffixes letters of which are unlikely to appear anywhere else in the words (such as suffix pair tel/čin).

3) We also see that because of misalignment algorithm sometimes splits letter combination ”ch”, while ”ch” always acts as a single letter, and therefore should

[1]We want to use the test dataset also in future, therefore we do not want to just go through it word by word.

never separated. We consequently added a rule to fix this behavior. In future it may be worth considering, whether we should not consider ”ch” a single letter since the very beginning, and have it as one letter even while processing it with neural networks.

4) There are also some broader groups of words with which this architecture has systematic problems, namely:

- Word compounds (e.g. čtyřstěn - tetrahedron): Their handling will be examined in future since it requires a completely separate approach. An example of such approach could be e.g. looking for words which seem to have long roots after the segmentation, and trying to decompose these ”roots” into two or more real roots possibly connected via interfixes.

- Foreign words and names: They are currently considered out of scope, although in future we may try to detect foreign words and either avoid their segmentation completely, or just detect the foreign part within them, and only segment the rest, as if it was a Czech word. The second approach may become handy on words such as ”vygooglit” (to Google), since the foreign root ”googl” is surrounded by the Czech affixes ”vy-” and ”-it”, which have their own fixed meanings.

- Segmentation of word forms: in this thesis we focused only on segmentation of word lemmas, but since the Czech is a very flective language and each word can have many different endings, it is very useful to be able to also segment the forms. Therefore We carried out an experiment to explore the behavior of the algorithm on word forms. We have run the segmentation algorithm on part of the MorfFlex datasetStraka and Straková [2016] and than manually examined the results.

Unfortunately we observed that the cleanup classifiers are systematically removing boundaries in word endings because they do not know them lemmas. Therefore the ensemble does not output the boundaries even if the Tree Segmentation detects them while processing the MorfFlex trees. To fix this behavior we would need a special annotated dataset of segmented word forms to retrain the Classifier 1, and afterwards generate another artificial dataset to retrain also the Classifier 2.

Therefore the segmentation of word forms also remains future work, although we believe that it will not require too many modifications for our algorithm to work on them.

4.4 The final results

In this section, we evaluate the algorithm on two datasets, namely verbs from the retrograde dictionary Slavíčková [1975], in the form discussed in the Related Work Chapter, and our segmented dataset, and we show that our algorithm achieves the state of the art performance on the problem of morphological segmentation of Czech words.

In 4.4 we can notice that the results in prediction of morphemes are much worse than the results in prediction of the boundaries. On the first sight it may look confusing, but it is a direct mathematical consequence of the fact that a single mistake in prediction of a boundary often causes two morphemes to be wrong.

Method	Morphemes			Boundaries			Words
	Precision	Recall	F1	Precision	Recall	F1	Accuracy
Our method	64.50%	58.60%	61.41%	93.30%	81.50%	87.00%	8.90%
Vidra [2018]	34.03%	24.08%	28.20%	70.33%	42.12%	52.69%	19.46%
Flatcat sup.	36.40%	21.20%	26.79%	83.70%	35.70%	50.05%	26.70%
Everywhere	10.17%	24.39%	14.35%	34.20%	100.00%	50.97%	0.00%
Nowhere	21.79%	5.90%	9.29%	X	0.00%	X	10.89%

Table 4.4: The results of evaluation of our algorithm, and two others on our manually segmented dataset. Everywhere and Nowhere are simple baselines predicting boundaries everywhere or nowhere, Flatcat is abreviation of Morfessor Flatcat Grönroos et al. [2014].

Method	Morphemes			Boundaries			Words
	Precision	Recall	F1	Precision	Recall	F1	Accuracy
Our method	58.94%	62.62%	63.74%	96.11%	91.63%	93.81%	72.15%
Vidra [2018]	64.82%	66.54%	65.67%	91.38%	87.42%	89.36%	35.45%
FlatCat unsup.	37.19%	21.57%	27.31%	97.98%	64.92%	78.10%	1.17%
FlatCat sup.	66.20%	57.72%	61.67%	92.59%	81.15%	86.49%	31.10%

Table 4.5: For fairness we also evaluate our method on the same dataset as was used in Vidra [2018] (namely Slavíčková [1975]) and compare our results with results presented there. Therefore the whole table except for our results is taken from Vidra [2018].

We have observed that our algorithm performs better on words uniformly randomly sampled from the list of all words (95% Precision, 81% Recall) than on the words sampled with regards to the number of occurrences in the corpus (90% P, 82% R). It is likely caused by a higher average number of morphemes in the words from the first dataset. It means that the first dataset contains longer, and thus likely more regularly formed words. That means that it contains many boundaries which are easy to guess, which improves the algorithm's score.

