READ

RECOGNITION & ENRICHMENT OF ARCHIVAL DOCUMENTS

D7.10
Language models

Maximilian Bryan, Soeren Laube, Nathanael Philipp
ASV

Distribution: http://read.transkribus.eu/

READ
H2020 Project 674943

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 674943

Project ref no.	H2020 674943
Project acronym	READ
Project full title	Recognition and Enrichment of Archival Documents
Instrument	H2020-EINFRA-2015-1
Thematic priority	EINFRA-9-2015 - e-Infrastructures for virtual research environments (VRE)
Start date/duration	01 January 2016 / 42 Months

Distribution	Public
Contract. date of delivery	31.12.2016
Actual date of delivery	28.12.2016
Date of last update	21.12.2016
Deliverable number	D7.10
Deliverable title	Language models
Type	report
Status & version	in process
Contributing WP(s)	WP7
Responsible beneficiary	ASV
Other contributors	
Internal reviewers	URO, UPVLC
Author(s)	Maximilian Bryan, Soeren Laube, Nathanael Philipp
EC project officer	Martin MAJEK
Keywords	ARPA, Neural Networks

Contents

1 Executive Summary

When given scanned documents, there are tools which segment the documents into lines which then are given to an HTR engine. That engine transforms the graphical input to a sequence of tokens. These tokens are characters or words respectively. With each token, the HTR returns a confidence (or probability) for every token or character in the dictionary. The language models receives a sequence of confidences along with the character sequence and adjust the character sequence according to the learned patterns in the language, that are unknown to the HTR. The goal of the language model is to give the most likely i.e. most probable sequence.

In the current state of this task, a wrapper has been implemented that makes models, trained using Python tools, usable. With this wrapper, forecasting tokens is possible using a generic interface. The interface is implemented using the mentioned wrapper[1] on the one hand and previously existing ARPA files on the other hand. An ARPA file contains a n-gram language model. In the future, models for recognizing abbreviations will be trained.

2 Context

Generally, a language model contains knowledge about a language, mostly patterns about sequentially appearing words. These patterns are learned from training data. The easiest way is to define a neighborhood and count the occurring token pairs. These statistics can then be used to adjust a recognized text to a more likely text according to learned patterns in the language. In the current state of the project, these basic types of language models are called n-grams and are usually stored in files using the *ARPA* format.

The advantage of the statistical approach is that it is easy to implement. The downside on the one hand is that every possible combination of tokens, together with their probability, have to be stored. On the other hand, it cannot handle patterns that it has not been seen in the training data. If the HTR has recognized *The dog was* and the training data did not contain that example but only *The dog has*, it will correct the grammatically and semantically plausible recognized text to another text, because it does not know about the other variant.

To improve the accuracy of such n-gram models one can define so called backoff values. These are used to enable combinations of n-grams which large n that were not seen in the training. To facilitated this the entries in the 1-grams have additional backoff values. With these values one can calculate probabilities for sequences of length n with OOV-combinations.

The mentioned problem of out of vocabulary sequences shall be tackled by neural networks. Neural networks are a powerful machine learning technique, which has received a lot of attention in the last year for every kind of problems. Here, the neural network can learn transition probabilities from the training data [1]. Since it also implicitly learns semantic relations between words, it can give reasonable response to input sequences

[1] `https://github.com/Transkribus/NeuralNetWrapper`

it has not seen before. Also, not every possible combination has to be stored in a file, the network can be asked for a sequence's probability. The downside is that the implementation of a neural network is way more complicated than looking for values in a file. In the current state of the task, a wrapper for trained neural networks is implemented and can be used for the given problem. In the future, more complex models for different use cases can be included into the HTR process.

3 Implementation

The ASV group is experienced in training neural networks for various use cases. The group is allowed to use hardware resources of TU Dresden which speeds up neural network training significantly. The downside is that the current training relies on specific frameworks and the programming language *Python*, whereas the current Transkribus source are implemented in *Java*. Thus, the trained networks cannot be used directly in Transkribus but there has to be a wrapper. That wrapper has to load the trained model and reimplement layer functionality. While neural network frameworks are all very complex, that complexity lies in the training process. The plain passing of values through the network on the other hand is quite simple.

