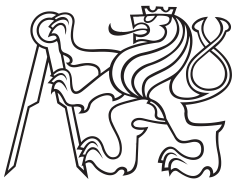
Bachelor's thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Text Summarization Using Named Entity Recognition

Štěpán Műller

Supervisor: Ing. Petr Marek
May 2020

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Műller Štěpán** Personal ID number: **474557**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Branch of study: **Computer and Information Science**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Text Summarization Using Named Entity Recognition

Bachelor's thesis title in Czech:

Sumarizace textu pomocí rozpoznávání pojmenovaných entit

Guidelines:

- Review the state-of-the-art in the problem of named entity recognition and text summarization.
- Propose and implement an algorithm for text summarization using a system for named entity recognition.
- Select appropriate metrics to evaluate the proposed algorithm.
- Propose and conduct experiments to assess the quality of the proposed solution.
- Test the proposed algorithm on selected corpora of text documents and report the results.

Bibliography / sources:

[1] Text Summarization Techniques: A Brief Survey, Mehdi Allahyari et al., ijacsa 2017.
[2] A Review Paper on Text Summarization, Deepali K. Gaikwad at al., International Journal of Advanced Research in Computer and Communication Engineering 2016.
[3] A Survey on Extractive Text Summarization, N.Moratanch et al. ICCCSP 2017.
[4] Text summarization using Wikipedia, Yogesh Sankarasubramaniam et al. Information Processing and Management 2014.
[5] A Survey on Recent Advances in Named Entity Recognition from Deep Learning models, Vikas Yadav et al. COLING 2018.

Name and workplace of bachelor's thesis supervisor:

Ing. Petr Marek, Department of Cybernetics, FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **09.01.2020** Deadline for bachelor thesis submission: **22.05.2020**

Assignment valid until: **30.09.2021**

Ing. Petr Marek	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to thank Ing. Petr Marek for being the best supervisor and for always making time to guide me through every step of this journey. I am also grateful to my girlfriend for her support.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 22, 2020

Abstract

Text summarization is a task in the field of natural language processing that consists of summarizing a text with a shorter and concise text. Summarization systems usually work just with the input text converted to a list of vectors. Our work explores the effect of enriching the input with named entity features. We used a translation neural network architecture, sequence to sequence, to achieve state-of-the-art results on the largest dataset for Czech text summarization, SumeCzech and we found that adding named entities helpes the model achieve better results on out of domain texts. We explored different named entity recognition datasets, methods and frameworks, compared them in terms of speed, memory requirements and F-score and chose the most suitable for text summarization.

Keywords: text summarization, seq2seq, PyTorch, SpaCy, named entity recognition, Czech

Supervisor: Ing. Petr Marek
Czech Institute of Informatics, Robotics and Cybernetics
Jugoslávských partyzánů 1580/3,
16000 Prague 6

Abstrakt

Sumarizace textu je úkol z oblasti zpracování přirozeného jazyka, který spočívá v shrnutí textu kratším a výstižným textem. Sumarizační systémy většinou pracují pouze s textem převedeným na seznam vektorů. Naše práce prozkoumává, jaký vliv má na výkon systému obohacení vstupního textu o příznaky pojmenovaných entit. Využili jsme architekturu neuronové sítě pro překlad, sequence to sequence, dosáhli jsme v současností nejlepších výsledků na největším českém datasetu pro sumarizaci textu zvaném SumeCzech a zjistili jsme, že přidání pojmenovaných entit pomohlo modelu dosahovat lepších výsledků na textech z jiné domény. Prozkoumali jsme různé dostupné datasety, metody a knihovny pro rozpoznávání pojmenovaných entit, porovnali jsme je z hlediska rychlosti, paměťové náročnosti a F-score a zvolili tu nejvhodnější pro sumarizaci textu.

Klíčová slova: sumarizace textu, seq2seq, PyTorch, SpaCy, rozpoznávání pojmenovaných entit, čeština

Contents

1 Introduction 1

2 Related work 3
2.1 NER 3
2.2 Text summarization 3

3 Datasets 5
3.1 NER 5
3.1.1 Czech 5
3.1.2 English 6
3.2 Text summarization 7

4 Text processing 9
4.1 Text representation 9
4.2 Token embeddings 10

5 Statistical models and neural networks 13
5.1 Conditional random field 13
5.2 Recurrent neural networks 13
5.3 Sequence to sequence 14
5.3.1 Attention 15

6 Frameworks 17
6.1 NER frameworks 17
6.1.1 Stanford CoreNLP 17
6.1.2 SpaCy 17
6.1.3 Flair 17
6.2 Neural network frameworks 18
6.2.1 PyTorch 18

7 Evaluation 19
7.1 Tasks 19
7.1.1 NER 19
7.1.2 Text summarization 19
7.2 Strict evaluation 19
7.3 Metrics 20
7.3.1 Rouge 20
7.3.2 ROUGE-RAW 20
7.3.3 F-score 20
7.4 Model performance 21

8 Model 23
8.1 NER 23
8.2 Text summarization 23
8.2.1 NER baseline 23
8.2.2 Seq2seq 24

9 Results 27
9.1 NER 27
9.1.1 English NER 27
9.1.2 Czech NER 28
9.2 Text summarization 29
9.2.1 Examples 30

10 Conclusion 33

Bibliography 35

Figures

3.1 CNEC 2.0 named entities type hierarchy [13]. 6

5.1 A schema of an RNN. x_i are the inputs, y_i the outputs and h_i are the previous states [20]. 14
5.2 The architecture of a sequence to sequence model [24]. 15

Tables

3.1 The number of documents from individual news websites. 8

4.1 The percentage of unknown tokens in titles from train and development parts of SumeCzech when tokenized by different methods. The SentencePiece tokenizer comes with BPEmb and was trained on the Czech Wikipedia. The "N most frequent words" models were trained on texts and titles from the training part of SumeCzech. 10

9.1 F-scores of our trained models are on white background, the rest are reported F-scores of other works. . 28
9.2 CPU time measured in ms per document in validation set. 28
9.3 GPU time measured in ms per document in validation set. 29
9.4 Memory of each system in MB. . 29
9.5 Precisions, recalls, and F-scores of all methods in the official SumeCzech metric, $\text{Rouge}_{\text{RAW}}$. "First - no full stop" is the same as first but if the last character is a full stop, it is removed. FT stands for FastText, BS for beam search. Our methods are on white background, reported baseline methods are on gray background. . 30

Chapter 1

Introduction

Imagine how hard it would be to use books if they had no cover. In 2012, less than 1% of the world's data was analyzed [1]. We need to efficiently utilize the large amount of data that is available to us, and unlabeled data is tough to work with.

But the need to summarize text spans further than just to corporations sitting on a large amount of user data. The internet is a vast ocean of information, and everyone connected to it must surf its waves or drown in it.

What if on top of the common clickbait news headlines, your internet browser would also show you an automatically generated summary, saving you precious time from clicking and reading through the text just to see what the news is actually about?

Text summarization is one of the disciplines in natural language processing (NLP). It is the task of summarizing a text, usually a paragraph or a whole document, with a shorter text, usually a paragraph, a sentence, or just a few words.

Czech text summarization, in particular, has not been deeply explored. We used a translation neural network architecture, sequence to sequence [2], to achieve state-of-the-art results on the largest Czech text summarization dataset, SumeCzech [3].

Named entities are another piece of information that can be extracted from texts and used to label data. For example, if you wanted to search for news articles that mention the company Apple, but not the fruit apple, you would need a tool that can recognize named entities.

Named entities are usually real-world objects that have a "unique" name. A named entity could be a location, a person, or an organization. Examples of named entities are "Prague", "Donald Trump" and "Google". Counterexamples are "bakery", "brother" and "university".

