**Sequence Tagging for Variation-Rich Languages Using Deep Learning**

**Abstract**

In this paper, we describe a novel approach to sequence tagging for languages that are rich in (e.g. orthographic) variation. We focus on lemmatization and part-of-speech tagging, two basic steps in many processing pipelines in the Digital Humanities. While these tasks have long been considered solved for modern languages such as English, there exist many historical or minority languages, for which these problems are notoriously hard to solve, due to a lack of resources and unstable orthography. Our approach is based on recent advances in the field of “deep” representation learning, where neural networks have led to dramatic increase in performance across several domains. Our system combines two approaches: on the one hand, we apply temporal convolutions to model the orthography of input words at the character level; secondly, we use distributional “embeddings” to represent the lexical context surrounding the input words. We demonstrate how this system reaches state of the art performance on a number of representative data sets, even without corpus-specific parameter tuning.

**Introduction**

Sequence tagging is a central problem in Computational Linguistics (Natural Language Processing or NLP). It includes well-known tasks such as part-of-speech tagging or lemmatization, which are a basic, yet foundational stage in many text processing pipelines in the Digital Humanities. Lemmatization, for instance, offers a form of normalization which helps to more efficiently process historic text corpora such as newspaper databases, ranging from plain searching to more advanced text categorization tasks in e.g. stylometry. Likewise, PoS-tagging is a crucial step in automated Named Entity Recognition, which is of increasing importance in many Digital Humanities projects. For present-day majority languages, such as English, tasks such as PoS-tagging and lemmatization are somewhat considered “solved”, as in recent years studies have only been able to report small improvements upon benchmark corpora such as the Wall Street Journal corpus. However, there exist many languages for which these tasks are much more difficult.

Firstly, there are the historic precursors of modern languages, Most historic stages of present-day languages are characterized by the absence of a generally accepted standard language and orthography. In pre-modern times, language users typically enjoyed a relatively larger free freedom with respect to spelling and spacing, since there only existed local (or at most regional) orthographical conventions (Piotrowski, 2012). Thus, the same word could appear in multiple, roughly equivalent spellings – even within the same text – which obviously presents major challenges to computational applications when it comes to word identification. The recent, dramatic growth in the availability of historic text corpora in electronic form has only increased users’ needs to be able to efficiently deal with this kind of data in computer applications, e.g. searching through genealogic databases or newspaper archives (Ernst-Gerlach *et al.*, 2006). Most systems developed for tagging present-day language are not equipped to deal with large amounts of orthographic variation in the input and struggle to reliably identify unknown words in unseen texts. Naturally, this results in a considerable percolation of errors to higher-end layers in a processing pipeline (e.g. syntactic parsers).

Secondly, there exist many present-day minority languages, for which far fewer resources available, in the form of e.g. annotated corpora. Perhaps unsurprisingly, these are typically also languages that are characterized by much more complex morphology and higher level of inflection than e.g. English. Apart from obvious examples from the Finno-Ugric language family, one can also point to many Sub-Saharan languages in Africa, where both the development of (digital) literacy and the application of NLP are still in its infancy. Both historic and minority languages form the focus of many Digital Humanities projects, targeting the better exploitation or preservation of (specific cultural artifacts in) these languages. For these too, however, many of the available sequence taggers are not sufficiently equipped to deal with the advanced sub-word level variation, nor to optimally exploit the few resources that are available.

Needless to say, the availability of efficient sequence taggers for historic and/or minority languages is a therefore a desideratum across various fields in computational humanities. In this paper, we describe a novel, language-independent approach to sequence tagging for variation-rich languages. To this end, we apply a set of techniques from the field of “Deep” Representation learning, a family of algorithms from Machine Learning based on neural networks, which has recently led to a dramatic increase in performance across various domains (in computer science, and elsewhere). In this paper, we will first survey some of the most relevant related research, before we go one to briefly introduce the field of deep learning. We then go on to describe the architecture of our sequence tagger and discuss the data sets on which we will evaluate our system. After presenting the results, we offer an interpretative exploration of our trained models. We conclude with a set of pointers for future research, stressing the considerable potential of representation learning for the Humanities at large.