This result is not too surprising. Most of the word lemmas n Czech are the long ones, and therefore our method of sampling tended to prefer the long words. Also the long words should, more often than not, be formed more regularly because of the principle of language economy. This principle tells us that effectivity is one of the principles guiding the development of language. Therefore we should rather expect that long words are formed regularly, in some language specific way, since it would be too complicated for speakers to remember meaning of each long word independently or to remember too many specific irregularities.

We have also observed that the behavior of the algorithm differs on the retrograde dictionary verb dataset, and on the standard data. The segmentation of the verbs seems to be much more centered around the rules - after the rule processing phase algorithm achieves 77.2% recall and 84.6% precision (compare with 43.5% recall and 96.1% precision on the standard dataset). It seems like the verbs were much more regularly formed, but also have much higher likelihood of the false positive matches, which were then fortunately removed by the classifiers in the cleanup phase.

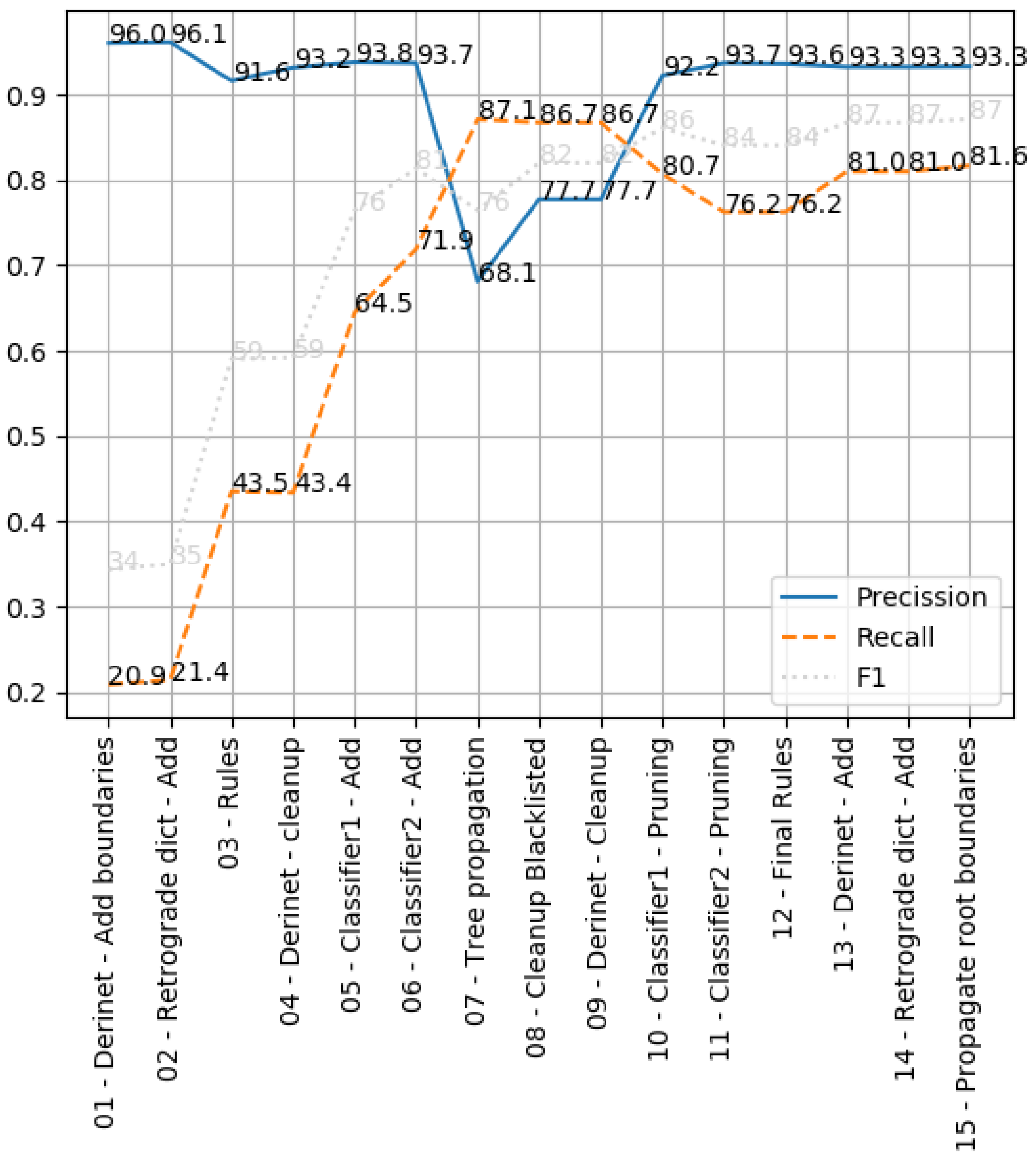

Figure 4.2: Plot of Precision and Recall of boundaries after each layer of the ensemble.