We used our trained named entity recognition (NER) model to label SumeCzech with named entities and trained our text summarization model with and without the additional information about named entities and report results for both cases.

Chapter 2

Related work

2.1 NER

In 2013, state-of-the-art for Czech NER was NameTag [4], open-source software that could learn and classify named entities in text. NameTag used A two-stage maximum entropy classifier with Viterbi decoder. It used many features: form, lemma, tag, the chunk of current and surrounding words in a window, gazetteers, capitalization, punctuation, etc.

In 2016, another important work was made which used neural networks, and even though it used no gazetteers, it still managed to improve the F-score by about 2% [5].

Recently, a breakthrough has been achieved by stacking contextual word embeddings on top of constant word embeddings. Examples of these contextual word embeddings are Flair and BERT for both Czech and English and ELMo for English only. These concatenated vectors are then used as inputs either to an LSTM neural network with a conditional random field decoder or to a sequence to sequence encoder-decoder [6]. These models achieve state-of-the-art results not only on Czech NER tasks but also on the English CoNLL-2003.

2.2 Text summarization

Allahyari et al. [7] provide a brief survey of text summarization. In general, text summarization algorithms can be divided into two categories, *extractive* and *abstractive*.

Extractive summarizers choose pieces from the original text, usually sentences, and combine them to form the summary.

From a high-level point of view, most extractive summarizers follow the same two steps: First, score all the sentences. Then, pick N sentences with the highest score. The main difference between individual extractive methods is how they score sentences.

Summarizing text by choosing sentences from it has an advantage and a disadvantage. The advantage is that no matter how simple the method, the summaries it will produce are always going to be syntactically correct, even

though they may not be good summaries. The disadvantage is that extractive summarizers are limited at what they can predict by the source text.

Abstractive summarizers generate new original text. Abstractive summarizers need to be very complex because, on top of learning to infer the meaning of the input text, they also have to learn to generate syntactically correct text. That is a hard thing to do using hand-written rules. The recent advance of machine learning and, in particular, neural networks, made abstractive summarization the current state-of-the-art.

Allahyari et al. found that there was no completely abstractive summarization method at the time of writing, which was in 2017. However, the current state-of-the-art is completely different. Specialized transformer [8] and sequence to sequence [9] machine translation neural networks achieve the best scores on English text summarization tasks.

Straka et al. propose 4 extractive and 1 abstractive baseline methods for SumeCzech. For the task of extracting headlines from texts, the extractive algorithms pick just one sentence with the highest score from the text, likely because headlines are usually just one sentence long. The proposed algorithms are the following:

- First - Return the first sentence.
- Random - Return a random sentence from the text.
- textrank - Uses Textrank [10] algorithm, which represents text as a graph of sentences based on their similarities and returns the most important sentence.
- clf-rf - Sentences were converted to vectors of features, consisting of the sum of TF-IDF for each word normalized by the sentence length, length of the sentence, cohesion (distance from other sentences), count of capitalized words in the sentence, count of tokens that consist of digits and count of common words that suggests the sentence relates to some other one. Random forest classifier over the features was used to score the sentences and the best one was picked.
- t2t - The only abstractive method, which uses a neural machine translation model of Vaswani et al. [11] Vocabulary of 32 000 word-pieces was used and the model was trained for 8 days.

Chapter 3

Datasets

3.1 NER

3.1.1 Czech

For Czech NER, the largest public dataset is the Czech named entity corpus (CNEC) [12] and it comes in 4 versions:

- CNEC 1.0
- CNEC 1.1 - Fixes minor issues in CNEC 1.0.
- CNEC 2.0 - Specifies slightly different entities than CNEC 1.0 and 1.1.
- CoNLL-based extended CNEC 2.0 - Contains no nested entities.

Entities in CNEC 1.0 - 2.0 can be nested. CNEC 2.0 specifies little different entities and also contains more data. The named entities are classified according to a two-level hierarchy taken from Ševčíková et al. [13], which can be seen in Figure 3.1.

For example, the name Donald is a first name, while also being a personal name. There are 46 types and 8 supertypes in CNEC 2.0. CNEC 2.0 also contains 4 container classes denoted as capital letters, A, C, P, and T. Here is an example of a sentence that has container class A, which stands for an address, in it:

<A<if Dopravní podnik <gu<pf Karlovy> Vary>, a. s.> >

CNEC 2.0 has 35 220 named entities in 8 933 sentences. The data is available in plain, HTML, XML, and treex formats. Here is an example of an annotated document from CNEC 2.0 in plain text, containing a nested entity pf:

Zpívali jí <oa Krásnou <pf Meredith> > , ovšem ;

For the creation of CoNLL-based extended CNEC 2.0, Konkol and Konopík [14] have taken CNEC 2.0, removed nested entities, and mapped it's entities to just 7 supertypes.

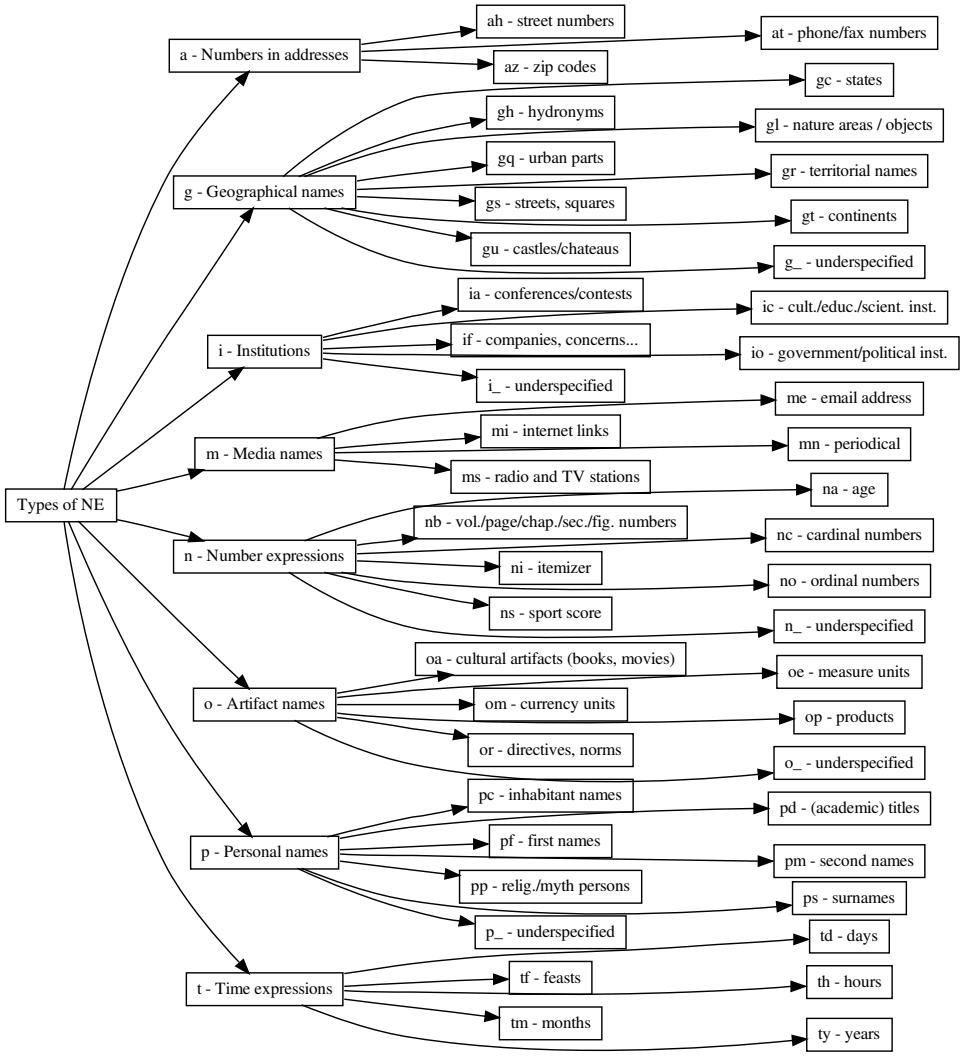


Figure 3.1: CNEC 2.0 named entities type hierarchy [13].