**Research into historical spelling variants has enjoyed a considerable increase in academic interest in recent years. An important impetus also comes from the considerable number of papers in the Computational Linguistics community on the normalisation of Internet-Mediated Communication (e.g. so-called ‘Twitterese’) of which Chrupała (2014) and Ljubešić *et al.* (2014) are representative examples. Finally, valuable unsupervised approaches have explored by e.g. Mitankin *et al*. (2014).**

**Related Research**

Morphological analysis and part-of-speech tagging are a popular research domain in computational linguistics, with recent studies especially focusing on (low-resource) languages with a rich morphology or inflectional system. An important boost in NLP research for such languages has been inspired by the increased relevance of computer-mediated communication (CMC, Crystal, 2001), including research into the normalization of Instant Messaging or posts on online blogging platforms, such as the popular micro-blogging service Twitter. A common solution to the problem of spelling variants in CMC is to apply some form of spelling normalization before subsequent processing. (Similar research has been reported in the field of post-OCR error correction (Reynaert).) Spelling normalization essentially involves replacing a non-canonical surface token by a standard form (e.g. Beaufort *et al.*, 2010, Chrupała, 2014), as would be done with spelling correctors in modern word processing applications. In social media, a variety of approaches have suggested, including e.g. transliteration approaches borrowed from Machine Translation (e.g. Ljubešić *et al.*, 2014).

The field of Natural Language Processing for Historical Languages has recently been surveyed in a dedicated monograph (Piotrowski, 2012). The problem of orthographical variation naturally plays a major role in this overview. Recent representative contributions in this field include Hendrickx et al. (2011) and Reynaert et al. (2012) for historical Portugese, Scherrer et al. (2013) for historical Slovene, Bollman (2012) for Early New High German, or Bouma et al. (2012) (on the automated syllabification of Middle Dutch). One characteristic feature that sets this kind of research into present-day languages apart from historical language research, is that for modern languages, researchers typically are able to normalize noisy language data into an existing, canonical standard form. Nevertheless, the majority of historical languages lack such a canonical orthographic variant, and therefore, this solution can not always be applied (Souvay et al., 2009): since, in the absence of a standard language, words do not have a single, canonical spelling by which other non-standard spellings can be replaced. It is therefore also common to annotate words with other sorts of labels, instead of attempting an explicit respelling. Apart from part-of-tags, lemmas are often used in this respect: a lemma is a normalized label, which unambiguously links words to the same entry in a lexical resource, such as dictionary, if they only differ in inflection or spelling (Knowles and Modd Don). Lemmatization too has been an active area of research in computational linguistics, especially for languages with a rich inflectional system. Many studies apply a combination of existing taggers (e.g. for contextual disambiguation) with a spelling normalization component to better deal with unknown words.

One rough distinction which could be made, is between ‘online’ and ‘offline’ approaches. In ‘online’ approaches, when confronted with an unknown word at the testing stage, systems will explicitly attempt to identify the unknown word as a spelling variant of a known token (i.e. a token which was seen during training) before applying any disambiguation routines. To this end, researchers typically apply a combination of (weighted) string distance measures (e.g. Levenshtein distance: Levenshtein, 1966). While this online strategy might result in a high recall, it can also be expensive to apply, because the unknown token has to be matched with all known tokens and some string distance measures can be costly to apply. The offline strategy follows a different road: at training time, it will attempt to generate new plausible spelling variants of known tokens, through aligning words and applying common edit patterns to generate new forms. At the testing stage, new words can simply be matched. Interestingly, this approach will be fast to apply during testing (and might result in a high precision), but it might also suffer from the noise being introduced in the variant generation phase. Van Halteren et al. (2013) have reported the successful application of such an approach, which is in many ways reminiscent of the generation of search query variants in Information Retrieval research.

**“Deep” Representation Learning**

In this paper, we describe a novel system for sequence tagging using “deep” representation learning, which to the best of our knowledge has barely been applied to the problem of sequence tagging for historic and/or minority languages. Importantly, we will show that a single system reaches state of the art tagging performance across a variety of datasets, even without the problem-specific fine-tuning of hyper-parameters. Deep Learning is a popular paradigm in Machine Learning. It is based on the general idea of neural networks, software system in which information is propagated through a layered architecture of inter-connected neurons, that iteratively transform the input information and feed it to subsequent layers in the networks. While theoretically, the idea of neural networks has been around for a long time, it has only recently become feasible to train these architectures, because of the millions of parameters that typically have to be optimized during training.