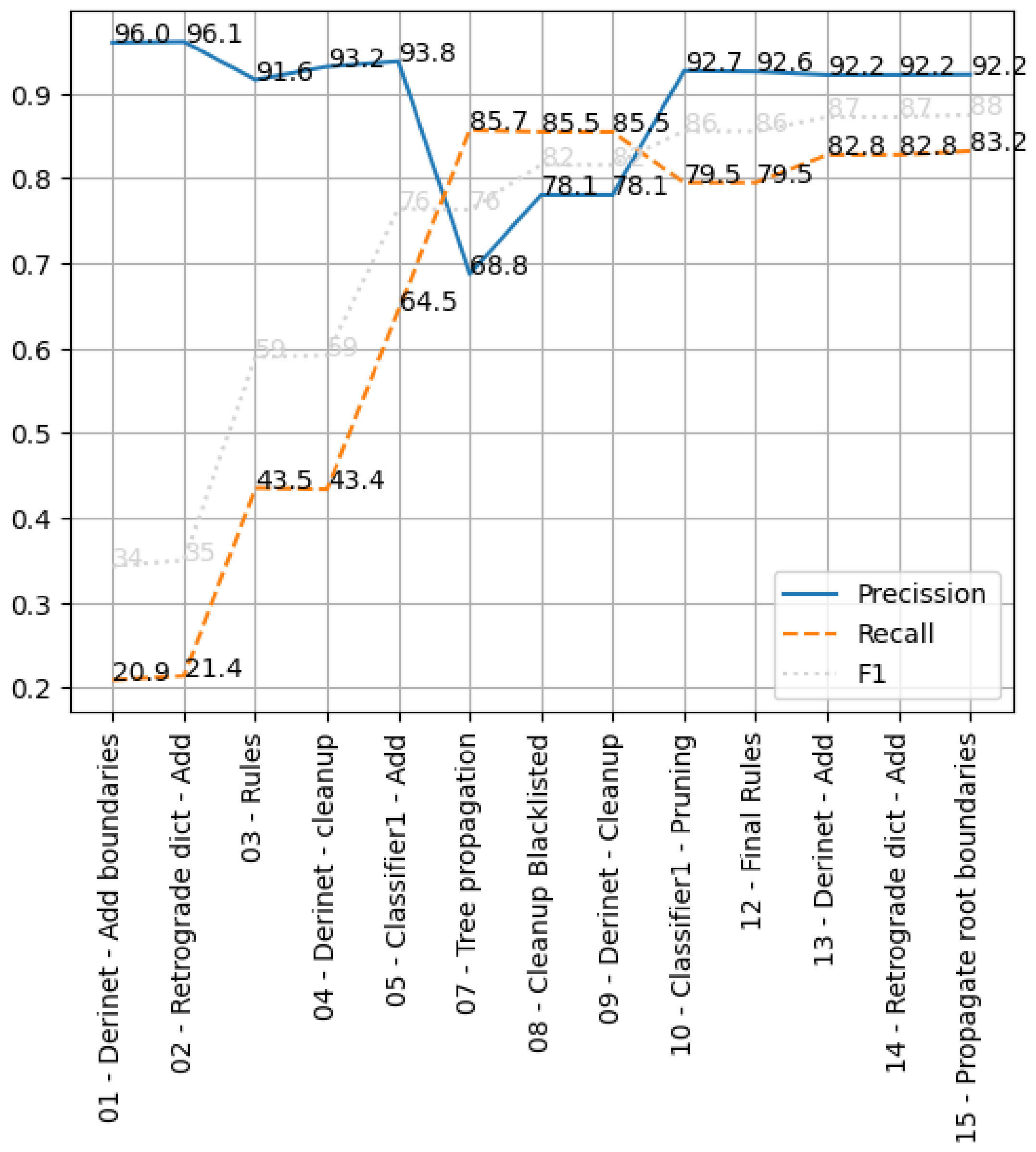

Figure 4.3: Plot of Precision and Recall of boundaries after each layer of the ensemble. This time without Classifier 2.

Conclusion

In this thesis we have presented a novel algorithm for morphological segmentation of Czech lemmas, which surpasses the previous state of the algorithm (Vidra [2018]) and which was accepted for publication in Bodnár et al. [2020]. The developed segmentation will also be added to the next release of the DeriNet dataset. We have also created many suggestions for improvements of the DeriNet dataset, namely suggestions of trees which should be joined because they are derived from the same word.

