DIKU NLP Course 2020/2021: Group Project

University of Copenhagen

October 30, 2020

Martin Birkmand
hpn761@alumni.ku.dk

Francesco Done'
qwg586@alumni.ku.dk
Exchange Student

Matthias Weiß
lqg753@alumni.ku.dk
Exchange Student

Abstract

Our aim with this paper is to define all the components that step-by-step led to the building of a multilingual question answering system. In particular, we will discuss the implemented code, and we will compare the results obtained with four languages (English, Arabic, Finnish and Korean). In chapter 1 (*Introduction to NLP*) we show how we made the data ready for the subsequent steps of implementation as well as an implementation of a feature-based binary classifier that outputs the prediction for a given question and document (YES or NO). In chapter 2 (*Representation Learning*) an extension of the classifier described in the previous chapter, in which features based on continuous vector representations of words are used, is presented. Chapter 3 (*Language Modelling and RNNs*) describes how we extract a sentence representation from an RNN language model. In chapter 4 (*Sequence Labelling with CRFs*) the implementation of a sequence labeller, which predicts which parts of a paragraph are likely to answer spans given the question, is described. Finally, in Chapter 5 (*Machine Translation*), we explain how to translate all of the English questions and their corresponding documents to two of the three non-English languages (Arabic and Finnish).

1 Introduction to NLP

The goal of the first task was to get familiar with the dataset and the preparation of that dataset for later tasks. The dataset used was a subset of the *MinSpan* dataset, more specifically yes/no questions in four languages (English, Arabic, Finnish and Korean), which in turn is part of the bigger TyDi QA dataset [1]. To get familiar with the data, we started by tokenizing the relevant questions using NLTK [2] and subsequently analyzing the most common first tokens of the questions. While Korean did not have any words occurring significantly more often than others, the word "onko" (= "is") in Finnish appeared over 700 times; other prominent first tokens are "voiko" (= "can") and "oliko" (= "it was", but we think that it preferably should be "was it"). Arabic was the most one-sided result, where almost all of the over 1300 questions started with a token that translates to "do you". English had the least amount of different first tokens besides Arabic, containing all apparent words like "can", "did", "do", and the one occurring most often, "is". To translate these words to English, we used Google Translate [1].

The next goal was to create a binary classifier, where we first chose a naive Bayes classifier, which we later replaced by a logistic regression model. Features were the number of occurrences of words in the concatenation of question and document text. The F1 scores, evaluated on the dev set of the dataset as mentioned earlier, for the languages were as follows:

Language	F1 score
English	0.5038
Arabic	0.8139
Finnish	0.6412
Korean	0.9043

Table 1: F1 scores of dev set, for different languages

While some of these scores already seem relatively high, they are not meaningful due to various reasons such as the size of the dataset (minimal), its skew (far more YES questions than NO questions) and lack of sophisticated classification techniques. It should also be noted that we chose the F1 score as a metric to compare our results, over accuracy for example, due to the skew of the answers of the questions.

[1] https://translate.google.com/

2 Representation Learning

We then continued to improve our classifier by introducing continuous vector representations of our words (and later on whole sentences). We made use of the popular Gensim [3] library to train Word2Vec [4] models based on the questions and documents in our dataset. We opted for a CBOW model instead of the skip-gram counterpart because it gave us higher F1 scores in the early phases (although we only trained our classifiers on a subset containing only English questions at that time). Converting the words on their own to vector representations before feeding them into the classifier resulted in the following, not significantly different, F1 scores:

Language	F1 score
English	0.4942
Arabic	0.8384
Finnish	0.6738
Korean	0.9043

Table 2: F1 scores of vector representations of the words, for different languages

We subsequently transformed the whole concatenation of question and document into a single vector by max-pooling the individual vectors, which gave us the following F1 scores: English 0.5444, Arabic 0.8125, Finnish 0.6073 and Korean 0.9043. These are lower than for the individual vector representations, which we think has to do with the way that we pooled the word representations. While this gave us higher results than, for example, averaging out the vectors, we still could not outperform the classifier using individual word representations. While we think that we could have improved our results here, e.g. by trying out various types of features, we decided to focus our time efforts on the remaining, more advanced tasks.

3 Language Modelling and RNNs

For this task, we have to build a language model using the RNN layout as given in *lab3* [5]. However to increase on the performance we have added packing of the sequences along more advanced tokenizers.