More formally, Deep Learning is part of a broader field called representation learning or feature learning. Deep learning can indeed be contrasted with more traditional approaches in Machine Learning, in that is very much geared towards finding good representations or “features” in the input data. Simplifyingly put, traditional learning algorithms would be very much dependent on a researcher’s representation of the input data and it would try to solve a particular task on the basis of the particular features suggested by the researchers. In Deep Learning, the process of feature engineering is to a larger extend outsourced to the learning algorithm itself. Instead of solely relying on the pre-specified features in the input data (hand-crafted by scholars themselves), deep neural networks will attempt to independently recombine the existing input features into new, nonlinear combinations of features, which will potentially offer a better representation of the input data. It does so by iteratively projecting the input onto subsequent layers of information units, which are typically said to increasingly capture more abstract and complex patterns in the input data.

The intuition behind deep learning is typically illustrated using an example from computer vision, the field where some of the earliest breakthroughs in Deep Learning have been realized (e.g. in handwritten digit recognition). In computer vision, images are typically represented as a two-dimensional raster of pixels, which can take certain values across a series of input channels (e.g. one for every color in the RGB spectrum). When propagating an input image through a layered neural network, it has been shown that the earliest layers in the network will be receptive to very raw, primitive shapes in the data, such as a corner, parts of a curve or a stark light-dark contrast. Only at subsequent layers in the network, these primitive forms are recombined into higher-level units, such as an eye or ear. At still higher layers, the network will eventually become sensible to complex units such as faces. It can therefore be said, that such neural networks is able to learn and detect increasingly abstract (re)combinations of the original features in the input. Interestingly, it has been shown that visual perception in mammals works in a highly similar, hierarchical fashion. Because of its hierarchical nature, this form of learning is commonly called ‘deep’, since it increasingly captures ‘deeper’ aspects of the problem it is looking at.

**Architecture**

The system architecture we describe here primarily builds upon two basic components derived from recent studies in Deep Learning NLP research: (i) temporal convolutions to model the orthography of input words at the character-level; and (ii) word embeddings to model the lexical context surrounding the input tokens, for the purpose of disambiguation. We will first introduce the type of ‘subnets’ associated with both components in our components, before we one on to discuss how our architecture eventually combines these subnets.

*Convolutions*

Convolutions are a popular approach in computer vision to battle the problem of ‘translations’ in images. Consider an object classification system, whose task it is to detect the presence of certain objects in an image. If during training, the system has come across e.g. a banana in the upper-left corner of a picture, we would like the system to be able to detect the same object in new images, even if the exact location of that object has shifted (e.g. towards the lower-right corner of the image). Convolutions can be thought of as a fixed size window (e.g. 5x5 pixels) which gets slided over the entire input image in a stepwise fashion. Convolutions are therefore also said to learn ‘filters’, in that they learn a filtering mechanism to detect the presence of certain features, irrespective the exact position in which these features occur in the image. Importantly, each filter will scan the entire image for the local presence of a particular low-level feature (e.g. the yellow-colored, curved edge of the banana), and detect it irrespective of the exact position of the feature (e.g. lower-right corner versus upper-left corner). Typically, convolutions are applied as the initial layers in a neural networks, before passing on the activation of filters to subsequent layers in the network, where the detected features can be recombined into more complex objects, e.g. the entire banana. Whereas convolutions in vision research are typically two-dimensional (cf. raster of pixels), here we apply convolutions over a single dimension, namely the sequence of characters in an input word. The idea to apply convolutions to character series in NLP has recently been pioneered by LeCun et al., reporting strong results across a variety of higher-end text classification tasks. Such convolutions have also been called ‘temporal’ convolutions, because they model a linear sequence of items.