3.1.2 English

NER on the English language is usually benchmarked on the Conference on Natural Language Learning (CoNLL) shared-task from 2003 [15]. CoNLL-2003 for English distinguishes 4 types of named entities: persons, organizations, locations, and miscellaneous names.

It contains 35 089 named entities in 22 134 sentences. The named entities are annotated using the IOB format which assumes that entities are non-recursive and non-overlapping. IOB stands for inside, outside, and beginning respectively. Each line contains four fields: the word, its part-of-speech tag, its chunk tag, and its named entity tag. Here is an example of a document from the training set:

S. NNP I-NP I-MISC
African NNP I-NP I-MISC

Afrikaners NNPS I-NP B-MISC
still RB I-ADVP O
seek VBP I-VP O
own JJ I-NP O
territory NN I-NP O
. . O O

This document contains two entities, <MISC S. African > and <MISC Afrikaners>. Usage of the IOB format can be seen in the example above:

- I-ENTITY - Indicates that this word is a named entity.
- B-ENTITY - Indicates that this word is a named entity but not part of the previous entity. Used only if the previous token was I-ENTITY of the same type.
- O - Indicates that this word is not a part of an entity.

IOB2 is a similar format derived from IOB. The only difference is that the B-ENTITY tag is used at the beginning of every entity, not just in the case specified above. The first 3 tokens, converted to the IOB2 format, would look like this:

S. NNP I-NP B-MISC
African NNP I-NP I-MISC
Afrikaners NNPS I-NP B-MISC

3.2 Text summarization

We focused only on summarization of Czech documents. We used the dataset SumeCzech, which contains more than a million documents, each consisting of a headline, an abstract, and the text. The dataset also contains additional metadata about each document: URL, subdomain, section, and date of publication.

The documents were gathered from Czech news websites. The amount of documents from individual websites is shown in Table 3.1.

SumeCzech is split into four parts. Three of them are the standard train, development, and test parts. To simulate a real-life situation where a model is trained on data from one domain and used on real data from other domains, Straka et al. also created an out of domain (OOD) part. Initially, they clustered the whole dataset into 25 clusters using K-Means on their abstracts and used one of the clusters to create the OOD test set. OOD seems to contain news about concerts and festivals.

The dataset is distributed with a downloader that downloads the data from the Common Crawl Project[1] using the Common Crawl API. The code to evaluate models using the metric suggested by Straka et al. is also provided along with the downloader.

[1] http://commoncrawl.org/

Website	Number	Percentage
ceskenoviny.cz	4 854	0.5%
denik.cz	157 581	15.7%
idnes.cz	463 192	46.2%
lidovky.cz	136 899	13.7%
novinky.cz	239 067	23.9%
Total	1 001 593	

Table 3.1: The number of documents from individual news websites.

Chapter 4

Text processing

4.1 Text representation

Most machine learning algorithms and, in particular, neural networks, require vectors of real numbers as inputs. We could try to represent the whole text as a single vector, but usually, texts are split into *tokens*, and each token is then converted to a vector.

The most popular ways to tokenize a text are word tokenization, character tokenization, and recently, byte-pair tokenization. For example, consider the Czech sentence "Karlův most." (Charles Bridge in English):

- Word tokenization - ["karlův", "most", "."]
- Character tokenization - ["k", "a", "r", "l", "ů", "v", " ", "m", "o", "s", "t", "."]
- Byte-pair tokenization - ["_karl", "ův", "_most", "."]

Character tokenization is nowadays rarely used, at least as the only feature. When representing text as a series of characters, the burden lies on the network to distinguish and learn the meaning of different words. With word tokens, the meaning is already encoded in their embedding and the network just has to learn the meaning of sequences of words. Character-level tokenization is sometimes used as an additional feature on top of word vectors. For example, Straková et al. used character-level embeddings of the first and last two characters of forms as an additional feature to a recurrent neural network [5].

Word tokenization has a drawback when used for neural machine translation because neural networks need a vocabulary of a predetermined size to predict text and the vocabulary would have to be too large if it were to allow the network to predict all the possible words. Byte-pair encodings solve this problem. They know the most common words but are also able to split less-known words into more tokens. The difference in the ability of word tokenization and byte-pair tokenization to cover titles in SumeCzech can be seen in Table 4.1.

Byte-pair tokenization also solves another problem that word tokenization has. It allows us to exactly reconstruct the original text, including spaces.

Model	Vocabulary size	Unknown in train	Unknown in dev
N most frequent words	50000	12.05%	12.53%
N most frequent words	25000	15.11%	15.64%
SentencePiece CZ	25000	0.003%	0.003%

Table 4.1: The percentage of unknown tokens in titles from train and development parts of SumeCzech when tokenized by different methods. The SentencePiece tokenizer comes with BPEmb and was trained on the Czech Wikipedia. The "N most frequent words" models were trained on texts and titles from the training part of SumeCzech.

For example, the sentence above, when reconstructed by joining the word tokens with a space between each one, is "Karlův most .", which contains one extra space. With byte-pair tokenization, when assuming that text cannot start with space, we are able to get the original sentence by joining the tokens and replacing the "_" characters with spaces.

The disadvantage of byte-pair tokenization is similar to that of character tokenization. It is harder for the network to produce syntactically correct text, as it has to learn to predict words that consist of more tokens.

4.2 Token embeddings

The question of how to best convert tokens, be it sentences, words, or byte-pairs to vectors has been of high interest in NLP because better conversion would lead to better results in all NLP tasks.

The simplest kind of word encoding is *one-hot encoding*. Let V be a vocabulary of words we would like to encode and let N be the number of words in the vocabulary. Each word has an index i in the vocabulary and no two words have the same index. Then each word is represented by an N-dimensional vector which has 1 on the i-th position and 0 elsewhere.

A small example of vocabulary with 3 words and their vectors:

$$V = \{cat, ate, dog\}$$
$$V_{cat} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$
$$V_{ate} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$
$$V_{dog} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

The *bag-of-words* model for sentence representation is very similar, it just sums the counts of words in a document to represent it as a multiset (aka bag).

Here is an example of the usage of a case-insensitive bag of words:

s_1="A dog ate a cat."
s_2="A cat ate a mouse"
BoW(s_1) = { "a":2, "dog":1, "ate":1, "cat":1 }
BoW(s_2) = { "a":2, "cat":1, "ate":1, "mouse":1 }

We can then convert these bags to vectors of term frequencies (TF):

$$V_{s_1} = \begin{bmatrix} 2 & 1 & 1 & 1 & 0 \end{bmatrix}$$
$$V_{s_2} = \begin{bmatrix} 2 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Both term frequencies and one-hot encoding lead to very large encodings as the size of vocabulary increases. We would like to encode words to a low dimensional vector space but still maintain "good" properties of the embeddings. This is where word embeddings like Global Vectors for Word Representation (GLoVe) [16] and FastText [17] come in. These models are trained on large corpora to represent words in vector spaces with dimensions 300 with the goal to make words that appear in similar context close to each other in the vector space.

FastText makes further improvements. Unlike previous models, it does not ignore the morphology of words and represents them as a sum of their subwords. This has another advantage, it allows us to represent out of vocabulary words as vectors, which is useful because we are most likely not going to have every possible word in the training corpus. Another advantage is when handling user input or any text that could contain typos in it.

