Digital Scholarship in the Humanities Advance Access published August 26, 2016

Lemmatization for variation-rich languages using deep learning

Mike Kestemont, Guy de Pauw, Renske van Nie, and Walter Daelemans
University of Antwerp, Belgium

Abstract

In this article, we describe a novel approach to sequence tagging for languages that are rich in (e.g. orthographic) surface variation. We focus on lemmatization, a basic step in many processing pipelines in the Digital Humanities. While this task has long been considered solved for modern languages such as English, there exist many (e.g. historic) languages for which the problem is harder 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 a dramatic increase in performance across several domains. The proposed 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 word 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 Middle Dutch data sets, even without corpus-specific parameter tuning.

Correspondence:
Mike Kestemont, University of Antwerp, Prinstraat 13 (S.D. 118), 2000 Antwerp, Belgium.
E-mail:
mike.kestemont@uantwerp.be

1 Introduction

Sequence tagging is a central problem in Natural Language Processing (NLP) Jurafsky and Martin (2009). It includes well-known applications such as part-of-speech (PoS) tagging or lemmatization Chrupala *et al.*, 2008 and Toutanova *et al.*, 2003, which are basic, yet foundational stages in many text processing pipelines in the Digital Humanities. Lemmatization offers a form of normalization which helps to more efficiently process historic text corpora such as newspaper databases, ranging from plain searching Manning *et al.* (2008) to advanced text categorization tasks in e.g. stylometry Van Dalen-Oskam and Van Zundert, 2007. For present-day resource-rich languages, such as English, the task of lemmatization is often considered solved. However, there exist many languages for which these tasks are much more difficult.

First, 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 premodern times, language users typically enjoyed a relatively larger freedom with respect to spelling and spacing, since there only existed local (or at most regional) orthographical conventions (Piotrowski, 2012). Therefore, 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 (Ernst-Gerlach and Fuhr, 2006). Most systems developed for the lemmatization of present-day languages are not equipped to deal with large amounts

doi:10.1093/llc/fqw034

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 more complex processing pipeline (e.g. full syntactic parsers).

Secondly, there exist many present-day resource-scarce languages, for which far fewer resources are available, e.g. in the form of annotated corpora. Both historic and resource-scarce languages make up the focus of many Digital Humanities projects, targeting the better exploitation or preservation of (specific cultural artefacts in) these languages. For these too, however, many of the available lemmatizers 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 lemmatizers for historic and/or low-resource languages is therefore a desideratum across various fields in humanities computing. In this article, 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 this article, we will first survey some of the most relevant related research, before we go on to introduce the field of deep learning. We then describe the architecture of our lemmatizer and discuss the Middle Dutch 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.

2 Related Research

Lemmatization is closely related to morphological analysis and PoS tagging, which are a popular research domain in computational linguistics, with recent studies especially focusing on unsupervised learning and on (low-resource) languages with a rich morphology or inflectional system (e.g. Mitankin *et al.*, 2014). An important boost in NLP research for such languages has been inspired by the increase in 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*. In CMC too, we find a wild proliferation of language variation in written communication, especially affecting surface phenomena such as spelling. 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-Optical Character Recognition(OCR) error correction (Reynaert *et al.*, 2012). 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 been suggested (Schulz *et al.*, 2016), including e.g. transliteration approaches borrowed from Machine Translation (e.g. Ljubešić *et al.*, 2014).

The field of NLP 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 and Marquilhas (2011) and Reynaert *et al.* (2012) for historical Portuguese, Scherrer and Erjavec (2013) for historical Slovene, Bollmann (2012) for Early New High German, or Bouma and Hermans (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 rarely be applied (Souvay and Pierrel, 2009): in the absence of a standard language, words simply 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. Lemmas are often used in this respect Van der Voort van der Kleij, 2005: a lemma is a normalized

label, which unambiguously links words to the same entry (headword) in a lexical resource, such as a dictionary, if they only differ in inflection or spelling (Knowles and Mohd Don, 2004).

In the present article, we conceive of lemmatization as a high-dimensional classification task, where lemmas are assigned to tokens as class labels (i.e. dictionary headwords are the system's output). Therefore, our lemmatizer does not perform any explicit PoS disambiguation (although the system could easily be trained on PoS labels too). Because of this classification setting, the proposed lemmatizer will not be able to predict new, unseen lemmas at test time because it has not seen these labels during training(cf. Chrupala *et al.*, 2008). However, this also prevents the lemmatizer from outputting non-existent, 'gibberish' labels, which is desirable, given the limited size of our data sets. Lemmatization too has been an active area of research in computational linguistics, especially for languages with a rich inflectional system (Daelemans *et al.*, 2009; Scherrer and Erjavec, 2013; De Pauw and De Schryver, 2008; Lyras *et al.*, 2008). Many studies apply a combination of existing taggers (for contextual disambiguation) with an optimized 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 the 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 against the newly generated forms. 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 (e.g. overlapping variants).[1] Van Halteren and Rem (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.