Representing input words as a sequence of characters (i.e. at the sub-word level) has a considerable advantage over more traditional approaches in sequence tagging. Here, input words are typically represented using a “one-hot vocabulary encoding”: given an indexed vocabulary of *n* known tokens, each token is represented as a long vector of *n* values, in which the index of one token is set to 1, whereas all others values are set to 0. Naturally, this leads to very sparse, high-dimensional word representations. In such a one-hot representation it is therefore not uncommon to apply a cut-off and only encode the more frequent vocabulary items, in order to battle the sparsity of the input data. Of course, a one-hot vector representation (especially with a severe cutoff) can obviously not represent new, unseen tokens at test time, since these do not form part of the indexed training vocabulary. Most systems for e.g. PoS-tagging would in those cases back off to additional (largely hand-crafted) features to represent the target token, such as the final character trigram of the token to capture the token’s inflection. The character-level system presented here does not struggle to model out-of-vocabulary words. This is worthwhile characteristic for our use-case: with low-resource languages with a rich orthographical variation, the dimensionality of word-level one-hot vectors dramatically increases because the vocabulary is much larger (and thus, the effect of applying a cutoff will also be more dramatic). Additionally, the problem of unknown target tokens at test time is of course much more prominent, due to spelling variation, as in traditional languages. Therefore, our architecture complements a conventional one-hot encoding with a convolutional component.

Following the convolutional strategy, our system models each input token as a sequence of characters, in which each character is represented by a one-hot vector at the character level. As the example in Table 1 shows, this leads to an at first sight primitive, yet much lower-dimensional token representation than with a traditional token-level one-hot encoding, which, additionally, is able to represent out-of-vocabulary items. We include all characters in the character index used. We specify a certain threshold *t* (e.g. 15 characters): words longer or shorter than this threshold are respectively truncated to the right to length *t*, and shorter words are padded with zero-filled vectors. In terms of computer vision research, this representation models characters as if they were ‘color channels’, with one channel dedicated to a single character.

Table 1: Illustration of the character-level representation of input tokens. Dummy example with length threshold *t* set at 15 and a restricted character vocabulary of size 7, for the Middle Dutch word *coninghinne* (‘queen’).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *char* | **c** | **o** | **n** | **i** | **n** | **g** | **h** | **i** | **n** | **n** | **e** | **-** | **-** | **-** | **-** |
| c | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| e | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | e | 0 | 0 | 0 | 0 |
| g | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| h | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| i | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| n | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| o | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Our model now slides convolutional filters over this two dimensional representation, with a size of 3 and a stride (or ‘step size’) of 1. In practice, this will mean that the convolutional layer will iteratively inspect, partially overlapping, consecutive character trigrams in the input string. This representation inspired by the generic approach to sequence modeling in NLP followed by LeCun (character-level convolutions), Dieleman (one-dimensional convolutions for music) and Greffenstetter (word-level convolutions on the basis of pretrained embeddings). At the input stage, each target token is represented by a *n* x *t* matrix, where *n* represents the number of distinct characters in a dataset and *t* is the uniform length to which each target token is normalized through padding with trailing zeros or cutting.

We originally motivated the choice for a convolution model to model orthography on the basis of the following considerations. When using a filter size of e.g. 3, the receptive field size of the learned filters roughly corresponds to that of syllables-like groups in many languages (e.g. consonant-vowel-consonant groups). One would therefore expect the filters to become sensitive morpheme-like characteristics in. The most interesting aspect of convolutions seems that the learned filters are being detected across all positions in the input string, which is likely to render the detection of morphemes less sensitive to their exact position. This is a valuable property for our corpora, since the non-standard orthography often allows the introduction of silent characters (e.g. ‘gh’ for ‘h’), causing local character shifts in the input, or the kind of ‘translations’ in computer vision, to which we know convolutions are fairly robust (‘gelik’ vs ‘ghelik’).[[1]](#footnote-1) Finally, it should be stressed that the learned filters can be expected to offer a flexible string representation: a filter which learned to detect the pattern ‘dae’ might still be responsive to the pattern ‘dai’, which is useful since ‘i’ and ‘e’ were often used interchangeably as vowel lengthening graphemes in medieval languages. We will come back to these intuitions in the discussion below, since they can be partially demonstrated to be wrong.