Byte-pair tokens also have pre-trained embeddings, called BPEmb [18], available for 275 languages. They are trained on Wikipedia, using the GLoVe algorithm. Heinzerling and Strube created vocabularies from size 1000 up to 200000, with embedding sizes ranging from 25 to 300.

Chapter 5

Statistical models and neural networks

5.1 Conditional random field

Conditional random field (CRF) [19] models the conditional distribution

$$\mathbf{P}(\mathbf{Y}|\mathbf{X})$$

where X are the observations and Y is the output variable. In our case, Y would be the named entity of a word, and X would be a vector representation of a word or possibly more features, like the previous and following word. Unlike a naive Bayes model, it does not assume that the observations X are independent.

5.2 Recurrent neural networks

Recurrent neural networks (RNN) take as input a sequence of vectors and output a sequence of vectors. A schema of an RNN can be seen in Figure 5.1 [20].

When using RNNs for NER, we can take a sentence that consists of words, convert each word into a vector, and feed it into the RNN. For each word, the output (after additional transformation) is a probability distribution of the word being a certain entity. We train our model so that the distribution is as close to the truth as possible.

The problem with RNNs for NER is that they only give one prediction for each word, which means they cannot predict nested entities. For example, consider this sentence:

<gu<pf Karlovy> Vary>

When the word "Karlovy" is fed into the RNN, what should the output be? Both B-pf and B-gu would miss one entity. We could set our RNN to predict multi-labels, but then the number of named entity output classes would grow combinatorially. To predict nested entities, it is better to use a sequence to sequence model, as shown by Straková et al [6].

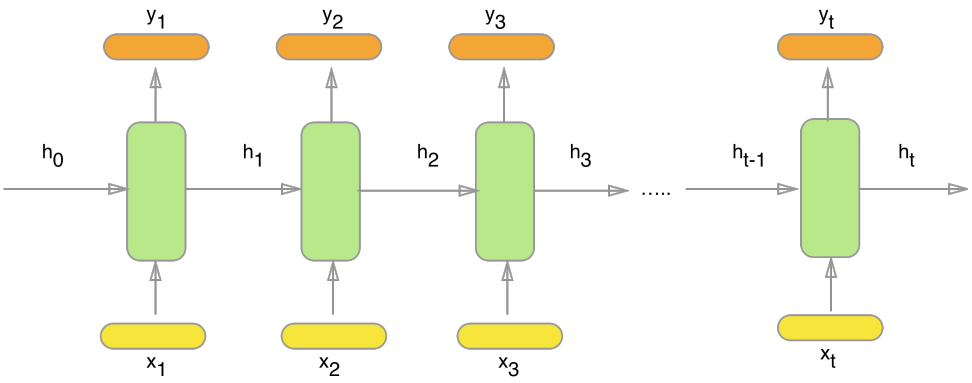


Figure 5.1: A schema of an RNN. x_i are the inputs, y_i the outputs and h_i are the previous states [20].

The most common types of RNNs are GRU [21] and LSTM. [22] LSTMs and GRUs are good for long sequences because they can remember old information well.

LSTMs have become very popular in solving most tasks, as regular RNNs struggled with long-term dependencies [23].

GRUs are faster to train because they have fewer parameters.

RNNs have limited memory and they have to decide, which information is important and which is not as they go through a text. It does not really make sense to have an RNN go only in one direction, from beginning to end, because at the end of the text, it could find that some previous information was important, but it has already forgotten it. Also, RNNs, like humans, tend to remember the most recent information the best. That is why bidirectional RNNs are used. Both forward and backward RNNs are used and their encodings and outputs are concatenated.

5.3 Sequence to sequence

Sequence to sequence [2] models allow converting sequences from one domain to sequences in another domain.

As used for NER by Straková et al. [6], it is a bi-directional LSTM that works as an encoder and then an LSTM working as a decoder. For each word, it outputs a sequence of the corresponding named entity labels until it outputs "<eow>" (end of word) and then moves on to predicting labels for the next word.

Seq2seq models are also widely used in translations, for example from English to French. The task of document summarization can be looked at as a translation from the domain of documents to the domain of titles. The encoder encodes the document as a vector and passes it to the decoder, which outputs words until it outputs "<eos>" (end of sequence). The outputted words are the predicted title.

In our text summarization model, we used a bi-directional GRU as an

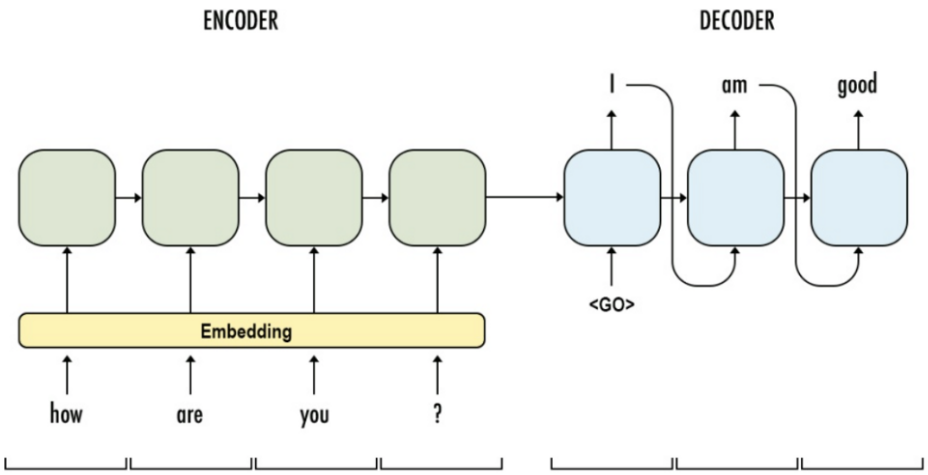


Figure 5.2: The architecture of a sequence to sequence model [24].

encoder and a regular GRU as a decoder.

5.3.1 Attention

A sequence to sequence model as described above would not be very powerful. The fact that the encoder has to represent each input sequence as a single vector creates an informational bottleneck. Previous hidden states of the encoder were ignored and only the last one was used for decoding. That is why *attention* mechanisms are used. Attention allows the decoder to pay attention to different hidden states of the encoder.

During each decoder prediction, the decoder predicts so-called energy for each hidden state of encoder based on its own hidden state and each of the encoder's hidden states. The energy is then transformed using softmax function and a convex combination of encoder hidden states is used on top of the decoder's hidden state to decide which symbol to output.

The approach described above is called *global* attention because it takes into account the entire history of the encoder's hidden states. Luong et al. [33] propose three different ways to calculate the energy for global attention but also suggests using local attention, which only focuses on a small window of encoder hidden states.

Let h_d be the hidden state of the decoder and h_e a hidden state of the encoder. For each hidden state of the encoder, the three suggested ways to calculate energy are the following:

- dot $h_d^\top h_e$
- general $h_d^\top W_g h_e$
- concat $v_g^\top tanh(W_g[h_d; h_e])$

W_g and v_g are the model parameters which are trained to predict energy.

For local attention, the window size is empirically selected. To determine the position of the window, either a monotonic alignment can be used, which assumes that the source and target sequences are monotonically aligned, or the position of the window can be predicted using the decoder's hidden state. These attention systems are called *local-m* and *local-p*.

Let S be the size of the source input and h_d the hidden state of the decoder. The center of the attention window p is predicted as follows:

$$p = S \cdot sigmoid(v_p^\top tanh(W_p h_t))$$

W_p and v_p are the model parameters trained to predict positions.