3 'Deep' Representation Learning

In this article, we describe a novel system for lemmatization using 'deep' representation learning, which has barely been applied to the problem of lemmatization for historic and/or low-resource languages specifically. Importantly, we will show that a single system reaches state-of-the-art tagging performance across a variety of data sets, even without the problem-specific fine-tuning of hyper-parameters. Deep learning is a popular paradigm in machine learning (LeCun *et al.*, 2015). It is based on the general idea of multi-layer neural networks, software systems in which information is propagated through a layered architecture of inter-connected neurons (or information units) 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 more complex architectures because of the millions of parameters that typically have to be optimized during training.

Deep learning is part of a broader field called representation learning or feature learning (Bengio *et al.*, 2013). Deep learning can indeed be contrasted with more traditional approaches in machine learning, in that it is very much geared towards finding good representations in the input data or extract 'features' from it. Roughly put, traditional learning algorithms would be very much dependent on a researcher's representation of the input data and would try to solve a task on the basis of the particular features suggested and manually engineered by the researchers. In Deep Learning, the process of feature engineering is to a larger extent outsourced to the learning algorithm itself. Instead of solely relying on the pre-specified features in the input

data (i.e. hand-crafted by humans), deep neural networks will attempt to independently recombine the existing input features into new combinations of features, which typically 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 (Bengio *et al.*, 2013).

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 reported (e.g. handwritten digit recognition: LeCun *et al.*, 1990). 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 called 'edges' (LeCun *et al.*, 2015). 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 are able to learn increasingly abstract (re)combinations of the original features in the input (Bengio *et al.*, 2013). Interestingly, it has been shown that visual perception in mammals works in a highly similar, hierarchical fashion (Cahieu *et al.*, 2014). Because of its hierarchical nature, this form of learning is commonly called 'deep', since it increasingly captures 'deeper' aspects of the problem it is working on.

4 Architecture

The system architecture we describe here primarily builds upon two basic components derived from recent studies in Deep Learning for NLP: (i) one-dimensional, or '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, before we go on to discuss how our architecture eventually combines these subnets.

4.1 Convolutions

Convolutions are a popular approach in computer vision to battle the problem of 'translations' in images (LeCun *et al.*, 1990, 2015). 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. 5×5 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 of 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-coloured, 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 network, before passing on the activation of filters to subsequent layers in the network, where the detected features can be recombined into more complex features (e.g. an 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 Zhang *et al.* (2015), 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 other 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, to battle the sparsity of the input data. Of course, a one-hot vector representation (especially with a severe cut-off) cannot represent new, unseen tokens at test time, since these do not form part of the indexed training vocabulary. Most systems for tasks like 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 (e.g. Zavrel and Daelemans 1999).

The character-level system presented here does not struggle to model out-of-vocabulary words. This is a worthwhile characteristic with respect to the present use case: in languages with a rich orthographical and morphological variation, the dimensionality of word-level one-hot vectors dramatically increases because the token vocabulary is much larger, and thus, the effect of applying a cut-off will also be more dramatic (Kestemont *et al.*, 2010). Additionally, the proportion of unknown target tokens at test time is of course much more prominent than in traditional languages, due to spelling variation. Therefore, our architecture aims to complement, or even replace, a conventional one-hot encoding with a convolutional component.

With respect to the convolutional subnet, our system models each input token as a sequence of characters, in which each character is represented as 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 compared to 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 or are padded with zero-filled vectors. In an analogy to computer vision research, this representation models characters as if they were 'pixels', with a channel dedicated to a single character.

Our model now slides convolutional filters over this two-dimensional representation, with a length 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 is inspired by the generic approach to sequence modelling in NLP followed by Zhang *et al.* (2015: character-level convolutions), as well as Dieleman and Schrauwen (2014: one-dimensional convolutions for music) and Kalchbrenner *et al.* (2014: word-level convolutions on the basis of pre-trained embeddings). At the input stage, each target token is represented by a $n \times t$ matrix, where n represents the number of distinct characters in a data set and t is the uniform length to which each target token is normalized through padding with trailing zeroes or through cutting characters.

We motivate the choice for a convolution approach 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 syllable-like groups in many languages (e.g. consonant-vowel-consonant groups). One would therefore expect the filters to become sensitive morpheme-like characteristics. 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 in a word. 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').[2] Finally, it should be stressed that the learned filters can be expected to offer a flexible string representation: a

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	1	0	0	0	0
g	0	0	0	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	1	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

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 the medieval language considered here.