*Embeddings*

Many sequence taggers crucially depend on contextual information for disambiguating tokens. In the classic example ‘The old man the boat.’, contextual information is needed to determine to that ‘man’ in this sentence should be tagged as *verb*, instead of the more common *noun* tag for this token. Most modern taggers therefore integrate a lexical representation of the immediate context on an input token, e.g. the two tokens preceding the input token to the left and a single token to left. Simplifyingly put again, traditional architectures model will typically represent this context using a one-hot encoder at the token-level: for each token in the positional skeleton surrounding the input token, a high-dimensional vector will the record the presence using the index of known training tokens. To battle sparsity in the contextual representation too, it is again not uncommon to apply a frequency-based cutoff here, or even only record the presence of a small set of functors surrounding the input token or even hand-crafted features (such as the last two characters of the word preceding the input string).

Because of the sparsity associated with one-hot encodings, research in NLP has stressed the need for smoother, distributional representations of lexical contexts in this respect. As the size of the context windows considered increases, it becomes increasingly difficult to obtain good, dense representations of contexts, if one restricts the model to strict n-gram matching. A simple example: in a word sequence like ‘the black fat cat’ or ‘the cute white cat’, the part-of-speech category of ‘cat’ can be reasonably inferred from the fact that two adjectives precede the target token. When presented with the new examples ‘the fat white cat’ or ‘the cute fat cat’, the left contexts ‘fat white’ and ‘cute fat’ will fail to find a hard match to one of the training contexts in the cast of a strict matching process. Naturally, we would like a model to be robust in the face of such naive transpositions.

An important line of research in nowadays NLP therefore concerns the development of models which can represent words not in terms of a simple one-hot encoder, but using smooth, lower-dimensional word representations. Much of this research can be situated in the field of distributional semantics, being guided by the so-called “Distributional Hypothesis” that words in themselves do not have real meaning in isolation, but that words primarily derive meaning from the words they tend to co-occur with, i.e. their lexical context (Firth etc.). While \*blarf is a non-existing word, its use in the following sentences suggests that it refers to some sort of domestic animal, perhaps a dog: “I’m letting the \*blarf out”, “I’m feeding the \*blarf”, “The \*blarf ate my homework”. Thus, by modelling patterns of word co-occurrences in large corpora, research has demonstrated that various unsupervised techniques can yield numeric word representations which offer useful approximations of the meaning of words. The vector which make up such word representations are also called “word embeddings”, because they reflect how words are contextually “embedded” in corpora.

The past years have witnessed a clear increase in the interest for distributional semantics, in particular in the field of deep learning for NLP. In a series of influential papers, Mikolov et al. at Google have introduced an influential skipgram method to obtain word vectors known as “word2vec”. Because the underlying method attempts to optimize a fairly simple training criterion, it can be easily applied to vast quantities of text, yielding vectors which have shown excellent performance in a variety of tasks. In one popular example, Mikolov et al. even showed that is possible to solve relatively advanced analogical problems applying plain vector arithmetic to the word embeddings obtained from large corpora. The result of the arithmetic expression “vector(king)-vector(man)+vector(woman)”, for instance, yielded a vector which was closest to that of the word “queen”. Other illustrative examples include: “vector(russia)+vector(river) ~ vector(wolga)” and “vector(paris)-vector(france)+vector(belgium) ~ vector(brussels)”.

Follow-up studies have stressed that other techniques can yield similar results and that the theoretical foundations of the word2vec algorithm still need a lot of clarification. Nevertheless, the fact that Mikolov et al. published a highly, efficient open-source implementation of the (in the meanwhile patented) “word2vec” algorithm has greatly contributed to the state of the art in the field of word representation learning. An increasing number of applications in NLP include some form of word embedding strategy in their pipeline as a “secret sauce”. An essential quality of modern word embeddings is that it has been shown that they are not restricted to semantic aspects of words but that they also model morpho-syntactic qualities of words, illustrated by the fact that purely syntactic analogies often can also be solved “vector(her)-vector(she)+vector(he) ~ vector(his)”.