3.1 Tokenization

We have expanded upon the tokenizer given in *lab3*, from hereon referred to as `lab3_tokenizer`. Given a sentence like: **`"Denmark is the state capital of Norway.[1]"`**, the `lab3_tokenizer` just splits each word, hence we have the resulting tokens:

`['Denmark', 'is'...'Norway.[1]']`

This will result in an unnecessary large vocabulary since we have that

`'Norway.[1]'` `!=` **`'Norway'`**

To overcome this we have used the `nltk.word_tokenizer` method to tokenize. In the above case for this method we have the tokens

`['Norway', '[', '1', ']']`

We have also build a tokenizer using regular expressions. With this tokenizer, we only pick out actual words. Hence only `Norway` would be a token in the above case. The only distinction we make here is for genitive/short-form versus plural. Hence we have that `Casper's` and `Caspers` (for referring to several people named Casper) are distinct. This regex tokenizer has been extended further upon by lowercasing all tokens. Each extension shrinkens the vocabulary.

3.2 Sequence Packing

We have added a packing mechanism [6] to the model, to avoid making steps over padding tokens.

3.3 Training

The language model was trained with the following parameters: `size_total=1000` (that is the number of records used for training, the maximum is 9211, but it seems to take too much time for doing it correctly with the full set of records inside the dataset), `n_epochs=5` and `batch_size=35`. The result is shown in the following chart:

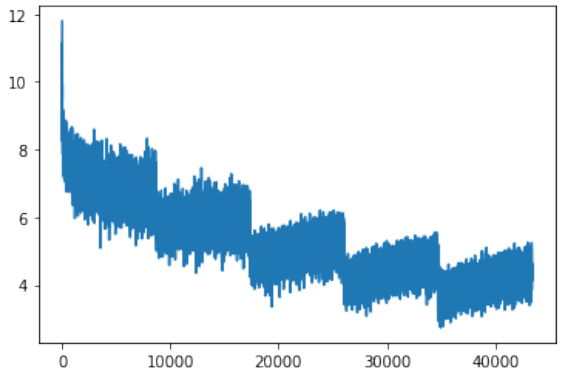

Figure 1: Losses for the training of the English version RNN with `size_total=1000`, `n_epochs = 5` and `batch_size = 35`

The chart demonstrates that the loss is slightly decreasing over time, but nothing significant.

3.4 Sentence Generator

The model is now ready to generate sentences. During the tests, it used as input the sentence *"**Art Deco**"*, and it generated the following sentence: *"**Art Deco architecture in the late 1930s and early 1960s the early 1960s and 1970s began in the 1930s with the increasing number of years before the age of**"*. The perplexity concerning our model, calculated with *"I want to buy some potatoes from the airport."* and *"gibberish ? . something something is"* resulted respectively 10.9639 and 11.4087.

4 Sequence Labelling with CRFs

We want to build an RNN for the task of assigning labels to each word of a given text. Here we use the data set `tydiqa-goldp-v1.1-train.json` consisting of 11 typologically diverse languages with 204K question-answer pairs. The labelling approach is IOB (Inside, Outside rsp. Beginning), and we label according to spans. Each span is set according to where the answer is found within a given document. We train two RNN's, one using a bare neural network, and one using a neural network extended with a CRF in the last layer. Lastly, we compare the two different network approaches.

4.1 Labeling

The dataset is a `.json`-structure consisting of (among other fields) a document text, an answer start index and an answer. We loop over each text and, we tokenize each word using `nltk.tokenizer`. For a sequence of characters like **`"America.[1]"`** this results in the tokens `['America', '.', '[', '1', ']']`, which is what we want instead of one token containing all characters. In this case, we would have two different tokens for **`"America.[1]"`** and **`"America."`**. We add tags according to whether a given token is at the beginning (B) of the answer text, inside the answer span (I) or outside the answer span (O). For this, we use the `nltk.span_tokenize` method: in this way, we obtain the index of each token to easily check if the token is inside or outside the index span.
We split the training set into three sets: a *training* set, a *test* set and a *dev* set. The last one has been handed out and, using it; we could extend the training set. The test set ($T'S$) is the last 10% of the original training set (OTS) which results in a new training set that is 90% of the original one. Finally, the dev set (DS) is the 10% of this new training set, so the final training set (TS) is 90% of the previous one.

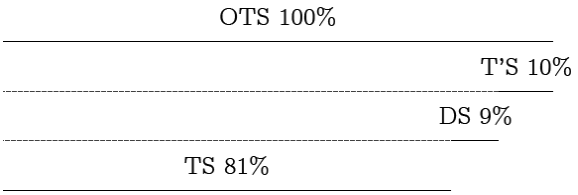

Figure 2: Size of different sets

4.2 Pretrained word embeddings

The model inputs are word embeddings: instead of adding a layer in order to construct the embeddings from the words, we use the pre-trained word embedding `Fasttext`. This is a list of words each with an embedding vector attached to it; this simplifies the process a lot since we do not have to deal with constructing the embeddings.
For the model, we need to construct a vocabulary, this has been done using the vocabulary from the whole training set along with the 20000 most common tokens from the pre-trained vectors, done so in order to accommodate for out of vocabulary words when using the model after training.