4.2 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 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 of an input token, e.g. the two tokens preceding the input token to the left and a single token to the right. Roughly put, traditional architectures 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 record the presence using the index of the previously indexed training tokens (e.g. Zavrel and Daelemans 1999). To battle sparsity in the contextual representation too, it is again not uncommon to apply a frequency-based cut-off here, or even only record the presence of a small set of functors surrounding the input token or even handcrafted features (such as the last two characters of the word preceding the input string as they often coincide with morphological suffixes).

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, especially if one restricts the model to strict n-gram matching (Bengio *et al.*, 2003). A simple example: in a word sequence like 'the black fat cat' or 'the cute white cat', the PoS category of 'cat' can be partially 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 case 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 current NLP therefore concerns the development of models which can represent words not in terms of a simple one-hot encoder, but by 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 (Harris, 1954; Firth, 1957). 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 that offer useful approximations of the meaning of words (Baroni and Lenci,

Table 2 Overview of some of the metadata and characteristics of the Middle Dutch corpora used in this study

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	Institute for Dutch Lexicography	CLiPS Computational Linguistics Group and Ruusbroec Institute, University of Antwerp	Radboud University Nijmegen etc.
# 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

2010). The vectors 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 in distributional semantics, in particular in the field of deep learning for NLP. Mikolov *et al.* (2013) have introduced an influential 'skipgram' method to obtain word vectors, commonly 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.* (2013) 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 some clarification (Levy and Golberg, 2014). Nevertheless, the fact that Mikolov *et al.* published a highly efficient open-source implementation of the algorithm has greatly contributed to the state of the art in the field of word representation learning.[3] An increasing number of applications in NLP include some form of word embedding strategy in their pipeline as a 'secret sauce' (Manning, 2015). 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 number of recent studies using word embeddings in NLP research is vast. A relevant example for the present research includes, for instance, the type of word embeddings used in the *Twitter* tagger

Table 3 Tabular overview of some the main token counts etc. of the Middle Dutch corpora used in this study. The 'upperbound unknown' column refers the proportion of tokens that have lemmata which also occur in the corresponding training data set; this is the theoretically maximal score which a lemmatizer could achieve for the unknown tokens

Corpus	Split	Number of words	Unique tokens	Unique lemmas	Number of tokens per lemma	Perc. unknown tokens	Upperbound unknown
relig	train	129,144	14,668	6,103	2.65	NA	NA
	dev	16,143	3,686	2,031	1.93	6.93	68.78
	test	16,143	3,767	1,999	2.01	7.44	72.86
crm-adelheid	train	575,721	41,668	11,163	4.1	NA	NA
	dev	165,883	17,466	5,547	3.42	6.46	78.05
	test	80,015	11,645	4,057	3.1	6.07	79.07
cg-lit	train	464,706	47,460	15,454	3.38	NA	NA
	dev	58,661	11,557	5,004	2.48	6.46	74.38
	test	58,903	11,692	5,001	2.51	6.65	74.94
cg-admin	train	518,017	42,694	13,129	3.42	NA	NA
	dev	54,290	8,753	3,527	2.57	6.39	74.76
	test	65,936	11,318	4,954	2.39	9.18	64.31

described by Godin *et al.* (2014), 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.[4] This model learns to project the tokens in our training data in a vector space consisting of 150 dimensions. The general intuition underlying the embeddings subnet in our architecture, is that the underlying skipgram model will have learned similar embeddings for words that have certain similar 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 are both adjectives referring to colours. 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 words that are spelling variants of each other, or words that are typical of specific dialects.

For the subnets representing the lexical context on either side, we first construct a simple one-hot encoding of the context tokens with a traditional, minimum frequency-based token cut-off of two occurrences. We then project these tokens onto a standard dense layer which has the same dimensionality as our skipgram model—here and elsewhere, taking the form of a simple matrix multiplication, the addition of a bias, and a nonlinearity (a 'rectified linear unit': Nair and Hinton, 2010). 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 simply 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.

4.3 Combining the subnets

In our system (see Figure 1 for an overview), we again use a networked strategy to combine the convolutional subnet, used to represent the focus token, with the subnets representing the left and right context of the focus token. Next, we feed the output of the convolutional net forward in the network using a plain dense layer that has a dimensionality of size $k = 1024$). At this point, the concatenated embeddings of the left and right context subnets have also been propagated to two vectors of size k. We can also include a one-hot encoding of the target tokens, which is propagated through a dense layer

