EBERHARD KARLS UNIVERSITÄT TÜBINGEN
SEMINAR FÜR SPRACHWISSENSCHAFT

INTERNSHIP REPORT

July 22, 2019

Submitted by:
Marko Ložajić
marko.lozajic@student.uni-tuebingen.de

Academic Supervisor:
Dr. Çağrı Çöltekin
ccoltekin@sfs.uni-tuebingen.de

Company:
Dekonta d.o.o.

Supervisor:
Aleksandar Ložajić
a.lozajic@dekonta.rs

Contents

1 INTRODUCTION

1.1 The Company

Dekonta d.o.o. is the Serbian sister company of the Czechia-based Dekonta a.s., whose offices are located in Belgrade. It specializes in environmental consulting, engineering and occupational health and safety services.

My internship here was spread over six weeks in February and March 2019. I was tasked with constructing an automated system for legal document updates, and was largely able to define my own work plans and strategy for achieving this goal.

1.2 The Task

When conducting field work, one needs to be aware of the exact laws and regulations of the country in order to be able to assess whether the client's operations are in agreement with the local legal regulations. This is of central importance in Dekonta's work, and as such, there is a constant need for up-to-date and clear legal guidelines.

During regular parliamentary proceedings, changes to existing laws are often proposed and noted down. For the sake of clarity, these changes often need to be included in the "consolidated" versions of laws, adjusted according to the proposed changes. In Serbia, this process is currently being performed manually, but in many countries in the region where Dekonta is operating, these consolidated versions are not created at all. It would therefore be a tremendous business aid, and a potential product as well, to develop a program that would be able to perform this task with minimal human supervision.

The task I was assigned during the internship was to develop precisely this system, that would read and "understand" natural language instructions on how a certain law ought to be amended, and perform the described amendments in the corresponding legal document.

2 PROJECT

Since there was nobody in the company with higher qualifications in computer science or linguistics-related fields (most of the employees were engineers and geologists), I was largely left "to my own devices" in coming up with a work plan. The plan I came up with initially was, with a few minor deviations, the one I ended up pursuing, and which this report is based on. It involved:

- translating the natural language instructions into an intermediate language (instructions in this language henceforth referred to as *annotations*)
- using the instruction-annotation pairs as input for a sequence-to-sequence (encoder-decoder) neural network

- standardizing the HTML of the legal documents, and
- developing a rule-based system for performing the changes described by the annotations in the corresponding HTML file

2.1 Annotations

The first difficulty I perceived was that there was substantial variability in the language of the instructions. Although the changes must follow a certain structural norm, the exact terms used and linguistic details such as word order and long-distance anaphoras made a strictly syntactic approach seem unfeasible, and a semantic approach plausible but daunting. I therefore opted for adopting a sequence-to-sequence [1] approach, and in order to relate my work more easily to ongoing research, reframed the legal update problem as a sort of machine translation task, where each instruction was to be "translated" into a concise, intermediate language, which would be used to represent freely-worded natural language instructions in the shortest possible form without introducing ambiguity. This standardized interlanguage would then be read by a rule-based system, which would be able to parse them with much less difficulty than their natural language counterparts.

With the above rationale in mind, I designed the language, together with a short "instruction manual" (Appendix) containing example translations and discussion of potentially problematic cases. Being aware that the programming requirements for the task at hand would be rather time-consuming, my supervisor agreed to hire several annotators for a few days and translate all the available data, which would eventually serve as input to the sequence-to-sequence model. The end result were not entirely consistent annotations, but were nonetheless extremely helpful and provided me with a lot of valuable data, in which I ironed out the occasional inconsistency upon encountering unexpected results when running the nascent program.

2.2 Data Set Description

Since the generation of annotated data was quite expensive, the training data turned out to be rather modest in size - only 865 sentence-annotation pairs were available to the model at training time, 97 were used for monitoring validation loss (development set), and 398 samples were used for testing. The lengths of the input sentences had a very large variance, spanning the range between 17 and 580 characters, with a mean of 61. The annotations, too, saw a very large disparity in length between the shortest and longest ones; the shortest annotation was only 2 characters long, the average one had 6.4 characters, whereas the longest one - which was in fact a concatenation of multiple annotations in the same paragraph - consisted of an enormous 103 characters. Already here it is clear that the encoder-decoder model could have a difficult time translating such long sequences, especially since long-distance dependencies have been shown to cause problems for similar models that were considered state-of-the-art

at the time [2].[1]

The distribution of the different operations in the training data was the following; 118 delete instructions, 428 replace instructions, 258 insert instructions and 61 miscellaneous ones. All the instructions labeled with **NNN** (unknown annotation) were omitted.