4.3 Building an RNN model

The model build here is essentially the same as the one build in *lab2* [7] of the labs in this course. However, instead of dealing with a single label for each answer span, we add a label for each word instead. In order to make cross-entropy loss on this label batch, we need to flatten both `logits` and `labels`. This is so since the cross entropy cannot deal with more than one dimension for the target argument. Then we evaluate the model using the accuracy: it can be done by running different times, tuning the hyper-parameters when training. With `n_epochs = 10` and `batch_size = 8` we have a final accuracy of `0.965` that is plotted in the following chart:

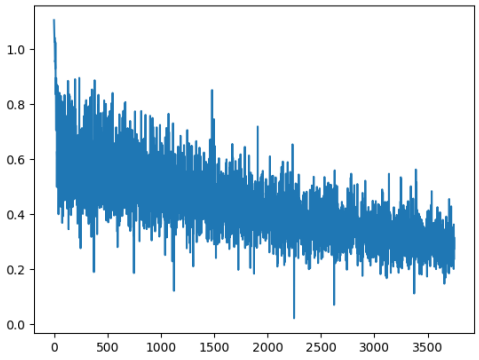

Figure 3: Losses for the training of the English version of seq-label RNN with `n_epochs = 10` and `batch_size = 8`

We can try training with `n_epochs = 14`, so we get an accuracy of `0.963` and a loss curve as seen in the following figure:

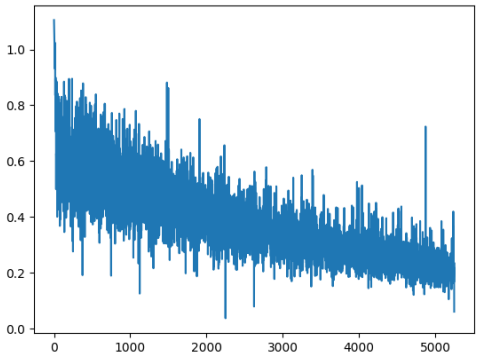

Figure 4: Losses for the training of the English version of seq-label RNN with `n_epochs = 14` and `batch_size = 8`

There have been some problems with accuracy for this task: we pick the best model with regards to accuracy. Moreover, it might be so that the best accuracy is not assigned to the best model. Alternatively, it can just be that the model with the highest accuracy is not the one with the lowest loss. However, we can do one more run using `n_epochs = 20`, the final score for the three-run is as follows:

Epoch	Score
epoch[10].loss	0.2957
epoch[10].acc	**0.965**
epoch[14].loss	0.2215
epoch[14].acc	0.963
epoch[20].loss	**0.1511**
epoch[20].acc	0.962

Table 3: F1 scores of vector representations of the words, for different languages

As can be seen: higher epoch count results in a lower loss (better loss), whereas the accuracy seems to get lower (worse), but from epoch[10] to epoch[20] the loss has been halved while accuracy is decreased a little bit, but nothing significant. Here below are reported other losses for the training of the remaining languages, each one has been set with `n_epochs = 10` and `batch_size = 8`:

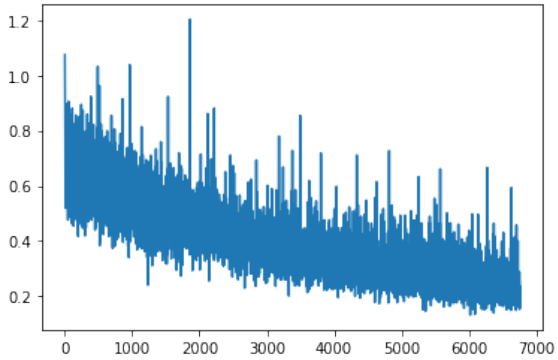

Figure 5: Losses for the training of the Finnish version of seq-label RNN with `n_epochs = 10` and `batch_size = 8`