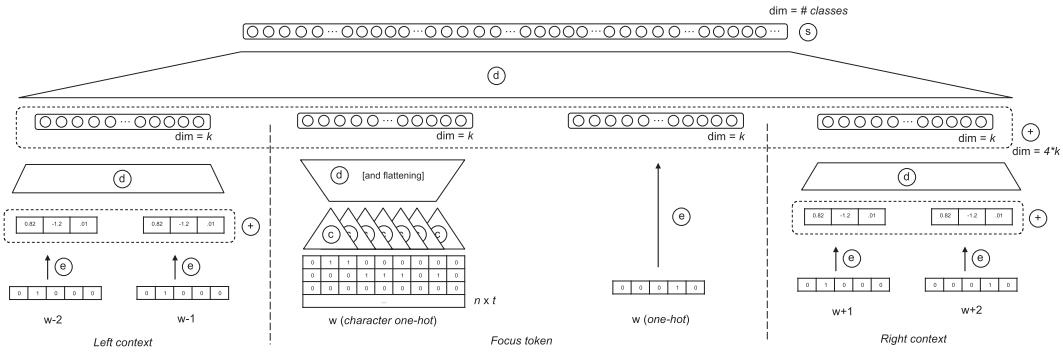

Fig. 1 Schematic visualization of the network architecture evaluated in this article. Embedding layers are used on both sides of the focus token as 'subnets' to model the lexical context surrounding a token. The token itself is primarily represented using a convolutional subnet at the character-level (and potentially also an embedding layer). The results of these subnets are concatenated into a single hidden representation, which is used to predict the correct lemma at the top layer of the network

of size k. This final layer might be important to have the network obtain a good fit of known tokens: below we will explore whether the convolutional layer alone has sufficient capacity to reliably represent the entire vocabulary of training tokens.

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 vector of size $4 \times k$ and project it onto an output layer which for each class in the sequence modelling 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* weights. 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. Then, it will determine how each weight individually contributed to this loss and subsequently apply a small update to this weight, which would have decreased the prediction error which it caused.

During a fixed number of epochs (100 in our case), we feed the training data through this network, divided in a number of randomly jumbled mini-batches, each time tightening the fit of the network with respect to the correct class labels. During training (see Figure 2), the loss of the network is not measured in terms of accuracy, but in terms of cross-entropy, a common loss measure for classification tasks. We adopt a mini-batch size of 50: this is a fairly small batch size compared to the literature, but this restricted batch size proved necessary to avoid the higher-frequency items from dominating the gradient updates. We use the adaptive, so-called Adadelta optimization mechanism (Zeiler, 2012). Importantly, the Adadelta routine prevents 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 loss history on a per-weight basis.

4.4 General comments

All dense connections in this network take the form of a matrix multiplication and the addition of a bias ($X \cdot W + b$) and make use of a 'relu' activation ('rectified linear unit'), which will ignore negative weights outputted by the dense layer by setting them to zero. Additionally, all dense layers make use 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 (Srivastava *et al.*, 2014). This technique has various beneficial effects on training, the most important one being that it forces the network to be less dependent on the specific presence of certain features in the input, thereby avoiding overfitting on specific instances in the training data. An implementation of the proposed architecture is freely available from the public software repository associated with this paper.[5] Where possible, this repository also holds the data used below. Our system is implemented in Python, using the *keras* library built on top of *theano* (Bastien *et al.*, 2012); the latter library provides automatic differentiation for arbitrary graphs and allows to execute code on the Graphics Processing Unit (GPU).[6] Other major dependencies of our code include *scikit-learn* (Pedregosa *et al.*, 2011) and *gensim*.

5 Data Sets

In this article we report results on four different Middle Dutch data sets (see Table 2), which offer a representative example of a medieval European language Van Kerckvoorde (1992). We should stress that our system is fairly language independent, and could easily be retrained on other types of corpora, as long as they use a character alphabet. In this article, we first of all use two administrative data sets, representing charter collections. The CG-ADMIN contains the charter collection edited by Gysseling (1977) which has been digitized and annotated by the Institute for Dutch Lexicography (1998). These charters all survive in originals predating ca. 1300 AD. 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 and Rem, 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 *et al.*, 2010).

Table 4 Example of some training data from the RELIG corpus, from the *Seven Manieren van Heilige Minnen* (‘*Seven Ways of Holy Love*’) by the mystical poetess Beatrice of Nazareth (‘These are seven ways of loving, which come from the highest.’). The first column shows the original input token, the second the lemma label. Note that some forms require a composite lemma, e.g. uit + de for *uten*)

Token	Lemma
seuen	zeven
maniren	manier
sin	zijn
van	van
minnen	minne
;	;
die	die
comen	komen
uten	uit+de
hoegsten	hoogste

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 contained in our code repository.

These data sets have all been annotated using common annotation guidelines, although they each show a number of important idiosyncrasies. Table 4 shows an example of the data. 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-lemmatizer for Middle Dutch charters (Van Halteren and Rem, 2013).[7]

Each token in the corpora has been annotated with a normalized dictionary headform or lemma in a present-day spelling. Broadly speaking, lemmatization allows us to abstract over individual instances of tokens which only differ in inflection or orthography (Knowles and Mohd Don, 2004). For historical corpora, this has interesting applications in the context of database searching, (diachronic) topic modelling 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.[8] 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.[9]

6 Evaluation

To train and evaluate our system, we apply a conventional split of the available data into a training set (80%), a development set (10%) and a held-out test set (10%) see Table 3. While we report the performance and loss for the training and development data, the final performance of our system is reported by evaluating a system on the held-out test data (the system trained on the training data only, i.e. not including the development data). We evaluate the system's performance on the development and test data in terms of overall accuracy, as well as the accuracy for known and unknown words (at test time) separately.