Chapter 6

Frameworks

6.1 NER frameworks

6.1.1 Stanford CoreNLP

Stanford CoreNLP [25] is an NLP framework written in Java that supports many different tasks. The model that is used in Stanford CoreNLP for NER is CRF. The user can choose the features from which it should try to predict the named entities, but word embeddings are not supported.

6.1.2 SpaCy

SpaCy [26] is an open-source library for natural language processing written in Python. It has a well-documented, simple API, it is fast and has pre-prepared neural network models that are "good" for most problems with little parameters that would need fine-tuning, which makes it easy to use on real-life problems. The downside is that one cannot use this library to conduct experiments, because there are not many parameters to experiment with and the neural networks' architectures cannot be modified. Another disadvantage is that SpaCy only supports static word embeddings like FastText and GLoVe.

6.1.3 Flair

Flair is a simple framework developed by Zalando research that achieves state-of-the-art NLP. It has its own NER architecture, a bi-directional LSTM with possible CRF decoder. A big advantage of this framework is that it supports all the popular word embeddings, from GLoVe to the newest contextual embeddings like Bert and Elmo. Akbik et al. propose their own contextual embeddings, called Flair [27].

Users of this library can experiment with combining different embeddings in a simple manner, without having to change the neural network architecture each time, like they would have to if they were using a machine learning framework like TensorFlow [28] or PyTorch [29].

6.2 Neural network frameworks

6.2.1 PyTorch

We decided to use the PyTorch machine learning framework. It implements many basic neural networks, builds computation graph dynamically, and like many other machine learning frameworks, automatically computes gradients, making neural network learning easy to implement.

PyTorch does not offer a built-in sequence to sequence model, so we used a modified implementation of an example from PyTorch tutorials. The details of our model will be discussed in more detail in the next chapters.

Chapter 7

Evaluation

7.1 Tasks

7.1.1 NER

There are two tasks on the CNEC datasets. The first one only counts as correct correctly predicting the named entity type, while the second counts as correct correctly predicting the correct supertype. As most recent works report only result for types, we are also going to report only results for the types task.

7.1.2 Text summarization

SumeCzech presents three differet tasks:

- abstract → headline
- text → headline
- text → abstract

We focused only on the text → headline task.

7.2 Strict evaluation

There are two ways to evaluate if a named entity was predicted correctly. Consider this correctly labeled sentence:

<gu<pf Karlovy> Vary>

and this prediction:

<gu<pf Karlovy> > Vary

The word "Vary" was incorrectly labeled as not being a part of any entity, so how will we evaluate the incomplete entity Karlovy labeled as gu? In strict evaluation, a named entity's span must exactly match the correct span for

it to be correct, so in this case, the named entity <gu Karlovy> would be counted as a mistake. Strict evaluation is usually used so we are going to also report results using strict evaluation.

7.3 Metrics

7.3.1 Rouge

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation [30]. It is originally a software package that includes metrics to automatically determine the quality of a summary by comparing it to other (ideal) summaries created by humans. These metrics are often used for evaluating summarizations and machine translations.

ROUGE-N measures the overlap of N-grams between the system and reference summaries.

ROUGE-L looks at the longest common subsequence between the reference and the predicted sequence.

N-grams of a text are all sequences of n-items. For example, for the sentence "A dog ate a cat.", the word 1-grams and 2-grams are the following:

- word 1-grams - ["a", "dog", "ate", "a", "cat", "."]
- word 2-grams - [["a", "dog"], ["dog", "ate"], ["ate", "a"], ["a", "cat"], ["cat", "."]]

7.3.2 ROUGE-RAW

Rouge only looked at recall and was English-specific, employing English stemmer, stop words, and synonyms. Straka et al. [3] propose $\text{ROUGE}_{\text{RAW}}$, a language-agnostic variant of ROUGE, which utilizes no stemmer, no stop words, and no synonyms. It measures not only recall but also precision and F-score.

Straka et al. reported the precision, recall, and F-score for their baseline algorithms using $\text{ROUGE}_{\text{RAW}}$-1, $\text{ROUGE}_{\text{RAW}}$-2, and $\text{ROUGE}_{\text{RAW}}$-L metrics. In order to get comparable results, we report the same metrics for our models.

7.3.3 F-score

F-score is defined as the harmonic mean of precision and recall:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$\text{precision} = \frac{\text{true positives}}{\text{false positives} + \text{true positives}}$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

For a prediction "A dog ate dog .", and it's reference sentence "A dog ate a cat .", all of the 5 prediction 1-grams are in the reference sentence 1-grams, but there is one problem. The word "dog" appears one too many times, so only 4 of them are correct and the $\text{Rouge}_{\text{RAW}}$-1 precision is 80%. On the other hand, the 1-grams "cat" and the second "a" are missing in the prediction, so 4 out of 6 reference 1-grams are predicted correctly and the recall is $\approx 66.7\%$, making the F-score 72.7%.

$\text{Rouge}_{\text{RAW}}$-2 would be calculated similarly.

For $\text{Rouge}_{\text{RAW}}$-L, let us first formally define a subsequence. A sequence $S = [s_1, s_2, ..., s_n]$ is a subsequence of another sequence $X = [x_1, x_2, ..., x_m]$, if there exists a strictly increasing sequence $[i_1, i_2, ..., i_n]$ of indices of X so that for all $j = 1, 2, ...n$, $x_{i_j} = s_j$ [30].

To calculate $\text{Rouge}_{\text{RAW}}$-L, first, the longest common subsequence is calculated. In this case, it is "A dog ate ." It is 4 word tokens long. The prediction is 5 tokens long and the reference is 6 tokens long, so the $\text{ROUGE}_{\text{RAW}}$-L precision is 80% and the recall is $\approx 66.7\%$, making the F-score 72.7%.

7.4 Model performance

We measured the performance of different NER frameworks and compare their average CPU time per document in the validation sets of each model for each framework. We also measured GPU time per document of all Flair models. CPU and GPU times were averaged over documents in the corresponding validation datasets over 10 runs. The hardware used was the following:

- CPU - Intel i7-8750h (16 x 2.20 GHz)
- GPU - NVIDIA GeForce GTX 1060 6GB

We measured the models' memory requirements using Performance monitor [31] and report their working set peak during the prediction of validation data.

Chapter 8

Model

8.1 NER

We chose CoNLL-based extended CNEC 2.0 as the training dataset for our NER model, as it is the largest and most up-to-date Czech NER dataset and because it contains no nested entities, which makes it easier to connect the NER model to our summarizer.

As our NER model architecture, we have chosen SpaCy's NER model because it is much faster than other frameworks, making it more applicable, while still achieving a good score.

We labeled the SumeCzech dataset with NEs in IOB2 format, one label for each word token.

To use NEs with byte-pair tokenization, we used the approach recommended in the BERT GitHub repositary[1] readme: "If you have a pre-tokenized representation with word-level annotations, you can simply tokenize each input word independently, and deterministically maintain an original-to-tokenized alignment."

Then we trained our text summarization model with and without the additional information about named entities in text and report both results.

8.2 Text summarization

8.2.1 NER baseline

We propose a baseline extractive text summarization method that utilizes named entity recognition to see whether information about named entities can be useful for text summarization.

We hypothesized that sentences that mention a lot of named entities carry a lot of information about what happened. Informally, a sentence that contains the who, where, and when, is likely going to be a better summary than a sentence that does not.

[1] I would like to thank the user "dsindex" for pointing that out at https://github.com/google-research/bert/issues/560

Sentences are scored by the number of word tokens that are parts of named entities, divided by their total length in words. The highest scoring sentence is picked as the title.

8.2.2 Seq2seq

We used a modified implementation of a seq2seq model with global attention from the official PyTorch tutorial [32]. The hidden sizes of the encoder and decoder were 256. The size of our vocabulary was 25000. We tried both word and byte-pair tokenization with embeddings of size 300. We used FastText for embedding words and BPEmb for embedding byte-pairs. We used dropout 0.1 on the outputs of both RNNs.