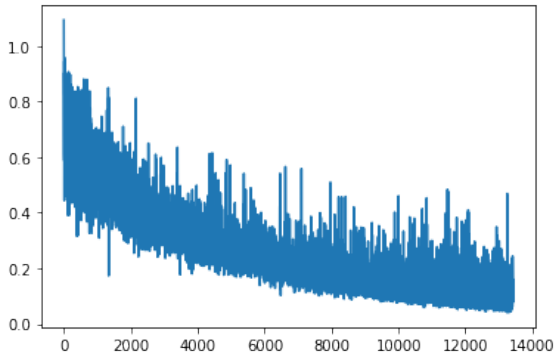

Figure 6: Losses for the training of the Arabic version of seq-label RNN with `n_epochs = 10` and `batch_size = 8`

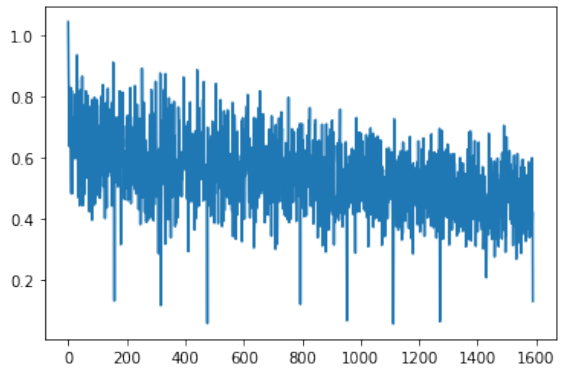

Figure 7: Losses for the training of the Korean version of seq-label RNN with `n_epochs = 10` and `batch_size = 8`

4.4 Building an RNN model with CRF

We can add a CRF layer after output on the model described in 4.3. This helps preserve the structure of the assigned tags. For example, in the non-CRF approach, nothing ensures that the order of tags makes sense. We could have an `O`-tag within a span [8]. For the CRF layer, we use the `torch-crf` library. Using the `CRF` function from this class we can obtain the $log-likelihood$ of the output, using the negative version for losses. We use F1 scores here instead of accuracy. F1 makes more emphasis on the outliers.

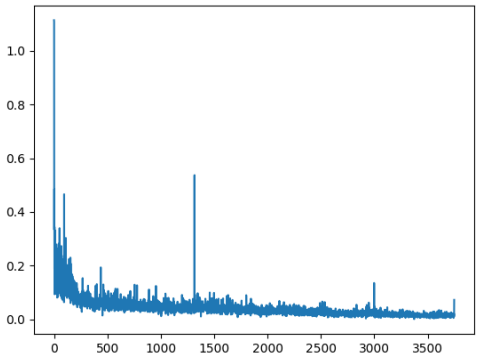

Figure 8: Losses for the training of the English version of seq-label RNN-CRF with `n_epochs = 10` and `batch_size = 8`

Though we use different measure functions, it makes sense that adding the CRF layer results in a better model since we somehow ensure the order of tags as described above. We have the following results when training with two different epoch sizes:

Epoch	Score
epoch[10].loss	0.014
epoch[10].acc	**0.982**
epoch[14].loss	**0.004**
epoch[14].acc	0.9824

Table 4: F1 scores of vector representations of the words, for different languages

Here below are reported other losses for the training of the remaining languages, each one has been set with `n_epochs = 10` and `batch_size = 8`:

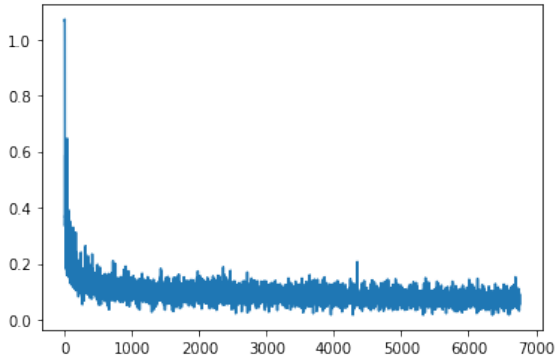

Figure 9: Losses for the training of the Finnish version of seq-label RNN-CRF with `n_epochs = 10` and `batch_size = 8`

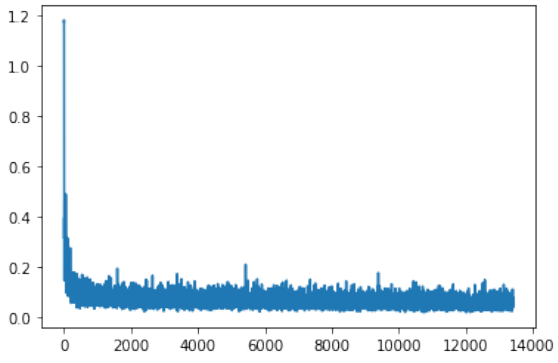

Figure 10: Losses for the training of the Arabic version of seq-label RNN-CRF with `n_epochs = 10` and `batch_size = 8`