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 and Rem. (2013) to make our splitting as comparable as possible to theirs.[10] Each charter in these collections is associated with a historic date: we first sort 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 a 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 *et al.* (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 and 0.70, and in the test set all items with an index between 0.70 and 0.80. The remaining items are appended to the training material. These splits are not ideal, in the sense that the system operates more 'in domain' than would be the case with truly 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 interesting. 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 the evaluation procedure because medieval texts also display 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 would first tag a part of the text and retrain the system before tackling the rest of a text.

7 Results

In Tables 5–8 we report the results of variants of our full architecture on all four data sets. As a baseline, we also report scores for the 'Memory-Based Tagger' (MBT), a popular sequence tagger (especially for modern Dutch) which is based on a nearest-neighbour reasoning approach. While this tagger is typically used for PoS tagging, it is equally useful for lemmatization in the classification framework we adopt here. We left most of the parameter settings at their default values, except for the following: we set the global matching algorithm to a brute nearest neighbour search and for unknown words, we set the global metric to the Levenshtein distance. Additionally, to account for the spelling variation in the material, we allowed a set of 1000 high-frequency function words to be included in the

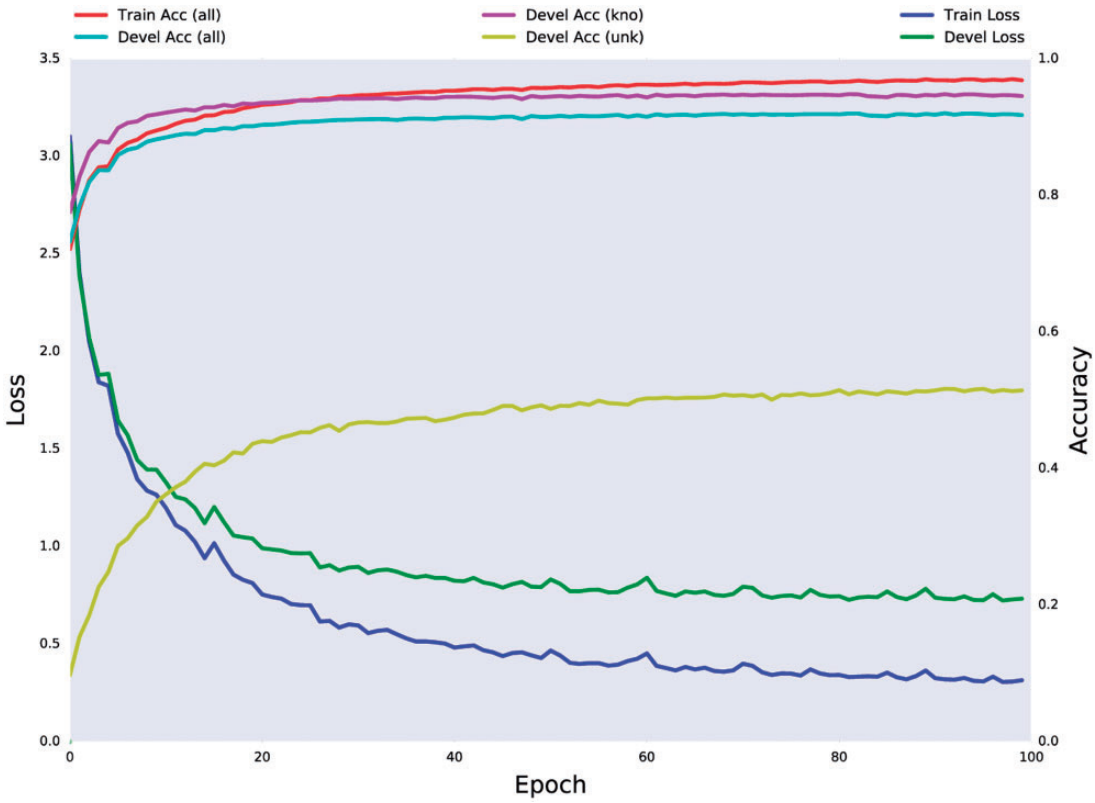

Fig. 2 Visualization of the typical training progress of the neural network (CG-LIT), showing how the prediction loss steadily decreasing over epochs, whereas the lemmatization accuracy gradually increases

Table 5 Training and development results (lemmatization accuracy for all, known, and unknown words) of the full systems for each of the four data sets. Note that for the training data, we cannot differentiate between known and unknown tokens