Our NER model distinguishes 7 entity types in the IOB2 format, with beginning and inside tags for each, so our NER vocabulary consists of 14 words for entities, 1 for outside of entity, 1 for padding, and 1 for start and end of sequence symbols. To utilize named entities, we concatenated these one-hot encoded entity vectors of dimension 17 to the word embeddings and used them as inputs to the encoder.

We trained our model on Tesla K80 GPU. Even though the GPU has 12 GB memory, we ran into memory issues during training, because our input sequences, being news, were very large, the longest text in train and development parts of the dataset having 17 506 word tokens and 23 103 byte-pair tokens.

The implementation in the tutorial used an approach to attention similar to concat global attention. For good utilization of the parallel nature of GPUs, they would copy the decoder's hidden state for each hidden state of encoder, then concatenate the hidden states with the encoder outputs and pass them to a linear layer to calculate energy.

We used batch size 16, so in case of the longest sentence, the concatenated tensor would have 147 890 688 floating numbers if word tokenization was used and 193 738 312 if byte-pair encoding was used.

To be able to train our model with attention without running out of memory on our GPU, we had to simplify how attention was calculated. We used an approach similar to general global attention. To calculate energy, an affine transformation on the decoder's hidden state was used to transform it into a vector of dimension 64. The hidden states of the encoder were also affinely transformed to vectors of the same dimension by a linear layer. The transformed decoder hidden state was then used as a multiplier and broadcast over all the encoder hidden states, making the calculation of energy much more memory efficient, because the decoder's hidden state did not have to be copied.

Even with our simplified attention, we were not able to finish the training of models with the byte-pair tokenizer. We still had to reduce batch size so that our GPU would not run out of memory, which made the training almost two times slower. The training was also slower because the model had to process more tokens. As for the model with byte-pair tokenization and NEs, we could not fit the entire dataset into our 64 GB memory. We

had to read the dataset from disk during each epoch, which slowed down the training even more, and we couldn't shuffle it, which would make its results inconsistent with the others. Considering all of these problems, we ended up not including byte-pair encoding models, as they were not necessary for our goals, and we only trained word tokenizing models.

We trained our models until validation loss started increasing and we picked the one with the lowest validation loss for evaluation.

Because word tokenization lead to a lot of unknown words in the titles, the model that used word tokenization learned to predict them. We had to forbid the model from predicting unknown tokens during evaluation to get meaningful titles.

We also report results of our models with and without beam search of size 8.

Chapter 9

Results

9.1 NER

The results of our trained models, compared with the results from other works, can be seen in Table 9.1. The average inference time in CPU is in Table 9.2 and the inference time in GPU is in Table 9.3. Memory usage of each model is in Table 9.4

9.1.1 English NER

There was a problem when training Stanford CoreNLP on CNEC because the memory requirements grow with the increasing amount of named entity classes. We were able to train it on the datasets with up to 7 entity types (CoNLL-2003, CoNLL-based Extended CNEC 2.0), but failed to train it on the CNEC 1.0/1.1/2.0 datasets even on a machine with 64 GB ram.

Flair had the most accurate results from the English libraries but was about 300 times slower than SpaCy. SpaCy had the second-best results and was the fastest. Stanford CoreNLP had the worst score while also being about 30 times slower than SpaCy.

SpaCy is most likely faster than Stanford CoreNLP because most of its complex computations are executed in an underlying optimized machine learning library which transforms python code into C and compiles it, while Stanford CoreNLP runs on a Java virtual machine.

Flair builds on PyTorch, so it should also be fast, but it is significantly slower than SpaCy. The reason is that while SpaCy just fetches word embeddings from a pre-trained static word embedding model, which basically works like a dictionary, Flair has to compute its contextual embeddings for each sentence every time it sees one because the word embeddings are different depending on other words in the sentence. The embeddings are cached for training because the same sentences are forwarded through the network in each epoch, but this idea can hardly be utilized in most real-life applications.

We have chosen SpaCy as our model for labeling SumeCzech because it would take Flair a considerable time to label SumeCzech. It takes Flair more than 100 ms to label a single document from the Extended CNEC dataset, and those documents are usually just a sentence, while SumeCzech contains

Model	CNEC 1.0	1.1	2.0	Extended CNEC 2.0	CoNLL-2003
NameTag	79.23	-	-	-	89.16
LSTM Straková et al.	81.2	-	79.23	80.79	89.92
LSTM-CRF Straková et al.	85.7	-	-	**86.39**	**93.38**
Seq2seq Straková et al.	**86.88**	**86.88**	**86.23**	85.92	93.07
SpaCy	76.76	76.83	76.43	78.45	88.38
Stanford CoreNLP	-	-	-	73.27	84.92
Flair	80.51	81.15	79.99	83.76	90.9

Table 9.1: F-scores of our trained models are on white background, the rest are reported F-scores of other works.

Model	CNEC 1.0	1.1	2.0	Extended CNEC 2.0	CoNLL-2003
SpaCy	**2.7**	**2.6**	**2.6**	**2.4**	**2.1**
Stanford CoreNLP	-	-	-	63.1	45.7
Flair	779.3	724.9	663.4	659.5	427.7

Table 9.2: CPU time measured in ms per document in validation set.

almost a million documents, each one having tens or hundreds of sentences.

If fast inference or high throughput is needed, SpaCy would be the best choice, because it achieves decent results with blazing fast inference time. For scientific experiments and in a situation where making a mistake would be costly, Flair seems to be the best choice. We do not recommend using Stanford CoreNLP in new applications, as SpaCy achieves both better accuracy and better F-score.

9.1.2 Czech NER

Neural Architectures for Nested NER through Linearization [6] achieves the best results for the Czech language and also for English NER by concatenating three different contextual embeddings. LSTM-CRF works better for flat corpora and seq2seq works better for corpora with nested entities.

We report results on the CoNLL-based extended CNEC 2.0 for both the Seq2Seq and the LSTM-CRF model. We also report their measured GPU time and memory, but these results are incomparable with the other frameworks. Their implementation requires contextual embeddings to be calculated and saved before the start of the process. The vectors are then loaded into memory before the inference, which means that the memory of the process is going to be much higher, as all the vectors for all the words in the dataset will be stored there. As a result, inference is going to be much faster than it would be in real usage because the contextual embeddings do not have to be calculated as the model receives the sentence.

Using more contextual embeddings seems to lead to better results, as these models had better performance than all the English libraries, which do not support contextual embeddings except for Flair, for which we used one contextual embedding.

Model	CNEC 1.0	1.1	2.0	Extended CNEC 2.0	CoNLL-2003
LSTM-CRF Straková et al.	-	-	-	-	**20.3**
Seq2seq Straková et al.	-	-	-	-	30.3
Flair	134.8	134.2	119.6	119.5	82.2

Table 9.3: GPU time measured in ms per document in validation set.

Model	CNEC 1.0	1.1	2.0	Extended CNEC 2.0	CoNLL-2003
LSTM-CRF Straková et al.	-	-	-	11 391	-
Seq2seq Straková et al.	-	-	-	10 312	-
SpaCy	3 737	3 736	3 755	3 745	3 745
Stanford NLP	-	-	-	4 159	3 810
Flair	2 950	2 949	2 950	2 949	1 425

Table 9.4: Memory of each system in MB.

9.2 Text summarization

The results of our methods, compared with the results of the baseline SumeCzech methods, can be seen in Table 9.5. For comparison and as a check, we tried replicating the reported results of the baseline methods suggested by Straka et al.