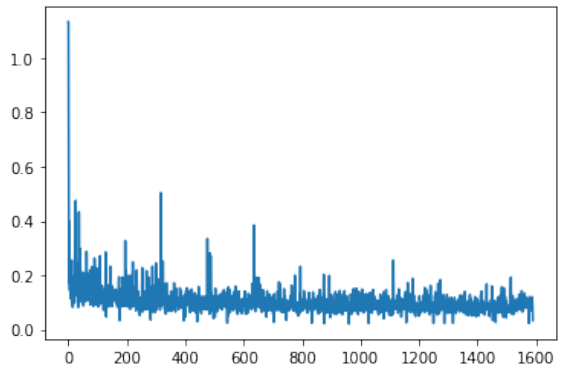

Figure 11: Losses for the training of the Korean version of seq-label RNN-CRF with `n_epochs = 10` and `batch_size = 8`

4.5 Drawing samples

We can draw a set of samples from the model using the test set. We get `[['Frank', 'O'], ['M.', 'O'], ... ]`, so does the sample make any sense? Not much. It could have something to do with training the model. However, it could also have to do with the idea of what we are doing. There is not much linguistic structure to exploit when predicting where in a text an answer to some question might be. It could have to do with statements, like

"London is the capital of England"

However, any of these statements (a sentence with this grammatical structure) in a document could be an answer span. This might be a problem with the whole idea here.
Say the idea is to use the structure of each answer as a way to predict the answer. In this case, we would have more data if we did parsing on the documents, obtained the grammatical structure of each answer, picked out the text bits with this structure in the test set and labelled this as a possible answer. That is picking some other labels than IOB on the span of answers.

5 Machine Translation

The next step was to translate questions from the *MinSpan* dataset, which we have already used in earlier tasks. For our solution we made use of pre-trained machine translation, in particular, the *huggingface transformers* [9]. To be exact, we used the MarianMT [10] models, which were published by the Language Technology Research Group at the University of Helsinki (a list of all available models can be accessed here [2]). We translate each of the training questions and documents from English to Arabic and Finnish (unfortunately there are not pre-trained models for the translation of English to Korean). Since integrating these models into an existing codebase is only a matter of a few lines of code, we were up and running relatively fast. We ran into some problems at first when we tried to translate entire documents, which could easily be solved by using sentence tokenization, in our case using NLTK [2], and subsequently translating each sentence on its own. Since none of our team members speaks Finnish or Arabic, we had to use online translation tools to rate our translated questions and documents. An example of a question in the dataset would be:

"Is Creole a pidgin of French?"

The pre-trained model translated this to the following Finnish sentences (:

"Onko Creole ranskankielinen pidgin?"

As a comparison, Google's translator [3] would translate the given English sentence as follows:

"Onko kreoli pidgin ranskaa?"

If we translate both of these results back to English, we get the following sentence for the translation generated by the pre-trained model:

"Is Creole a French pidgin?"

The above sentence seems like a better translation than what we get after re-translating the translation from Google's translator back to English, which looks like this:

"Is Creole Pidgin French?"

While we saw similar results with the translations for Arabic, it is noteworthy that it is much more obvious that some words are not translated by the model when you look at the Arabic translations, since the non-translated words appear in the 26-letter alphabet that the English language uses rather than the expected Arabic alphabet.

Contributions

During the entire project period, we tried to split up the assignments in a way that each of the team members had a specific task they could work on.

[2] https://huggingface.co/Helsinki-NLP
[3] https://translate.google.com/

Since we encountered problems due to misunderstandings or malformed code we had to look for a new solution, which was to work together through zoom and occasional meetings at the department, focusing on all the problems that we had together as a group.

Conclusions

While we had promising results in the first couple of assignments, we unfortunately did not manage to finish the later tasks. This was due to lots of issues that we had in assignment 3, especially in the associated classifier. Since the last assignments are based on this classifier, we tried our best to implement everything else and see if we could fix our problems with assignment 3 in the meantime.

References

[1] Jonathan H. Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. Tydi qa: A benchmark for information-seeking question answering in typologically diverse languages. *Transactions of the Association for Computational Linguistics*, 2020.

[2] Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[3] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[4] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[5] Laboratory - third lesson. lab_3.ipynb.

[6] Minimal tutorial on packing and unpacking sequences in pytorch. link.

[7] Laboratory - second lesson. lab_2.ipynb.

[8] Daniel Jurafsky and James H. Martin. Speech and language processing. In *Speech and Language Processing*.

[9] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[10] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia, July 2018. Association for Computational Linguistics.