Corpus/Phase	Train	Dev		
SUBSET	All (%)	All (%)	Known (%)	Unknown (%)
RELIG	97.63	91.26	94.67	45.44
CG-LIT	96.88	91.78	94.56	51.45
CRM-ADELHEID	98.38	93.61	96.17	56.59
CG-ADMIN	98.89	95.44	97.89	59.48

features used for the contextual disambiguation. Because of the experimental setup using a separate train, development and test split of the available data, our results are not directly comparable to previous studies in this topic, which typically used 10-fold cross validation. This is a computationally intensive setup which is not realistic for our neural approach, given the considerable time required for training. Finally, our training data only covers ca. 90% of the training data that other systems have available in a 10-fold cross validation setup without a separate development set. This means that our system can be expected to have a slightly worse lexical coverage, so that a minor drop in overall accuracy can be expected here.

The main results are reported in Table 5 and 6, where we list the results for the train, development and test splits for all four corpora. In Figure 2, we have added a visualization of the typical training progress of the neural network (for the training on the CG-LIT corpus), showing how the prediction loss steadily decreases over epochs, whereas the lemmatization accuracy gradually increases. For the test

Table 6 Final test results (lemmatization accuracy for all, known, and unknown words) of the full systems for each of the four data sets. We included the baseline offered by the Memory-Based Tagger, which is especially competitive in the case of known tokens

Corpus/method	MBT			Our system		
Subset	All	Known	Unknown	All	Known	Unknown
RELIG	88.50	94.03	19.65	90.97%	94.50%	47.04%
CG-LIT	88.72	93.93	15.64	91.67%	94.45%	52.65%
CRM-ADELHEID	91.87	95.90	29.44	93.95%	96.15%	59.86%
CG-ADMIN	88.35	95.03	22.30	90.91%	95.35%	46.99%

results, we include the results for MBT. As can be seen, the algorithm obtains a reasonably good fit of the training data with overall scores above 96%. Naturally, there is a considerable drop when applying the trained system to the development and test splits. Importantly, there is no large difference between the development and test scores, with the exception of the CG-ADMIN test split, which does seem to be much more difficult than the development split. Overall, the RELIG and CG-LIT appear to be the most difficult corpora—the former arguably because of its limited size; the latter because of the rich diversity in text varieties which it includes. Both charter corpora yield similar scores, with CRM-ADELHEID being slightly more difficult than CG-ADMIN. The scores for known word accuracies are fairly solid and always above 94% (in the case of CG-ADMIN even almost 98%). As can be expected, the unknown word accuracies are much lower, ranging from 45.55% for the smaller RELIG corpus, to the encouraging 59.48% which we can report for the CG-ADMIN corpus. All in all, this suggests that for large enough corpora, the present system can accurately lemmatize over half of the unknown words in unseen material.

As can be gleaned from Table 6, MBT yields a very strong baseline for the known words in the evaluation splits, and for this category, the new system only marginally outperforms the baseline. It is of course especially for the unknown words that the present, optimized system yields superior results, where the improvement spans dozens of percentage points. In comparison to other studies, the results appear to be strong: Kestemont *et al.* (2010) report a lemmatization of 45.52% for the unknown words in the CG-LIT corpus, which the present approach clearly improves upon (52.65%), notwithstanding the fact that it has access to less training data. Van Halteren and Rem (2013) use 10-fold cross-validation on the CRM-ADELHEID and report an overall lemmatization accuracy of 94.97%; the present system is just below that score (93.95%). Van Halteren and Rem (2013) do not report scores for known and unknown words separately, which makes it difficult to speculate about this issue but, apart from the validation procedure itself, the fact that the present system has less training data available might be responsible for this drop. Additionally, their tagger is in fact based on a meta-learner which combines the output of several independent taggers, an approach which will almost always outperform the performance of a single system.

The present article placed a lot of emphasis on the use of character-level convolutions as a replacement for the traditional one-hot encoding. For unknown tokens, such an approach seems advantageous, but the danger of course exists that such a modelling strategy will lack the capacity to effectively model known words, for which a one-hot encoding does seem a viable approach. In Tables 7 and 8, we present the results of two version of the system: the first version only includes a convolutional model of the focus tokens, whereas the second version adds a one-hot embedding of the focus tokens. Both versions use 5,000 filters, but leave out any contextual features, to be able to zoom in on the convolutional aspects of the

Table 7 Developments results for two variants of the system: both variants have a convolutional subnet (5,000 filters) but no contextual subnets. Importantly, the second system adds an embeddings layer which might allow a tighter fit of the known words than the convolutional-only variant

Corpus/method	Only convolutions (5,000 filters)			Convolutions (5,000 filters) and a one-hot encoding of the focus token		
Subset	All (%)	Known (%)	Unknown (%)	All (%)	Known (%)	Unknown (%)
Relig	89.45	92.69	45.97	87.59	91.67	32.83
Cg-Lit	88.46	91.20	48.79	82.86	87.42	16.86
Crm-Adelheid	89.58	92.28	50.57	87.96	91.73	33.38
Cg-Admin	91.85	94.45	53.76	89.62	93.28	36.03