The application of word embeddings in NLP research is simply vast. Relevant examples for the present research includes, for instance, the type of word embeddings used in the Twitter tagger described by Godeleer et al, but the literature abounds in other examples. In our system, we train a vanilla word2vec model, using a popular implementation of the original skipgram-model. This model learns to project the tokens in our training data in a vector space consisting of 300 dimensions, which is common dimensionality in the literature. The general intuition underlying the embeddings subnet in our architecture, is that the underlying skipgram model will have learned similar embeddings for words that certain qualities, both at the semantic and morpho-syntactic level – in the ‘cat’ example above, we might expect that a model would learn similar embeddings for the ‘white’ or ‘black’, because they both are both adjectives referring to colors. Additionally, an interesting aspect for our medieval data, is that the word embeddings learned might also be sensitive to orthographic or dialectical variation, providing similar embeddings from word that are spelling variant of each other, or words that are typical of specific dialects. We will come back to this issue in the discussion.

For the subnets representing the lexical context on either side, we first construct a simple one-hot encoding of the context tokens . We then project these tokens onto a standard dense layer which has the same dimensionality as our skipgram model (taking the form of a simple matrix multiplication). Importantly, this implies that we can initialize the weights of this layer using the pretrained embeddings from the skipgram model, but that these weights can be further refined during the training phase (see below). This initialization strategy can be expected to speed up the convergence of the model. We simple concatenate the dense vectors obtained for each context word on either side of the target token into a left-context subnet and a right context subnet. The number of context words is of course a hyper-parameter which can be further tuned.

*Combining the subnets*

In our system, we again use a network strategy to combine in the convolutional subnet, used to represent the focus token, and the subnets representing the left and right context of the input token. The activations of the convolutional layer are flattened and concatenated; next, they are fed forward in the network using a plain dense layer that has a dimensionality of size *k* (e.g. 1028). The concatenated embeddings of the left and right context subnets are then also propagated through a dense layers of size *k*. Finally, we also include a one-hot encoding of the target tokens, which is propagated through a dense layer of size *k*. This final layer is important to have the network obtain a good fit of known tokens: early experiments showed that the convolutional layer alone has insufficient capacity to reliably represent the entire vocabulary of training tokens (which is clearly is an issue to be addressed in future research).

At this point, we have four subnets in the network, each of dimensionality *k*: one resulting from the one-hot encoding of the input token, one resulting from the convolutional encoding of the input token, and two resulting from the embeddings subnets for the left and right context. Finally, we concatenate the output of these subnets into a flat vector of size 4 x *k* and project it onto an output layer which for each class in the sequence modeling task has a unique output neuron. A standard “softmax” normalization is applied to normalize the activations of these output neurons, as if they were probabilities (summing to 1). All the parameter weights in this architecture are first initialized using a random distribution, except for the contextual subnets, which are initialized using the word2vec encoding. We train the neural network using gradient descent. Briefly put, this learning algorithm will start predicting a batch of the training examples using the randomly initialized weights. It will then calculate the prediction loss in which this results with respect to the correct labels, which will initially be extremely high of course. Then, it will determine how each weight individually contributed to this loss and thus apple a small update to this weight, which would have decreased the prediction error which it caused.

During a fixed number of epochs (e.g. 100), we feed the training data through this network, divided in a number of random jumbled mini-batches, thus each time tightening the fit of the network with respect to the correct class labels. The actual loss of the network is not measured in terms of accuracy, but in terms of cross-entropy, a common loss measure in the field. We adopt a mini-batch size of 50: this is a fairly small batch size in the literature, but this restricted batch size proved necessary to avoid the higher-frequency items from dominating the gradient updates in a mini-batch. We use the adaptive, so-called “adagrad” optimization mechanism, which will for each weight individually. Importantly, this avoids the need to empirically set a learning rate at the beginning of training, since the algorithm will independently re-adapt this rate, keeping track of the track of the loss history on a per-weight basis.

*General comments*

All dense connections in this network make use of a “relu” activation (*rectified linear unit*), will ignore negative weights, encouraging connections to only collect positive features. Additionally, all dense layers of the so-called “dropout” procedure during training (with *p*=0.50), meaning that in each mini-batch, half of the connections in the dense layers are randomly dropped. This technique has various beneficial effects on training, the most important being that it forces the network to be less dependent on the specific presence of certain features in the input, thus avoiding over-fitting on specific instances in the training data. Apart from its general usefulness during training, the dropout procedure has one important advantage in our system: during training this procedure will also affect the dense layer mapping the one-hot encoding of the input tokens to the target labels, which may or may not affect the single activation of the actual target token. When this activation is indeed affected, this will simulate the presence of an unknown token during training, since it will seem no one-hot encoding is present for a particular token. This will cause the network to shift the attention during the backpropagation procedure to the convolutional layer, and thus also learn useful filters for known words.