2.3 Preprocessing

The goal of the preprocessing step was to extract the instructions from their respective files, and associate them with the corresponding annotations. Sections of the instructions that were enclosed in quotation marks also had to be treated differently, since the annotations "ignored" them, and were designed to deal with them implicitly. Determining what passages were enclosed within quotes proved to be quite a difficult task, due to issues such as nested quotes, inconsistent use of Unicode characters for opening quotes, and in some cases even unopened/unclosed quotes, which often led to the entire file being parsed incorrectly. An unexpected benefit of the project thus emerged, in that missing quotation marks, sometimes causing significant structural ambiguity, could be detected, and temporarily fixed, although an additional reading from a legal professional might be required in some cases to ensure the "more intuitive" reading was indeed the intended one.

2.4 Input Representation

The training data for the neural network were the natural language instructions, and the target data were the annotations. In this way, the problem is largely analogous to machine translation, for which encoder-decoder networks coupled with a gating mechanism have been yielding promising results in recent years, as exemplified by the work of Cho et al. [3] and Bahdanau et al. [4]. Of several different ways of representing the input that were attempted, including word embeddings and one-hot character encodings, character embeddings proved to be most performant. The character embeddings were extracted from FastText word embeddings [5] and constructed using a "bag-of-characters" technique, that is by averaging the word vectors in which a certain character appeared, as suggested by Max Woolf [6], where characters that tended to co-occur in the provided words would end up with relatively similar embeddings. The resulting embeddings were reduced from the initial dimensionality of 300 to a more computationally accommodating 64, by means of principal component analysis. Due to the inordinate quantity of padding dictated by the longest instructions - 580 characters was the longest instruction length while the average length was only 61 characters - an approach knows as "bucketing" was adopted, where the sequences were sorted by length and split into batches, each of which were then padded not to the length of the longest instruction overall, but only to the length of the longest instruction within that batch. Splitting the data into batches of size one would have enabled the input

[1] Although the referenced paper was dealing with word embeddings, unlike the character embeddings treated here, the problem of long-distance dependencies is likely to be problematic in both cases.

to be truly variable-length, without any unnecessary padding, however it would have likely slowed down the training process by a large factor.

2.5 Neural Network

The neural network itself was written in Keras, developed by Chollet [7], closely following its main developer's sequence-to-sequence Keras tutorial.[2] The input to the network had a fixed dimensionality due to the specific implementation used, but it represented variable length input, since the fixed dimensionality was enforced by adding semantically irrelevant padding, encoded by a predefined random word embedding, similarly as for unknown words. These vectors were fed to a Gated Recurrent Unit (GRU) layer, a common choice of gating mechanism for dealing with long-distance dependencies along with LSTM [8], but requiring less time to train [9]. The central aim of such layers is to improve retainment of information found significantly earlier in the input, for which regular recurrent neural networks are often insufficient due to the vanishing gradient problem, as discussed by Hochreiter [10]. The final state of the encoder was a fixed-length vector with a dimensionality of 256, which was then further passed to the decoder; another GRU layer, taking the annotations as the second input, and trained to output the same annotations "offset into the future" by one, so that it would learn not to predict the currently observed character but the subsequent one. Since the decoder can be interpreted as repeatedly classifying the most likely output characters given the characters until then, a softmax activation function was used, in order to transform the decoder output at each step into a list of probabilities. This decoding process continued until an explicitly encoded "end-of-sequence" character was input.

Inference was largely analogous to the training, with the difference that the input character to the decoder was not provided at each time step, but the decoder's prediction at each time step was "recycled" instead, and was used to make the next prediction, obtained by using argmax. This way the decoder only used the final hidden state of the encoder and a dummy input character as a starting point, and was responsible for generating all subsequent characters on its own, until a maximum allowed length was reached or the end-of-sequence token was output.

The optimizer that was settled on was RMSProp, using Keras' default learning rate of 0.001. The validation split was set at a fairly low 0.1, since training data was so valuable, and the model weights were updated upon observing every thirty-second input (batch size 32). Due to the fact that the model was essentially performing a classification task, the loss used was categorical cross-entropy.