Table 8 Test results for two variants of the system: both variants have a convolutional subnet (5,000 filters) but no contextual subnets. Importantly, the second system adds an embeddings layer which might allow a tighter fit of the known words than the convolutional-only variant

Corpus/Method	Only convolutions (5,000 filters)			Convolutions (5,000 filters) and a one-hot encoding of the focus token		
Subset	All (%)	Known (%)	Unknown (%)	All (%)	Known (%)	Unknown (%)
Relig	88.95	92.38	46.38	87.37	91.53	35.55
Cg-Lit	88.25	91.03	49.21	82.46	87.11	17.28
Crm-Adelheid	89.91	92.24	53.88	88.63	92.00	36.40
Cg-Admin	86.54	90.91	43.39	83.27	88.81	28.46

model only. The results are somewhat surprising in that it appears that the first, convolutional-only version consistently outperforms the system that includes a one-hot embedding. Interestingly, the drop in performance across all corpora comes from the unknown words: it seems that the presence of the one-hot embeddings dominates the optimization process, so that much less information is able to be backpropagated to the convolutional subnet, leading to a much looser fit of the unknown vocabulary in the test data.

8 Discussion

What does the presented architecture learn? To address this interpretative question, we offer two intuitive visualizations. In Figure 3, we plot the embeddings which are learned by the initial *word2-vec* model (Mikolov *et al.*, 2013) which is fed into the embedding layers of the networks before training commences (here for the cg-lit corpus). We use the *t-SNE* algorithm (Van der Maaten and Hinton, 2008) which is commonly used to visualize such embeddings in two-dimensional scatterplots. We have performed a conventional agglomerative cluster analysis on top of the scatterplot, and we added colours to the word labels according to this cluster analysis as an aid in reading. This visualization strongly suggests that morpho-syntactic, orthographic as well as semantic qualities of words appear to be captured by the initial embeddings model. In purple to the right, we find the names of the biblical evangelists (*mattheus*, *marcus*, ...) which clearly cluster together on the basis of semantic qualities. In light-blue (top-left), we find a semantic grouping of the synonyms *seide* and *sprac* ('[he] said'). Orthographic similarity is also captured, which is remarkable because the embeddings model does not have access to sub-word level

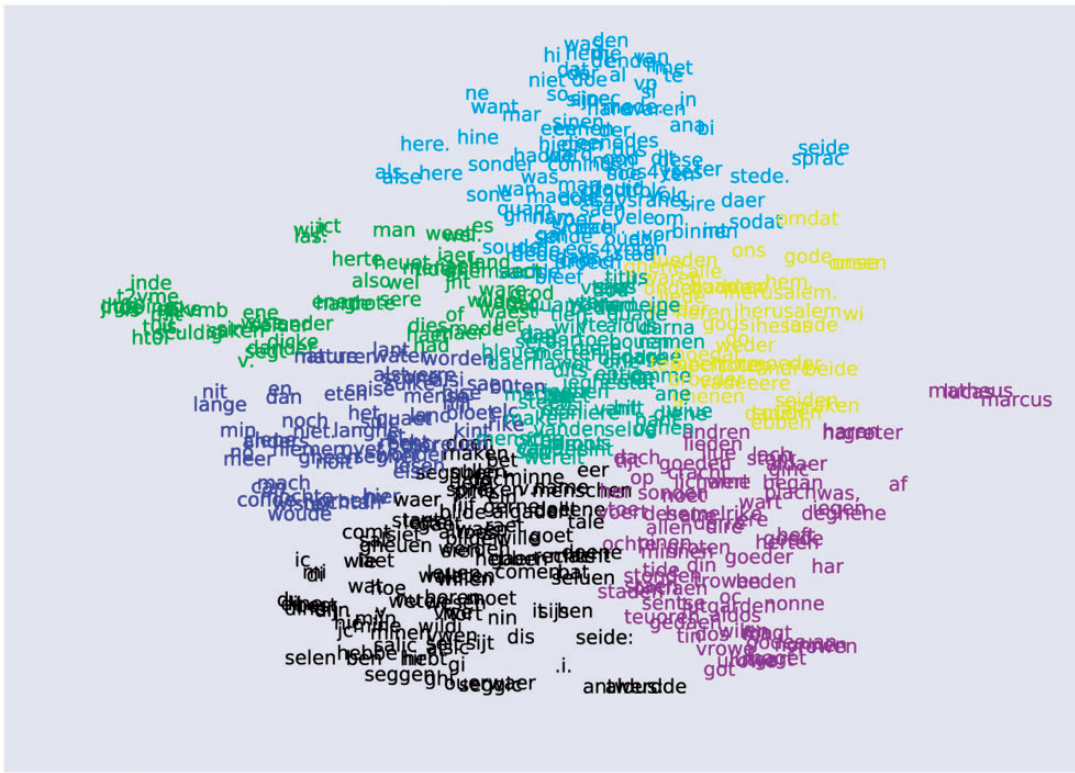

Fig. 3 Two-dimensional scatterplot created by the *t-SNE* algorithm, representing the embeddings learned by the *word2vec* pretrainer (for the CG-LIT corpus). These embeddings are eventually fed into the embedding layers of the network before training commences (so that they can be further optimized in the light of a given task). Both morpho-syntactic, orthographic and semantic qualities of words appear to captured by the model