From the global perspective we may conclude that the achieved results (Precission 93.30%, Recall 81.50% on boundaries) are promising, which shows that the combination of a manually segmented dataset with the derivational tree dataset DeriNet (Vidra et al. [2019]) was very fruitful, yet there is still space for future improvements.

In future, we would like to achieve further quantitative improvements but most importantly we would like to (and we plan to) focus on the segmentation of word forms instead of segmenting just lemmas. We believe that this could be achieved relatively easily by retraining the classifiers, but it will require a development of a novel dataset, this time also containing the word forms.

There is also space for improvements in the core of the algorithm, namely we may focus on the false removals of the discovered boundaries between the tree segmentation part and the clean-up classifiers, as well as on the problematics of word compounds and foreign words.

Bibliography

Christopher M. Bishop. *Pattern recognition and machine learning.* Springer, 2013.

Jan Bodnár, Zdeněk Žabokrtský, and Magda Ševčíková. Semi-supervised induction of morpheme boundaries in Czech using a word-formation network. In *Proceedings of the 23st International Conference on Text, Speech and Dialogue - TSD 2020.*, Brno, Czech Republic, sep 2020. Springer [to be published].

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016. URL `http://arxiv.org/abs/1607.04606`.

F. Čermák. *Jazyk a jazykověda.* Charles University, 2011. ISBN 9788024619460.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL `http://arxiv.org/abs/1409.1259`.

Mathias Creutz and Krista Lagus. *Unsupervised Discovery of Morphemes*, volume cs.CL/0205057. 2002. URL `https://arxiv.org/abs/cs/0205057`.

Mathias Creutz and Krista Lagus. Induction of a simple morphology for highly-inflecting languages. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, pages 43–51, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W04-0106`.

M. Dokulil. *Tvoření slov v češtině: Teorie odvozování slov.* Nakl. Československé akademie věd, 1962.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1207/s15516709cog1402_1.

John Goldsmith. An algorithm for the unsupervised learning of morphology. *Nat. Lang. Eng.*, 12(4):353–371, December 2006. ISSN 1351-3249. doi: 10.1017/S1351324905004055. URL `https://doi.org/10.1017/S1351324905004055`.

Sharon Goldwater, Thomas Griffiths, and Mark Johnson. A bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112:21–54, 04 2009. doi: 10.1016/j.cognition.2009.03.008.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

Stig-Arne Grönroos, Sami Virpioja, Peter Smit, and Mikko Kurimo. Morfessor FlatCat: An HMM-based method for unsupervised and semi-supervised learning of morphology. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1177–1185, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics. URL https://www.aclweb.org/anthology/C14-1111.

Stig-Arne Grönroos, Sami Virpioja, and Mikko Kurimo. Morfessor em+prune: Improved subword segmentation with expectation maximization and pruning, 2020.

Zellig S. Harris. From phoneme to morpheme. *Language*, 31(2):190–222, 1955. ISSN 00978507, 15350665.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL http://arxiv.org/abs/1412.6980.

Kevin Knight. Bayesian inference with tears. 2009.

Michal Křen, Václav Cvrček, Tomáš Čapka, Anna Čermáková, Milena Hnátková, Lucie Chlumská, Dominika Kováříková, Tomáš Jelínek, Vladimír Petkevič, Pavel Procházka, Hana Skoumalová, Michal Škrabal, Petr Truneček, Pavel Vondřička, and Adrian Zasina. SYN2015: representative corpus of written czech, 2015. URL http://hdl.handle.net/11234/1-1593. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Radford M. Neal and Geoffrey E. Hinton. *A View of the Em Algorithm that Justifies Incremental, Sparse, and other Variants*, pages 355–368. Springer Netherlands, Dordrecht, 1998. ISBN 978-94-011-5014-9. doi: 10.1007/978-94-011-5014-9_12. URL https://doi.org/10.1007/978-94-011-5014-9_12.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

Frank F. Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. *American Journal of Psychology*, 76:705, 1963.

Teemu Ruokolainen, Oskar Kohonen, Kairit Sirts, Stig-Arne Grönroos, Mikko Kurimo, and Sami Virpioja. A comparative study of minimally supervised morphological segmentation. *Computational Linguistics*, 42(1):91–120, 2016. doi: 10.1162/COLI_a_00243.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015. URL http://arxiv.org/abs/1503.03832.

Magda Ševčíková, Anja Nedoluzhko, and Šárka Zikánová. Lecture notes - variability of languages in time and space (npfl100), charles university, 2019.