We compare our results to the results in the readme file of the SumeCzech downloader because those are updated to match the tokenization of the accompanying evaluation code and are slightly different than the results reported in [3]. The results for the CLF-RF baseline were not updated, partly because they were not deemed interesting enough (M Straka 2020, personal communication, 18 May), so we have not included it in the comparison.

Our baselines have very similar results, the slight difference could be explained by different tokenization. We used NLTK [34] tokenizers to tokenize both words and sentences.

NER baseline achieves slightly better results than random, suggesting that NEs could indeed help with text summarization. It is not better than first. However, first is a tough baseline to beat, as the first sentence in a news article often summarizes the text.

Our sequence to sequence model with FastText word embeddings achieved state-of-the-art results on both test and OOD splits of the dataset. Same as other models, it's performance on the OOD set was worse than on the test set.

Textrank baseline still had the highest recall in almost all cases, but its precision was very low, similar to random.

Adding information about named entities seems to help the model generalize better. The model with named entities was slightly worse on the test set and slightly better on the OOD set. Overall, the model with named entities performed better.

It is possible that the model with NEs just got lucky, as neural networks' learning is not deterministic and usually, their results will differ every time they are trained, even if the architecture remains the same.

Beam search helps the model predict more human-like sentences, which

Dataset	Classifier	Rouge-Raw 1			Rouge-Raw 2			Rouge-Raw L		
		P	R	F	P	R	F	P	R	F
Test	First	7.4	13.5	8.9	1.1	2.2	1.3	6.5	11.7	7.7
	Random	5.9	10.3	6.9	0.5	1.0	0.6	5.2	8.9	6.0
	Textrank	6.0	**16.5**	8.3	0.8	2.3	1.1	5.0	**13.8**	6.9
	Tensor2Tensor	8.8	7.0	7.5	0.8	0.6	0.7	8.1	6.5	7.0
	First	7.1	12.9	8.5	1.2	2.2	1.4	6.2	11.3	7.4
	First - no full stop	7.5	13.4	8.9	1.1	2.1	1.3	6.5	11.7	7.8
	Random	5.9	10.0	6.8	0.5	0.9	0.6	5.2	8.7	6.0
	NER baseline	6.4	9.9	7.0	0.8	1.4	0.9	5.7	8.7	6.2
	Seq2seq-FT	**15.4**	13.7	**14.1**	2.4	2.1	2.1	**13.9**	12.4	**12.8**
	Seq2seq-FT-NER	15.3	13.6	14.0	2.4	2.0	2.1	**13.9**	12.4	12.7
	Seq2seq-FT-BS	15.2	12.7	13.5	2.9	**2.4**	**2.6**	**13.9**	11.7	12.4
	Seq2seq-FT-BS-NER	15.1	12.6	13.4	**3.0**	**2.4**	**2.6**	**13.9**	11.6	12.3
OOD	First	6.7	13.6	8.3	1.3	2.8	1.6	5.9	12.0	7.4
	Random	5.2	10.0	6.3	0.6	1.4	0.8	4.6	8.9	5.6
	Textrank	5.8	**16.9**	8.1	1.1	**3.4**	1.5	5.0	**14.5**	6.9
	Tensor2Tensor	6.3	5.1	5.5	0.5	0.4	0.4	5.9	4.8	5.1
	First	6.6	13.2	8.1	1.4	2.9	1.7	5.9	11.8	7.2
	First - no full stop	6.8	13.5	8.4	1.4	2.8	1.7	6.0	12.0	7.4
	Random	5.3	9.9	6.3	0.7	1.4	0.8	4.7	8.8	5.6
	NER baseline	6.2	10.6	6.8	1.3	2.3	1.4	5.6	9.5	6.2
	Seq2seq-FT	12.6	11.4	11.6	1.9	1.6	1.7	11.7	10.7	10.8
	Seq2seq-FT-NER	**13.0**	11.6	**11.9**	1.9	1.7	1.7	**12.0**	10.8	**11.0**
	Seq2seq-FT-BS	12.2	10.1	10.7	2.3	1.8	1.9	11.4	9.4	10.0
	Seq2seq-FT-BS-NER	12.5	10.1	10.8	**2.4**	1.9	**2.0**	11.7	9.7	10.1

Table 9.5: Precisions, recalls, and F-scores of all methods in the official SumeCzech metric, $\text{Rouge}_{\text{RAW}}$. "First - no full stop" is the same as first but if the last character is a full stop, it is removed. FT stands for FastText, BS for beam search. Our methods are on white background, reported baseline methods are on gray background.

can be seen in the increase of the models' $\text{ROUGE}_{\text{RAW}}$-2 score, but since it focused on the most probable sequences and not on the most probable words, their $\text{ROUGE}_{\text{RAW}}$-1 and $\text{ROUGE}_{\text{RAW}}$-L scores decreased.

9.2.1 Examples

We have chosen a few examples from the Test and OOD datasets to show how different methods summarize. For convenience, we also provide English translation.

Only very simple post-processing was done on the output of our seq2seq models. Filtering of the start of sentence and end of sentence symbols, removing spaces before punctuation, stripping the text of any starting or ending space, and then capitalizing the first letter. It was all automatic.

In this example, NER baseline predicted a sentence with numbers in it. Seq2seq-FT-NER has returned a semantically correct sentence, unfortunately with a bit of a different meaning. Seq2seq-FT-BS and Seq2seq-FT-BS-NER both predict a good summary.

- Gold: "Trendy podlahy vyzývají ke kreativitě i návratu k přírodě"

("Floor trends call for creativity and a return to nature")

- First: "Volba podlahy do interiéru je zásadní."

  ("Choosing the right floor for the interior is crucial.")

- NER baseline: "Cena 1074 Kč/m2."

  ("Price 1074 kč/m2.")

- Seq2seq-FT: "Jak vybrat správný vybrat správný domov?"

  ("How to choose the right choose the right home?")

- Seq2seq-FT-BS: "Jak vybrat kvalitní podlahu"

  ("How to choose a quality floor")

- Seq2seq-FT-NER: "Jak vybrat správný byt?"

  ("How to choose the right apartment ?")

- Seq2seq-FT-BS-NER: "Jak si vybrat správný podlahu"

  ("How to choose the right floor")

The first sentence in the following document is very similar to gold, however, there is also some garbage at the start of the sentence that gets returned with it. Seq2seq-FT-NER returns a nonsensical sentence. NER baseline chooses a sentence that contains many names and carries a lot of information. Unfortunately, the sentence is not very similar to the headline.

- Gold: "Naposled na sebe vezmu masku, říká Bale před uvedením Nolanova Batmana"

  ("I'll wear a mask for the last time, Bale says in front of a picture of Nolan's Batman")

- First: "Zobrazit fotogalerii"Rozhodl jsem se, že to bude naposledy, co si na sebe vezmu masku Batmana," řekl Bale podle agentury Reuters při interview v Beverly Hills."

  ("Show photogallery"I have decided, that this will be the last time that I am wearing Batman's mask", said Bale according to the Reuters agency during an interview in Beverly Hills.")

- NER baseline: "Vedle Balea coby Batmana se objeví Michael Caine jako Batmanův komorník a Gary Oldman v úloze komisaře Gordona."

  ("Alongside Bale as Batman, Michael Caine will appear as Batman's valet and Gary Oldman as Commissioner Gordon.")

- Seq2seq-FT: "Herec a další konec....."

  ("An actor and another end.....")