Following commonly layers of a neural network have been implemented:

- Type of recurrent networks:
  - GRU [3]
  - MGU
- Activation functions:
  - Sigmoid
  - Hard Sigmoid
  - Tanh
  - Softmax
  - LeakyReLU [4]
  - Word embeddings [2]

4 Existing functionality

After having implemented basic neural network functionality, an interface[2] for basic language model tasks has been implemented:

```
public interface ILanguageModel
{
```

[2] https://github.com/Transkribus/TranskribusLanguageResources/blob/master/src/main/java/eu/transkribus/languageresources/languagemodels/ILanguageModel.java

```
    public Map<String, Double> getProbabilitiesForNextToken(List<String>
        sequence) throws UnsupportedSequenceException;
}
```

Listing 1: Interface definition for forecasting the next most probable toke

The interface offers a method which receives a list of tokens and returns a map. The map contains a probability, represented by a double, for every token in the dictionary. The double value represents the probability of a token appearing given a sequence. The interface is implemented by an ARPA and by a neural language model.

5 Expand Abbreviations Experiments

As historical texts often abbreviate a lot of words, there exists interest in expanding these abbreviated words to their original form. For some abbreviations, this is easy, but, in many cases, the expansions of the abbreviation depends on the context of the word. This is even more the case for languages like German or Dutch, where the last letters of words often contain grammatical information. Without context, it is hardly possible to give the correct expansion for an abbreviation.

5.1 Data preparation

Thus, the ASV group has started to train models that are given a text with abbreviated words. Since they are not eplicitly annotated, the model first has to learn to recognize abbreviated words. Then it has to learn which expansion corresponds to the abbreviation. The current experiments have been done with the dataset *Itinerianova* which contains 27,000 XML documents and about 430,000 lines. The unparsed lines look like this:

```
It(em) beat(ri)x shoeven van tyeldonc es comen in jegenw(or)dich(eit) d(er)
    scepen(en) va(n) lov(en)
en(de) heeft gekent en(de) ghelijt dat sij hoirs danx en(de) hoirs wille(n)
    ghegaen
```

Listing 2: Examplary lines from the Itinerianova data.

The brackets indicate abbreviations: The token *It(em)* has been abbreviated to *It*, its expanded form is *Item*. As one can easily see, every line contains lots of abbreviations. For training, every line has been parsed into two formats: one with all brackets and its contents removed, one with just the brackets removed. That results in one format with everything abbreviated and one form with everything expanded. The abbreviated form looks like this:

```
It beatx shoeven van tyeldonc es comen in jegenwdich d scepen va lov
en heeft gekent en ghelijt dat sij hoirs danx en hoirs wille ghegaen
```

Listing 3: Examplary training lines from the Itinerianova data.

The data then has been splitted into training and test data, whereas randomly whole XML documents have been chosen as training data with the probability 0.9 and as test data otherwise. This means that 90% of the documents are training data and 10% of the documents are test data.

After seperating the data, a dictionary has been created. The dictionary has been filled with words from the two kind of training files, which are the file with the abbreviated lines and the file with the expanded lines. Thus, the dictionary contains types that have been abbreviated and the expanded forms, but also all other words. If there are words in the test data that are not found in the training data, that word cannot be recognized. This should cover the fact that, if a trained model is used on unkown data, not all words are in the dictionary of known words. Also, all the words are stored in lowercase to reduce the number of variations.

Sometimes, whole words are being left out, but this is only the case for the word *et*. This results in the not-abbreviated sequences being longer than the abbreviated sequences or having more words respectively.

5.2 Experiments

In the current state of the project, to different neural network architectures have been implemented and trained.

5.3 Simple network structure

First, a very simple neural network has been trained. It is given a random token from the training lines together with the left and right neighbour. The network has to learn three things:

- whether a word has been abbreviated
- the expansion of the word
- whether after the focused word the word *et* appears in the non-abbreviated sequence

Technically, the network has three outputs. The first output has a single value between 0 and 1 which represents the network's confidence whether the word has been abbreviated. The second output contains as many values as there are words in the dictionary. The activation for that output is softmax, thus all the values are between 0 and 1 as well and if being summed up, the sum equals 1. This means that the output can be treated as a probability, i.e. every value represents the probability of the corresponding type being the expansion of the abbreviated form. The third output is a binary value representing the network's confidence that the word *et* has been left out in the abbreviated sequence.