C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1948.tb01338.x.

Eleonora Slavíčková. *Retrogárdní morfematický slovník češtiny.* Academia, 1975.

Benjamin Snyder and Regina Barzilay. Unsupervised multilingual learning for morphological segmentation. In *Proceedings of ACL-08: HLT*, pages 737–745, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/P08-1084`.

Richard W. Sproat, Chilin Shih, William Gale, and Nancy Chang. A stochastic finite-state word-segmentation algorithm for Chinese. *Computational Linguistics*, 22(3):377–404, 1996. URL `https://www.aclweb.org/anthology/J96-3004`.

P. Štekauer, S. Valera, and L. Kőrtvélyessy. *Word-Formation in the World's Languages: A Typological Survey.* Word-formation in the World's Languages: A Typological Survey. Cambridge University Press, 2012. ISBN 9780521765343.

Milan Straka and Jana Straková. Czech models (MorfFlex CZ 161115 + PDT 3.0) for MorphoDiTa 161115, 2016. URL `http://hdl.handle.net/11234/1-1836`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Jonáš Vidra, Zdeněk Žabokrtský, Lukáš Kyjánek, Magda Ševčíková, and Šárka Dohnalová. DeriNet 2.0, 2019. URL `http://hdl.handle.net/11234/1-2995`. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Jonáš Vidra. *Morphological segmentation of Czech Words, Diploma thesis, Institute of Formal and Applied Linguistics, Charles University, Prague. Supervisor: Žabokrtský, Zdeněk.* Charles University, 2018.

Linlin Wang, Zhu Cao, Yu Xia, and Gerard de Melo. Morphological segmentation with window lstm neural networks. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2842–2848. AAAI Press, 2016.

František Čermák. Discrete language units: The case of circumfixes. *Jazykovedny Casopis*, 69:78–98, 01 2008. doi: 10.1515/jazcas-2017-0012.

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 Output Samples

Here we show samples of output on our dataset.

Alarik/ův	Rón	frajkumšt
Alžbět/in/č/in	Salvet	hlad/í/v/a/n/ý
Andrl/e	Skořep/ův	impaktit
Barret/o	Somorovsk/ý	indikátorčin
Blaufeld	Třečk/ův	intrakraniál/n/ě
Bo/ogerd	Uerdingen	iris/ová/v/a/teln/ý
Buyss/e	Uusikaupunk/i	jednoduš/e
Carreras/ův	Voběr/e/k	jednofotonov/ost
Chondroste/i	Vrecion/ov/á	jehlař/ův
Cipras/ův	Vričan	judaiz/ujíc/í
Dehk/án/í	Waters/ov/á	jár/ek
Dobříň	Weill/ův	klikv/ov/ý
Durrán/í	bezfaset/ov/ost	krep/ovat/ě/n/í
Duš/ík/ův	bezkalibrač/n/ě	krup/ař/stv/í
Fenn/ov/á	bezo/hled/n/ost	kufr/ová/v/a/n/ost
Filin/ov/á	bíd/nic/k/y	mantinel
Gautsch/ův	bílich/ovsk/ý	megawatt/ov/ost
Gavin	de/kriminal/iz/ová/v/aj/íc/í	minilož/nice
Harciník/ův	debet/ova/teln/ě	motoball
Jablunk/ovsk/o	div/ost	na/elox/ova/n/ost
Kadrák	do/dých/a/t	na/šplh/ac/í
Kernen/ův	do/parazit/ová/v/a/c/í	na/ťuk/á/v/aj/íc/í
Litět/in/y	drezír/ová/v/a/n/ý	nach/ovočerv/en/ost
Mladc/ov/á	dvojnehod/a	ne/ofenomenologick/ost
Nazdice	dvou/vaječn/ě	ne/plát/c/ův
Petrus/ov/á	dvouletoun	ny/ktalop/i/e
Pošíval	dějepis/ář/k/a	o/kol/k/ová/v/a/c/í
Raisin	fanariot/sk/ost	o/kou/t
Ramund	fetišiz/ová/v/a/n/ý	o/rádl/ová/v/a/teln/ost
Reneš	flaš/k/ov/ý	o/slav/ová/v/a/teln/ě