An implementation of the proposed architecture is freely available is from the public software repository associated with this paper – where possible, this repository also holds the data sets used below, as well as trained models which can be applied to new data out of the box. Our system is implemented in Python, using the *keras* library built on top of *theano*; the latter library provides automatic differentiation for arbitrary graphs and allow to execute code on the GPU. Other major dependencies of our code include *skikit-learn* and *gensim*. The system is language-independent and could in principle easily be retrained on new, similar datasets.

**Datasets**

In this paper we report results on four different Middle Dutch data sets, which offer a representative example of European historic languages. First of all, we use two administrative data sets, both charter collections. The cg-admin contains the charter collection edited by Gysseling (Gysseling, 1977) which has recently been digitized and annotated (Institute for Dutch Lexicography, 1998). These charters all survive in originals predating ca. 1300 AD.[[2]](#footnote-2) Secondly, we use the crm-adelheid collection, a comparable collection of fourteenth century Middle Dutch charters, which has been the subject of a study comparable to ours (Van Halteren *et al.*, 2013). As literary materials, we first of all use the literary counterpart of cg-admin in the Corpus-Gysseling, a collection of Middle Dutch literary texts that all survive in manuscript copies predating ca. 1300 AD (Gysseling, 1980-1987; Institute Dutch for Lexicography, 1998; Kestemont, De Pauw & Daelemans, ). Thirdly, we also use a newly created smaller corpus of Middle Dutch religious texts (e.g. sermons, bible translation, visions, …), most of which are at least semi-literary in nature. This last data set (relig) is released together with this article and is documented in appendix A.

Table 1: Overview of the Middle Dutch corpora and splits

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | Corpus-Gysseling: Literary texts | Corpus-Gysseling: Administrative texts | Miscellaneous religious texts | Van Reenen-Mulder-Adelheid-corpus |
| **Abbreviation** | cg-lit | cg-admin | relig | adelheid |
| **Origin** | Institute for Dutch Lexicography (INL) | Institute for Dutch Lexicography (INL) | CLiPS Computational Linguistics Group and Ruusbroec Institute, University of Antwerp | Radboud University Nijmegen etc. |
| **Number of tokens (incl. punctuation, excl. line breaks)** | 583,135 | 638,243 | 161,568 | 821,619 |
| **Vocabulary size** | 59,524 | 50,433 | 17,547 | 44,183 |
| **# individual texts** | 30 | 1,574 | 33 | 6,792 |
| **Text variety** | Miscellaneous literary texts (mostly rhymed) which survive from original manuscripts dated before 1300 AD | 13th century charters | Miscellaneous religious texts (e.g. Bible translations, mystical visions, sermons) spanning multiple centuries and regions | 14th century charters |

These data sets have all been annotated using common annotation guidelines, although they each show a number of important idiosyncracies. Generally speaking, we adopt the annotation practice which has been developed by Piet van Reenen et al. and which has been carefully documented in the context of the release of the Adelheid tagger-lemmatiser for Middle Dutch charters (Van Halteren *et al.*, 2013). [[3]](#footnote-3) This system adopts a fairly rich part-of-speech tag set, using a head tag, indicating a word’s main part-of-speech category (e.g. verb, noun, adposition, …), followed by a number of subtags or attributes which are often related to inflection (tense, mood, etc.). Most importantly, we deviate from this standard for this research in that we ignore the ‘form’-subtag in this system for a number of reasons. (The “form-t” attribute, for instance, indicates that a word ends in at.) First of all, we try to reduce the sparsity of the tag set by ignoring this information. Secondly, this attribute rarely does more than reproducing the last character of surface forms; unfortunately, the few cases in which the form-attribute does add information (e.g. comparative forms of adjectives) are highly inconsistently annotated throughout the corpora. For most cases, the form-attribute is so highly influenced by the surface form that is becomes questionable whether its included indeed has a beneficial normalization effect, for example, when it comes to subsequent use of the system’s output in applications such as stylometry.