- Seq2seq-FT-BS: “Na podzim bude hrát v novém roce 2017, bude hrát v lednu”

  (In the fall he will play in the new year 2017, he will play in January)

- Seq2seq-FT-NER: “V londýně se vrátí do konce roku. bude hrát i s s”

  (“He will return in London by the end of the year. Will play even with with”)

- Seq2seq-FT-BS-NER: “James bond se vrací na scénu, bude hrát v březnu”

  (“James Bond returns to the scene, he is going to play in March”)

In the last example, both Seq2seq-FT-NER and Seq2seq-FT-BS-NER create a good news article headline, like a human would, but in other words than gold:

- Gold: Hrad Bouzov nadchne cyklisty i zájemce o mučení a draky

  (“Bouzov Castle will delight cyclists and those interested in torture and dragons”)

- First: “Nová prohlídková trasa nazvaná Draci a drakobijci prochází severním sklepením hradu.”

  (“A new sightseeing route called Dragons and dragonslayers passes through the northern cellar of the castle.”)

- NER baseline: “Pátek 31. srpna Areál hradu otevřen od 18:00 Karnevalový zahajovací večírek v hradních prostorách – každá maska dostane dárek.”

  (“Friday, August 31 The castle grounds open from 6:00 pm Carnival opening party in the castle premises - each mask will receive a gift.”)

- Seq2seq-FT: “Hrady a zámky na hrad. kde se můžete vidět i na hrad”

  (“Castles and chateaux on the castle. where you can see yourself even on the castle”)

- Seq2seq-FT-BS: “Tipy na výlet na výlet na výlet do hradu”

  (“Tips for a trip for a trip for a trip to the castle”)

- Seq2seq-FT-NER: “Na kole na hrad”

  (“Ride a bike to the castle”)

- Seq2seq-FT-BS-NER: “Na výlet na hrad”

  (“A trip to the castle”)

Chapter 10

Conclusion

Text summarization is usually done directly from text converted to vectors without additional information extraction. We explored whether feeding information about recognized named entities in text to a machine translation model could lead to an improvement.

We looked at several NER datasets, tested several NER frameworks, and chosen the most suitable combination for our task. The advantages of different frameworks should now be clear, especially when considering the right framework to use for the Czech language.

SpaCy is the fastest and most suitable for practical applications and Flair is the best performing and is most suitable for experimentation. Since it takes Flair about 100 ms to label a document on a GPU and about half a second on a CPU, it is hardly usable for big data labeling.

Our results can be taken into account even when considering which framework to choose for a different task, as all of them support more tasks than just NER.

We looked at Czech text summarization and measured how the performance of summarization methods can be enhanced by adding information about named entities. To find out, we evaluated a baseline algorithm that utilized NER and also trained and measured the performance of a seq2seq model with and without NEs.

Our baseline algorithm was slightly better than random, suggesting that NE tags do indeed carry some information.

By training and evaluating our seq2seq model, we have seen that NEs help the model generalize better, making it more applicable because, in practice, it is likely going to be used on texts from other domains. When using SpaCy, tagging named entities does not represent a huge computational overhead.

Our text summarization model only used static embeddings, but since we have seen how contextual embeddings have improved NER, it would be interesting to see if concatenating more types of embeddings could lead to a similar improvement in text summarization.

Testing out different attention mechanisms could also lead to further improvements.

Bibliography

[1] Burn-Murdoch, J. (2012), 'Study: less than 1% of the world's data is analysed, over 80% is unprotected', The Guardian, 19 December. Available at: https://www.theguardian.com/news/datablog/2012/dec/19/big-data-study-digital-universe-global-volume Accessed: 20 May 2020

[2] Sutskever, I., Vinyals, O. and Le, Q.V., 2014. Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

[3] Straka, M., Mediankin, N., Kocmi, T., Žabokrtský, Z., Hudeček, V. and Hajic, J., 2018, May. SumeCzech: Large Czech News-Based Summarization Dataset. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018).

[4] Straková, J., Straka, M. and Hajic, J., 2014, June. Open-source tools for morphology, lemmatization, POS tagging and named entity recognition. In Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations (pp. 13-18).

[5] Straková, J., Straka, M. and Hajič, J., 2016, September. Neural networks for featureless named entity recognition in Czech. In International Conference on Text, Speech, and Dialogue (pp. 173-181). Springer, Cham.

[6] Straková, J., Straka, M. and Hajič, J., 2019. Neural architectures for nested NER through linearization. arXiv preprint arXiv:1908.06926.

[7] Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E.D., Gutierrez, J.B. and Kochut, K., 2017. Text summarization techniques: a brief survey. arXiv preprint arXiv:1707.02268.

[8] Takase, S. and Okazaki, N., 2019. Positional encoding to control output sequence length. arXiv preprint arXiv:1904.07418.

[9] Yan, Y., Qi, W., Gong, Y., Liu, D., Duan, N., Chen, J., Zhang, R. and Zhou, M., 2020. ProphetNet: Predicting Future N-gram for Sequence-to-Sequence Pre-training. arXiv preprint arXiv:2001.04063.

[10] Mihalcea, R. and Tarau, P., 2004, July. Textrank: Bringing order into text. In Proceedings of the 2004 conference on empirical methods in natural language processing (pp. 404-411).

[11] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[12] Kravalová, J. and Žabokrtský, Z., 2009, August. Czech named entity corpus and SVM-based recognizer. In Proceedings of the 2009 Named Entities Workshop: Shared Task on Transliteration (pp. 194-201). Association for Computational Linguistics.

[13] Ševčíková, M., Žabokrtský, Z. and Krůza, O., 2007, September. Named entities in Czech: annotating data and developing NE tagger. In International Conference on Text, Speech and Dialogue (pp. 188-195). Springer, Berlin, Heidelberg.

[14] Konkol, M. and Konopík, M., 2013, September. CRF-based Czech named entity recognizer and consolidation of Czech NER research. In International conference on text, speech and dialogue (pp. 153-160). Springer, Berlin, Heidelberg.

[15] Sang, E.F. and De Meulder, F., 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. arXiv preprint cs/0306050.

[16] Pennington, J., Socher, R. and Manning, C.D., 2014, October. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[17] Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T., 2017. Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics, 5, pp.135-146.

[18] Heinzerling, B. and Strube, M., 2017. Bpemb: Tokenization-free pre-trained subword embeddings in 275 languages. arXiv preprint arXiv:1710.02187.

[19] Lafferty, J., McCallum, A. and Pereira, F.C., 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[20] Kostadinov, S. 2017. Recurrent neural network, viewed 20 May 2020, <https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7>

[21] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

[22] Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. Neural computation, 9(8), pp.1735-1780.

[23] Bengio, Y., Simard, P. and Frasconi, P., 1994. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), pp.157-166.

[24] Sequence to sequence, viewed 20 May 2020, <https://cdn-images-1.medium.com/freeze/max/1000/1*QWVZX2bVBK5tHOgDJK38aA.png>

[25] Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J.R., Bethard, S. and McClosky, D., 2014, June. The Stanford CoreNLP natural language processing toolkit. In Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations (pp. 55-60).

[26] Honnibal, M. and Montani, I., 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. To appear, 7(1).

[27] Akbik, A., Blythe, D. and Vollgraf, R., 2018, August. Contextual string embeddings for sequence labeling. In Proceedings of the 27th International Conference on Computational Linguistics (pp. 1638-1649).

[28] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M., 2016. Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) (pp. 265-283).

[29] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A., 2019. PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems (pp. 8024-8035).

[30] Lin, C.Y., 2004. ROUGE: a Package for Automatic Evaluation of Summaries. In Proceedings of the Workshop on Text Summarization Branches Out (WAS 2004)

[31] Performance monitor. (1993). Microsoft.

[32] Weidman, S 2017, Language Translation With TorchText, viewed 20 May 2020, <https://pytorch.org/tutorials/beginner/torchtext_translation_tutorial.html>

[33] Luong, M.T., Pham, H. and Manning, C.D., 2015. Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025.

[34] Bird, S., Klein, E. and Loper, E., 2009. Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc.".