[2] `https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html`

```
<p class="auto-style12" align="center">
Article<i>38.</i></p>

<p class="clan" style="line-height:1.0500002rem">Article 32
<span lang="sr-cyrl">*</span></p>

<p class="clan">"Official Gazette of the Republic of Serbia",
nr. 86 from 18. November 2011, 50 from 29. June 2018.</p>
<p class="clan">I. BASIC REGULATIONS</p>
<p class="clan">Article 1.</p>

<h4 class="clan">Article 1.</h4>
```

Figure 1: HTML inconsistencies. Normally, the "clan" class refers to an article, but as can be seen this is not consistently enforced, and there is no reliable nesting hierarchy. The last HTML snippet is how articles are represented in the revised HTML form. HTML text translated from Serbian into English and shortened for easier comprehension.

2.6 HTML Standardization

One of the signs that the update of legal texts was still not being performed automatically was the unsystematic, inconsistent and semantically opaque nature of the documents' source HTML. Examples of this can be found in Figure 1.

The extent of the variability in the HTML representations of nearly all relevant elements was so large, that parsing the source HTML would have been close to impossible. Therefore, the need arose to standardize the HTML to allow for an intuitive parsing process, which could only be reasonably achieved by ignoring the HTML altogether and reconstructing it from nothing but visual information. The HTML documents were transformed using the free "HTML 2 PDF" online service,[3] and then converted back from the resulting PDF form to HTML using a similar tool, namely "PDF to HTML".[4] The resulting HTML was crude and contained no semantic information, but did contain important visual cues such as the size and weight of fonts, margin sizes and font color. This enabled the development of a script that could combine these structural elements with some basic semantic content, gleanable from the text itself, to create a well-formed, consistent and semantically meaningful HTML representation of the legal documents, visually almost indistinguishable from the original ones. An example snippet of the resultant HTML can also be seen in Figure 1.

2.7 Annotation Parsing

The last section of the internship involved the interpretation of the annotation instructions (ideally retrieved as the output of the sequence-to-sequence model), their combination with

[3] https://html2pdf.com/
[4] https://www.pdftohtml.net/

the appropriate quoted passages from the natural language instructions, and ultimately the appropriate operation (deletion, replacement, insertion) in the legal document, following the conventions present in the manually performed edits. With a consistent HTML format, it became considerably easier to recurse into the right section of the HTML, where the desired change was to be performed. For most common instructions this was undertaken without issues. However, certain less common types of instructions were still very difficult to address, in particular appendices and sections, which were not reliably addressed in the HTML parsing step, as well as "metainstructions", which entailed a sophisticated cross-referencing mechanism that would require a detailed explanation to make sure that the structural rules of the legal documents were being abided by.

During the internship, no intrinsically ambiguous annotations were encountered, indicating that the performance of this section of the internship could have the potential to become a fast and reliable way of performing the relevant changes. One larger and as of yet unresolved issue was that only the instruction text (no markup) was extracted from the instruction files, which made it impossible to insert larger blocks of text, such as articles, that would conform to the HTML format. This could, however, be resolved by developing a script that would standardize the instruction HTML as well, for which plenty of code from the existing standardization script could be reused. The HTML of each article could then be individually extracted, and directly injected into the legal documents, since the similarities in structure between the two types of files could be "abused" to ensure a common HTML format.

3 EVALUATION

Overall, the model has yet to overcome several major hurdles, the extent of which is so great that metrics such as the BLEU score would probably not reveal any important information at this stage. The test set had a size of 398 sentence-annotation pairs, of which 57 represented deletions, 204 replacements, 123 insertions and 14 other instructions. The model accuracy, meaning the percentage of annotations that were replicated perfectly, was only 9%. This is even less impressive considering the fact that some of the test samples are likely to have been seen in a similar or identical form in the training data, which is corroborated by the fact that the accuracy dropped to only 6.8% when the formulaic "miscellaneous" instructions were disregarded, all 14 of which were reconstructed perfectly. These best results were achieved with a GRU encoder-decoder model described in Section 2.5, run for 120 epochs.

The average Levenshtein distance to the correct annotation paints an equally bleak picture, in that the predicted annotations had a distance of 3.2 characters to the gold annotations on average, where the mean predicted and gold annotation lengths were 5.9 and 6.4 characters, respectively. The one encouraging aspect, however, is that the accuracy in determining the relevant operation was a comparatively impressive 94.7%, all the more since this information was often found close to the beginning of a sentence, which is normally considered disadvantageous to such networks due to information being increasingly blurred the further

away it is found from the end of the sequence. The majority baseline would only be 51%, so it is clear that some learning did take place.