Table A.1: Sample of output of our algorithm 1/2

o/žlut/i/vš/í	roz/kon/stru/ova/n/ě	z/melodičt/ěn/ý
obe/zd/ěn/ý	roz/masakr/ová/v/a/c/í	z/mrhank/a
od/flusn/ou/t	rural/izmu/s	z/pod/pír/a/n/ost
od/hrk/á/v/a/n/ě	s/mrsk/nut/ý	z/ruš/ova/n/ost
od/sklíp/k/ová/v/á/n/í	sebedrsněji	z/teolog/izova/teln/ý
pasáž/ov/ý	sed/ě/t	z/važ/ova/c/í
po/ná/lep/k/ová/v/a/n/ě	souk/en/í/k	z/vlášt/n/ůstk/ář/č/in
po/smrk/á/v/a/t	spoluz/a/váz/á/v/a/t	z/výz/namň/ova/n/ost
pod/kluz/ova/t	starokošíř/sk/ý	za/hán/í/v/a/teln/ost
poet/iz/ová/v/a/n/ý	starousedlic/e	za/isol/ova/c/í
polohlasit/ě	sub/trop/y	za/plomb/ova/teln/ě
prask	subjekt/iv/ova/vš/í	za/syp/ac/í
pro/faktur/ová/v/a/n/ý	super/zem/ě	za/řad/ěn/ě
pro/pis/ová/v/a/teln/ě	tematiz/ová/v/a/teln/ý	za/šklíb/nu/teln/ý
pro/šlechť/ova/c/í	tygrobijčin	zcukern/ě/vš/í
prorůst/ov/ý	tříhřbet/ost	ze/stručn/i/teln/ě
proven/cal/sk/ý	tříkrál/ovsk/y	ze/síť/ova/c/í
psaníčk/ovit/ě	u/hrab/á/v/aj/íc/í	zebr/ov/ě
pře/dob/chod/n/ě	u/vřísk/á/v/a/c/í	zelenohnědočern/ost
pře/klimatis/ova/teln/ě	u/za/vír/á/v/á/n/í	zgramatis/ová/v/a/n/ý
pře/komun/ik/ova/teln/ý	večeř/í/v/a/teln/ost	zješit/n/ěn/í
pře/kouzl/i/vš/í	vojvod/íc/í	zponenáhl/i/vš/í
pře/tíž/en/ost	vy/bruš/ová/v/a/teln/ě	záplat/ová/v/a/teln/ě
pře/škrt/á/v/á/n/í	vy/bíd/nu/teln/ě	Čubrič/ův
před/registr/ová/v/a/teln/ý	vy/kyd/nu/vš/í	Čumpelík/ův
při/clán/í/v/a/teln/ý	vy/specifik/ová/v/a/n/ě	ďob/nou/t
při/cvak/á/v/a/t	vy/vyš/ova/t	Šebelk/ův
při/kvap/en/í	vy/štěk/a/teln/ě	šašlik/ov/ý
pří/je/zd	vy/žen/i/teln/ý	šroub/ová/v/a/n/ě
roz/brnk/á/v/a/teln/ý	z/důraz/ň/ová/v/a/c/í	ženští/v/a/n/ě

Table A.2: Sample of output of our algorithm 2/2

A.2 Triplet Loss suggestions

As a side product, the triplet loss offered us some suggestions regarding possible improvements of the DeriNet dataset (Vidra et al. [2019]).

We did two comparisons, namely detecting, which words connected with a derivation edge are too far apart, and comparing which roots of distinct trees are too close to one another.

Due to the computational complexity, we did not compare all the 200 thousand trees to one another in the second comparison. Instead we separated the trees into two groups, namely big trees with 5 or more nodes, and small trees, and we compared the distances between the big trees and between the big trees and the small trees.

We show 60 randomly selected samples of output for each experiment ant various thresholds. Due to vulgarisms and similar cases we had to decide to censor a few words.

We have also examined edges in trees and looked for possible wrong connections by detecting that two connected words are too different. With lower threshold the algorithm discovered a few mistakes, but in general this approach does not seem to be very successful. But this could be partially caused by DeriNet's preference of precision over recall. At least these samples tell us something about the behavior of our triplet loss model and point out problematic linguistic phenomena.

Alexandr	Saša	plán	raketoplán
Pankrác	Bonifác	pojít	pošedší
Václava	Vendula	povyjít	povyšedší
biograf	biják	raketa	rachomejtle
chtivý	burčákochtivý	rex	methylaminorex
dojít	došedší	ský	házenářskost
dárce	službodárce	ský	házenářsky
gen	fibrinogen	stanice	trafostanice
gen	kalyptrogen	tombak	tumpachový
lyzin	fibrinolyzin	téka	filmotéka
metr	bilirubinometr	vyjít	vyšedší

Table A.3: Triplet Loss Suggestions - Edges to remove - Samples which are connected with an edge but are too far apart, and therefore may belong to a different tree. Threshold 150. 22 samples out of 23 suggestions. (1 sample were censored, 1 of which was a correct suggestion.)