Apart from part-of-speech tags, each token in these corpora has been annotated with a normalized dictionary headform or lemma in a present-day spelling. Broadly speaking, lemmatisation thus allows us to abstract over individual instances of tokens which only differ in inflection or orthography (Knowles et al. 2004). For historical corpora, this has interesting applications in the context of database searching, (diachronic) topic modeling or stylometry. The lemmas used in our corpora all correspond to entries in the Integrated Language Database (GTB) for the Dutch language, which can be consulted online and which is maintained by the Institute for Dutch Lexicography.[[4]](#footnote-4) The major advantage of using a present-day lemma in this context, is that this allows scholars to normalize texts from multiple historic stages of Dutch to the same variant. This in turn allows interesting diachronic analyses of Dutch corpora, such as the ones currently prepared in the large-scale Nederlab project.[[5]](#footnote-5)

**Evaluation**

To evaluate our system, we make of a conventional split of the available data in a train set, development set and a test set.

For evaluation purposes, we report on tokenization, POS tagging and lemmatization results on our four different corpora. We divide the available data for each corpus into two non-overlapping sets: a training set (80%), a development set (10%) and a held-out test set (10%). We use the development set to tune the hyperparameters of our system, which has been trained on the available training data. The final performance of our system is evaluated by training on the training data and testing on the held-out test, instead of the development data.

For the literary and administrative data, we follow two different strategies to divide the available data into training, development and test sets. For the administrative data (the charter collections cg-admin and crm-adelheid), we follow the approach by Van Halteren *et al.* (2013) to make our splitting as comparable as possible to theirs.[[6]](#footnote-6) Each charter in these collections is associated with a historic date: we first sort the all the available charters in each corpus according to their date and assign a rank to each charter. We then divide the material by assigning to the development set all charters which have an rank index ending in “9”, and to the held-out test set all those with an index ending in “0”. The training material consists of the rest of the charters. The advantage of this splitting approach is that charters from various time periods and regions are evenly distributed over the sets.

For the literary data sets (cg-lit and relig), the situation is more complex, since we only have at our disposal a limited set of texts that greatly vary in length (especially for cg-lit). To handle this situation, we adopt the approach suggested by Kestemont, De Pauw and Daelemans (2010). We first assign an index to each item in the texts and normalize these to the range 0.00-1.00. In the development set, we put all items with an index between 0.60-0.70, and in the test set all items with an index between 0.70-0.80. The remaining items are appended to the training material. These splits are of course suboptimal, in the sense that the system operates more “in domain” than with free text. Unfortunately, the enormous divergence between the length of the literary texts renders other splitting approaches (e.g. leave-one-text-out validation) even less optimal. A clear advantage of the evaluation strategy is nevertheless that it allows to assess how well the system is able to generalize to other portions in a text, given annotated material from the same text. This is a valuable property of evaluation procedure, because medieval texts display also a significant amount of intra-text variation (i.e. the same text often uses many different spellings for the same word). Moreover, this strategy is insightful when it comes to incremental or active learning, whereby scholars first tag a part of the text and retrain the system before tackling the rest of a text.

Discussion

Conclusion

Pointers for future research:

More extensive pretraining: both of embeddings and convolutions

More filters, less one-hot

Extend convolutions to modeling context, also sequential nets.

Unfeasible to train on a cpu, requires gpu

Left-to-right

Imporant: no bells and whistles

1. In computer vision, convolutional layers are traditionally alternated with max-pooling layers, although the surplus value of max-pooling is increasingly questioned. In preliminary experiments, max-pooling did not yield a beneficial effect on our data, nor did the introduction of stacked convolutions. [↑](#footnote-ref-1)
2. Thanks to INL and Piet van Reenen. [↑](#footnote-ref-2)
3. The most comprehensive discussion of the annotation guidelines has been provided in the Adelheid project and can be consulted online: http://adelheid.ruhosting.nl/ (last accessed 6 November 2014). [↑](#footnote-ref-3)
4. Online at: http://gtb.inl.nl/ (last accessed 6 November 2014). [↑](#footnote-ref-4)
5. See: https://www.nederlab.nl/home (last accessed 6 November 2014). [↑](#footnote-ref-5)
6. Based on personal communication with the lead author. [↑](#footnote-ref-6)