It is worth noting that the flaws were not necessarily tied to the model itself - a meager 865 training samples were used, whereas state-of-the-art machine translation systems often rely on millions of such samples, notably from the WMT datasets [2], [1]. Still, the creation of annotated data is far from trivial, and expends such a large amount of human effort that it might not be worthwhile attempting to improve the results in this way. One of the central aspects of the model that had a visible impact on the results proved to be the choice of character representations. The "semantic" character embeddings proved to be rather performant on detecting the semantics of the instructions themselves, and almost invariably led to the model correctly guessing the operation to be performed, as well as occasionally capturing some considerably more complex content from the instructions, such as the presence and structure of conjunctions, nested locations and valency of the instruction (how many quoted passages it implicitly required, for example evident in the instruction **YC5RR**, which calls for the replacement of a given word(s), presumably in quotation marks, with another word(s), also in quotation marks). The inconsistencies in delivering these results could indeed be largely attributed to an insufficient amount of training data, however one hurdle seems insurmountable - the distinction between numeric characters. The issue with "semantic" character embeddings is that the context-based vector representations of numeric characters end up hugely similar, leading the model to reproduce numeric characters not much better than at random (Figure 2).

This is a critical issue, since the correct operation performed on a wrong section would edit the law in an undesirable and unpredictable way. A way of countering this could be to incorporate attention with a copying mechanism [11], which would encourage the network to replicate items such as named entities, quoted sections, and most relevant in this use case, numbers, verbatim. This option, however, would greatly increase the model complexity, and it is questionable whether it would provide any positive impact on the translation itself, beyond finding the appropriate information to copy.

Coupled with Vasiljević's [12] observation that the structure of the natural language instructions is very predictable, as well as that their phrasing must follow certain predefined requirements, rule-based systems might in fact constitute a more promising approach than statistical methods in addressing the task at hand.

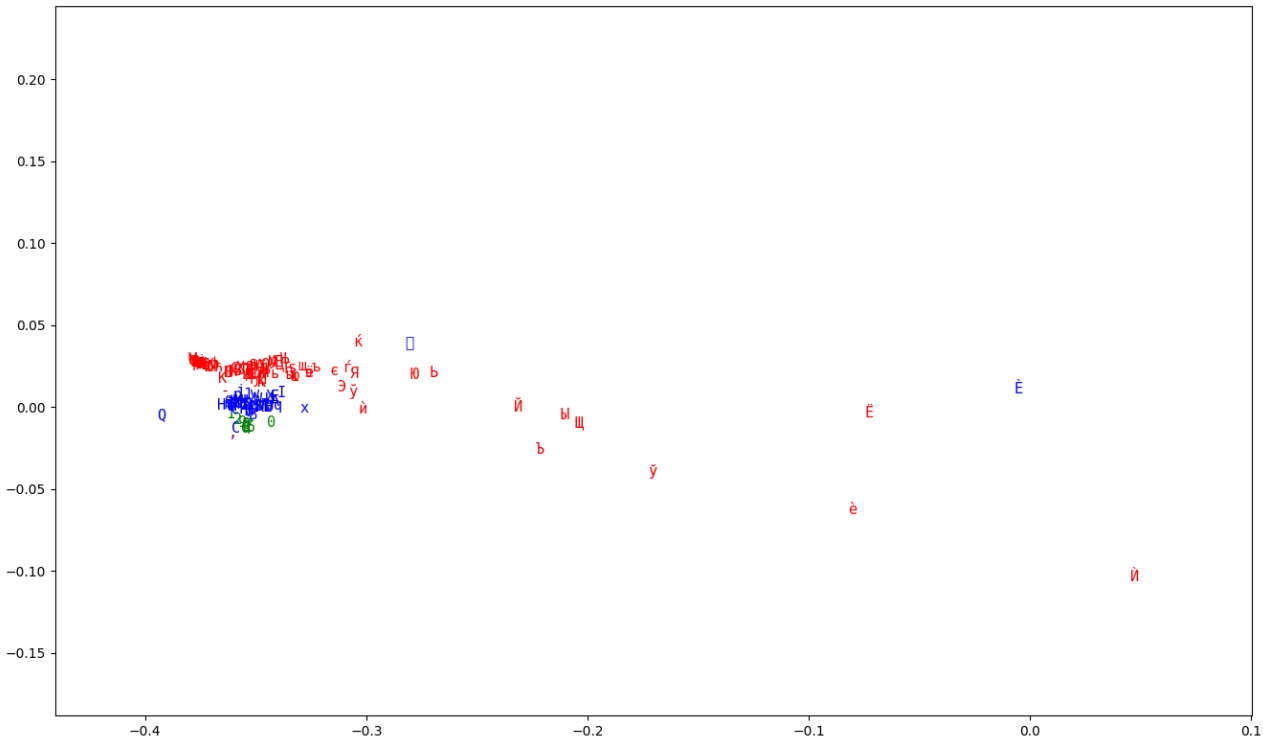

Figure 2: Enlarged view of the character embeddings reduced to two-dimensional space. Numeric characters are colored green, most punctuation is off the scale to the right.