Alexandr	Alex	přijít	přicházet
Kyrgyz	Kirgizův	relykrosový	rallyecrossovost
Mikuláš	Mikeš	rozepsat	rozpis
Mikuláš	Miky	sejmout	sňatek
Shakespeare	šejkspírovský	sejmout	sňatý
absolvovat	absolutorium	sejít	scházet
analýza	kryptoanalýza	sejít	sešedší
business	byznys	sežnout	sžatý
být	budoucí	ský	házenářsky
cement	azbestocement	sníh	sněhulák
dvěstětunový	dvousettunovost	typ	genotyp
gram	daktylogram	tónický	sylabotónický
hibernace	hibernakulum	těsný	palivotěsný
kazit	kažený	třít	trdlice
klít	klatba	uhnout	uhýbat
kruh	okrouhlý	ventilace	filtroventilace
kvazimonopolní	quasimonopolně	vepsat	vpisek
ležet	ložnice	vyjmout	vynětí
metr	bilirubinometr	vyjmout	vyňavší
metrie	granulometrie	vysoký	navýšit
mánie	agentománie	vysoký	povýšit
mést	zmítat	vzejít	vzcházet
nitril	naftonitril	vzít	vezmoucí
obejít	obešedší	vápenný	struskovápenný
obežnout	obžavší	vát	věječka
odvézt	odvážet	žnout	žatva
přemoci	přemohší	žravý	semenožravý

Table A.4: Triplet Loss Suggestions - Edges to remove - Samples which are connected with an edge but are too far apart, and therefore may belong to a different tree. Threshold 100. 54 samples out of 358 suggestions. (6 samples were censored, none of them was a correct suggestion.) Here we see mosly false positives, which shows us which things are unintuitive for the network.

Andrejevský	Andrejev	Skalický	Skalice
Ban	bán,báň	Skalička	Skalice
Banovec	Bánovec	Valente	Valentina
Blanke	Blanka	Virginia	Virginie
Bořice	bořit	Wait	Waits
Bořita	bořit	batman	Batmanov
Budínka	Budín	fantaz	fantazie
Bulgakovová	Bulgakov	galeje	galejní
Chalupný	chalupa	generálka	generál
Chan	chán	granátit	granát
Chvíla	chvíle	kompenzovna	kompenzovat
Chánov	chán	manažér	manažer
Dolinek	doliňák	minižáček	minižák
Dominick	Dominic	nirvana	nirvána
Drozda	drozd	plantace	plantat
Horne	horna	přeštíhlý	štíhlý
Ignace	Ignác	remonstrace	remonstrant
Kaván	Kavan	trávař	tráva
Kikujo	Kikuj	trávníkářství	trávníkář
Kmenský	kmen	verbovní	verbovat
Konstantinky	Konstantin	vincentka	Vincent
Kožušnik	Kožušník	vroucno	vroucný
Lednice	lednit	záhvozdí	záhvozd
Liana	liána	Červenkov	Červenka
Lusitánie	Lusitán	Štětice	štětina
Montána	Montana	Švihovsko	Švihov
Platini	platina	Šíša	Šiša
Ramus	rámus	žemla	žemle

Table A.5: Triplet Loss Suggestions - Roots to merge (small trees with big trees) - suggestions regarding which trees should be merged because their roots contain very similar words. Threshold 10. 56 samples out of 2113 suggestions (4 censored, 3 of them being correct suggestions

Davidov	David	týl	Tyl
Haná	Hana	vraštit	vráštit
Koperník	Kopernik	vzdušnit	vzdušnice
Koč	kočí	zakatalogisovat	katalogisovat
Lombardie	lombard	zdrobnět	zdrobnit
Martinov	Martin	zesličnit	zesličnět
Soliman	Solimán	zhorečnit	zhorečnět
Valentina	Valentin	zjednostranět	zjednostranit
Vaník	Vaněk	zjednotvárnit	zjednotvárnět
Vaněk	Vaník	zklasičtět	zklasičtit
doinicialisovat	inicialisovat	zkonvenčnit	zkonvenčnět
dolinář	Dolina	zkonvenčnět	zkonvenčnit
dosystemisovat	systemisovat	zlacinět	zlacinit
flitrovat	oflitrovat	zlogičtit	zlogičtět
hendikepový	hendikepovat	zmalichernit	zmalichernět
horolezec	horolezení	zneklidnět	zneklidnit
kosmopolita	kosmopolitický	zněžnit	zněžnět
líto	lítý	zoptimalisovat	optimalisovat
mandl	mandle	zpikantnit	zpikantnět
odpínat	podpínat	zpovrchnět	zpovrchnit
oflitrovat	flitrovat	zpovšechnět	zpovšechnit
plakat	plakát	zprůchodnit	zprůchodnět
poleno	Polena	zrabiátnit	zrabiátnět
produchovnit	produchovnět	zteatrálnit	zteatrálnět
prýmkář	prýmka	ztechničtit	ztechničtět
punk	punč	zvýznamnět	zvýznamnit
román	Roman	zásvětí	zásvětný
rozmobilisovat	zmobilisovat	Řihák	Říhák
stonek	Stone	Štětkař	štětkář
stříhnout	ustříhnout,nastříhnout		