5.4 Bidirectional recurrent network structure

Secondly, a more complex neural network architecture has been implemented. It does not get static input like the previous network but gets whole sequences, in our case whole

lines as input. The network then iterates over the given sequence twice, forwards and backwards. The results per timestamp are then concatenated into another sequence. This gives the advantage that the concatenated sequence at every timestep contains information about the whole sequence since it gets information from the forward and from the backward network.

The output is the same as in the simple architecture, but now the network has to output a sequence for every one of the three outputs. The values then represent the same three things as the simple network's output, but now for every timestep of the input sequence.

5.5 Results

When testing the network, we use the previously created test data. A line is being put into the network and it then outputs, as trained, for every token wether that token is an abbreviation and if yes, what is that token's expansion. When evaluating that process, we are interested in two things:

- How could are abbreviations found?
- How good are they being expanded?

For the first question, precision/recall are good measurements. Here, precision indicates if the network says a token is an abbreviation, how often the network is correct. Recall indicates how many of the existing abbreviations have been found. When measuring the correct expansion, we just measure the accuracy, i.e. a percentage value of how often the correct expansion has been chosen.

Number of lines	43864
Number of words	464980
Number of abbreviations	152815
Average number of words per line	10.6
Average number of abbreviations per line	3.04

Table 1: Basic test data statistics

The results in table 2 show that the more complex bidirectional recurrent architecture is way better with expanding abbreviations that the simple structure, though both are not perfect as indicated by the precision and recall values. It can be assumed that the recurrent architecture makes good use of the bigger input information and then is better at choosing the grammatically correct expansion form.

```
Input line:    elx jaers daen bynne ome en voe xii molevate rox
Expected line: elx jaers daeren bynnen omme ende voer xii molevaten rox
Created line:  elx jaers daeren bynnen omme ende voer xii molevate rox
```

Listing 4: Examplary test results from Itineria expansion experiments

	simple architecture	bidirectional recurrent architecture
Number of correctly found abbreviations (true positive)	65198	90361
Number of wrongly assumed abbreviations (false positive)	87620	70109
Number of not found abbreviations (false negative)	87619	62456
Precision	39.62%	42.67%
Recall	56.31%	59.13%
Percantage of correctly expanded abbreviations	56.63%	85.85%

Table 2: Experiment results

When looking at the data, it is interesing to see the cases where both networks have problems. Listing 4 shows three lines. The first line is the abbreviated input line, that is given into the network. The second line shows the correct line (ground truth). The third line is the line created by the network. One can see, that that line contains five abbreviations: *daen*, *bynne*, *ome*, *en*, *voe* and *molevate*. The first four have been recognized as abbreviations and were expanded correctly. *molevate* was not recognized as an abbreviation was thus was not expanded. This example was chosen to show that the abbreviated and expanded words are often very similar. Also, many words, like *molevate*, also exist in their abbreviated form, which makes it even harder to learn the correct form.

5.6 Outlook

The results show that bringing more context information into the network, the network is able expand more accurately. We believe that combining the recurrent architecture with convolutional networks[3] will enhance the use of context information. Also, since many words differ only slightly in their spelling, we believe it would make sense to also give character information into the network.

An open question is how to handle the word *et*. In the given data set, it is the only word being left out completly. Thus, the input sequences with the abbreviated words and the target sequences with the expanded words sometimes differ in their length. Currently, this problem is solved by using a dedicated output which determines whether the target sequence contains the words *et* that was not there in the input sequence. For the given data set, this is a plausible solution. But this approach makes the network unusable for other data sets.

Thus, in the next phase of the project, we want to create a more complex network

[3]E.g.: `http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/`

structure that makes use of convolutional layers and character information and does not contain a special solution for the word *et*.

[1] Bengio, Y.: A Neural Probabilistic Language Model, 2003
[2] Gal, Y.: A Theoretically Grounded Application of Dropout in Recurrent Neural Networks, 2016
[3] Jozefowicz, R.: An Empirical Exploration of Recurrent Network Architectures, 2014
[4] Xu, B.: Empirical Evaluation of Recti ed Activations in Convolution Network, 2015