4 REFERENCES

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *CoRR*, vol. abs/1409.3215, 2014.

[2] M. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," *CoRR*, vol. abs/1508.04025, 2015.

[3] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *CoRR*, vol. abs/1406.1078, 2014.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[5] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

[6] M. Woolf, "Pretrained Character Embeddings for Deep Learning and Automatic Text Generation," 2017.

[7] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[9] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," *CoRR*, vol. abs/1412.3555, 2014.

[10] S. Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 04 1998.

[11] J. Gu, Z. Lu, H. Li, and V. O. K. Li, "Incorporating Copying Mechanism in Sequence-to-Sequence Learning," *CoRR*, vol. abs/1603.06393, 2016.

[12] N. Vasiljević, *Automatic Processing of Legal Text in the Serbian Language*. PhD thesis, University of Belgrade, Faculty of Philology, 06 2015.

5 APPENDIX

Following is a set of instructions similar to the ones that the annotators were provided.

Every instruction document is divided into articles. Every article from the document should be described in the specified annotation file, whose structure would roughly follow the format:

Article 1
<annotation>
Article 2
<annotation>

Where the annotations are written in an invented, intermediate language. The intermediate language understands three basic operations. These are:

- **X** - deletion
- **Y** - replacement
- **Z** - insertion

Every description in the language begins with one of these three characters.

Every operation must be specified according to its exact location in the legal document. For instance, articles and paragraphs would have to be specified. Below is the "location lexicon", with English and Serbian translations.

- **C** - article (sr. član)
- **S** - paragraph (sr. stav)
- **N** - title (sr. naslov)
- **A** - indent (sr. alineja)
- **T** - point (sr. tačka)
- **PT** - subpoint (sr. podtačka)
- **G** - chapter (sr. poglavlje/glava)
- **O** - section (sr. odeljak)
- **PO** - subsection (sr. pododeljak)
- **@** - form (sr. obrazac)

The article/paragraph number would simply be marked by a number, for example *article 18* paragraph 2 would be **C18S2**.

If individual words are being added or replaced instead of entire articles/paragraphs, at the end of the annotation **R** or **RR** should be added, depending on whether the word *word* appears once or twice. If words are being inserted within parantheses, **(R)** should be written, and if behind a punctuation symbol *?*, then **?R**. It could happen that the instruction says *The word 'bla' is being replaced by the words 'bla blu bla' in parentheses*, in which case the end of the annotation would be **R(R)**. The same logic applies for when parantheses are being omitted, that is **(R)R**. When deletion is occurring, the word *word* should be contained in the instruction only once, so simply writing **R** would suffice.

All the above rules apply to numbers (**B**, sr. broj) and values (**I**, sr. iznos). Similarly, an appendix (sr. prilog) is to be marked with **P**, but it is of relevance to us only if an appendix number is specified (e.g. *Appendix II* would become **P2**). The other appendices can be **NNN** (unknown annotation) for now.

If the letter *đ* is encountered, the code for it is **$**.

If a list of numbers is encountered e.g. *2 to 5* instead of *2, 3, 4 and 5*, it is written in one line using the ^ symbol. *Delete articles 2 to 5* would therefore be annotated as **XC2^5**.

If multiple operations are noticed within one article (in the instruction document), it should be "pretended" that a new article has been encountered. For example, if article 3 says *word A is being replaced by word B*, and right afterwards *word C is being replaced by word D*, in the annotation file it would look as follows:

Article 3
Y-...
Article 3
Y-...

The ellipses, would, of course, be substituted with the exact instructions.

The last article in each set of instructions typically contains a message such as:

This instruction document becomes effective eight days after publication in the "Official Gazette of the Republic of Serbia"

It should be annotated as **008**, where 8 refers to the number of days after publication.

If an article does not have a recognizable structure, the annotation should be **NNN**. If something seems to be translatable but it is not clear how, it should be reported or made

more precise with the help of a comment, which is to be found right next to the annotation, after the # symbol.