Table A.6: Triplet Loss Suggestions - Roots to merge (big trees with big trees) - Pairs of roots of big trees, which are too close to one another, and therefore likely contain related words and should be merged into one tree. Threshold 10, 59 samples out of 277 (1 censored, out of which 1 was correct)

Bíl	bílý,běl	ošklíbat	pošklíbat
Gal	Gál	pasivní	pasivovat
Jan	Ján	patron	patrona
Jánoš	Janoš	pozápadnit	pozápadnět
Král	Králík,Královák	prýmka	prýmkář
Loučka	loučit	přespecialisovat	zespecialisovat
Marija	Maria	rolničit	rolnička
Mikulič	Mikulík	rozcensurovat	zcensurovat
Poledník	poledne	rozgeneralisovat	zgeneralisovat
Vinklár	vinklář	sardinkář	sardinka
amalgam	amalgám	stážovat	stáž
amortisovat	zamortisovat	ustabilisovat	stabilisovat,dostabilisovat
assimilovat	asimilovat	ustabilisovat	nastabilisovat
chroptit	chroptět	zabstraktnět	zabstraktnit
cvočkář	cvočkař	zavulkanisovat	vulkanisovat
doinicialisovat	inicialisovat	zdredovatět	dredovatět
doinicialisovat	zinicialisovat	zklasičtět	zklasičtit
doinicialisovat	rozinicialisovat	zmnohonásobit	zmnohonásobnit
horký	Horka	změlčet	změlčit
jelenář	Jelena	zobyčejnit	zobyčejnět
kampan	kampaň	zobčanštit	zobčanštět
kat	kát	zpestřet	zpestřit
krejčí	Krejčík	ztisíceronásobnit	ztisíceronásobit
krčit	Krč	čmuchat	dočmuchat
kát	kat	Špetík	špetka
nastehnout	zastehnout	štětkář	Štětkař,štětina
neutralisovat	zneutralisovat	žaludek	žalud
nový	Novák,nit	žvástat	žvastat
oploštět	zploštět		

Table A.7: Triplet Loss Suggestions - Roots to merge (big trees with big trees) - Pairs of roots of big trees, which are too close to one another, and therefore likely contain related words and should be merged into one tree. Threshold 10, 59 samples out of 839 (5 censored, out of which 4 were correct)

Aachenová	Aachen	Verlaine	Verlain
Absolut	absolutní	Vostrov	ostrov
Alan	Alán	argentinsko	Argentina
Ban	bán,báň,banka	balonkový	balon
Berl	berle	bělka	běl,Běla
Bimbác	bimbat	chatt	chata
Boba	bob,Bob	ergometrin	ergometrie
Brdec	Brdečka	fašančář	fašankář
Buchář	Buchár	fenomen	fenomén
Chud	chudý,chudý	glazé	glazovat
Dačický	Dačice	guvernérování	guvernér
Dráža	dráha	jednotvárný	zjednotvárnit,zjednotvárnět
Gene	gen	koučka	kouč
Ghání	Ghana	nirvana	nirvána
Hodice	hodit,hodina	pětinásobný	pětinásobit
Honda	Honza	ráno	rána,raný
Hořec	hořet	sebelítostný	sebelítostivý
Hroňová	Hron	sociál	sociální
Juste	justice,Just	stotisíc	stotisící
Kopřivík	kopřiva	stěží	stěžovat
Kubíče	kubík	sumatrae	Sumatra
Mása	masa	tatranka	Tatran
Panev	pánev	teigovec	Teigov
Pavlánský	pavlán	vedro	Vedra
Prasek	prase	Čáslavský	Čáslav
Rychtář	rychta	Římský	Řím
Staněk	Staník	Šilhan	Šilhán
Stodo	Stoy	Šušák	Šuša
Tyne	týn	špic	špice

Table A.8: Triplet Loss Suggestions - Roots to merge (small trees with big trees) - suggestions regarding which trees should be merged because their roots contain very similar words. Threshold 15. 58 samples out of 6650 suggestions (1 censored, 1 of them being a correct suggestions