information: the spelling variants *als* and *alse* ('if') are for instance grouped. At the morpho-syntactic level, we see that the subordinating conjunctions *want* and *maer* cluster (light-blue). Even collocational patterns are present, e.g. *ict*, *wi*, and *las*, which cluster together in green), a clustering which seems to stem from the collocation 'as I/we read'.

At the subword level, it is equally interesting to attempt to inspect the filters which have been learned at the character level. In Table 9, we offer a representation of a number of interesting filters which have been learned. For each position in the convolutional slots, we record the three characters which had the highest activation with respect to this specific filter. While this discussion is necessarily anecdotal, these results do suggest that the convolutional filters have indeed become sensitive to both morphemic and orthographic information. Filter 17, for instance, is clearly sensitive to the combination of the vowel *i* and a velar stop *c*, *k*, or *ck*. This filter would be useful to detect the frequent pronoun 'ick' ('I') but might be equally useful for similar morphemes inside words (e.g. *sticke*, 'piece'). Filter 40, on the other hand, would be sensitive to the presence of these velar stops in combinations with the front vowel *a* (potentially long *aa*). Here too, we clearly notice a certain flexibility in the filter as to the exact spelling of the velar stop or the length of the vowel. Filter 8, finally, does not seem sensitive to a particular consonant-vowel combination, but rather seems to pick up the mere presence of the labio-dental fricative *f* or *v* in the final slot of the filter, including the allographic spelling *u* for the latter

Table 9 Anecdotal representation of a number of interesting convolutional filters learned by the network, after training on the CG-LIT corpus. For three example filters, we show for each slot, the three characters with the highest activation across the alphabet (activation scores shown between brackets). The patterns suggest that the network has become sensitive to both morphemic and orthographic features, but is flexible as to where these patterns should occur in the filter's slots

Filter ID	Pos	1st char	2nd char	3rd char
Filter 17 ('ick'?)	1	i (1.08)	k (1.06)	c (0.81)
	2	k (1.71)	c (1.47)	i (0.67)
	3	k (3.46)	c (2.48)	i (i.17)
Filter 40	1	a (0.69)	b (0.33)	l (0.30)
('ak', 'ack',	2	a (0.61)	c (0.59)	m (0.56)
'aak', 'aac',. . .)	3	a (1.18)	k (1.17)	c (0.90)
Filter 8 (u-v-f as	1	u (0.41)	a (0.29)	k (0.25)
consonant)	2	s (0.59)	v (0.48)	u (0.47)
	3	f (1.40)	v (1.33)	u (0.72)

consonant. Here too, the filters allow these characters to some extent to be detected in various slots in the filters, thereby illustrating how the filters have obtained a certain translational invariance as is typically achieved in image recognition.

9 Future Research

While our system yields excellent results across all data sets, a number of interesting issues remain to be addressed in future research. First of all, our architecture does not consider any form of recurrent neural networks, through which recently excellent results have been obtained in various tasks across NLP. Of course, it would be well feasible to run e.g. an Long Short-Term Memory (LSTM) layer (cf. Hochreiter and Schmidhuber, 1997) over our convolutions or embeddings, instead of a traditional dense layer. Preliminary experiments on these specific data sets did not suggest that dramatic performance increases can be gained here (perhaps because of the limited length of the sequences considered here), but further research is needed in this respect. Additionally, it might be worthwhile to combine LSTMs with an approach which does not only slide convolutions over the focus token, but also over the lexical context, thus making the system more robust towards unknown tokens in the focus token's context as well. Bidirectional layers are another interesting extension in this respect, since the present model only considers a straightforward left-to-right strategy. Another possible extension would be to pre-train certain components of the model (e.g. embeddings or character convolutions) on larger unannotated data sets (provided these are available). Further research should also overcome the limitation of the present system that it cannot predict new, unseen lemmas at test time. Here too, recurrent network architectures that are able to generate new strings on the basis of an input string will prove a valuable approach (LeCun *et al.*, 2015).

One strong component of the system, is that a single hyperparameterization yields strong results across a variety of corpora. This suggests that further performance increases can be obtained through more specific fine-tuning. Additionally, our system reaches high scores although it is only trained on 80% of the available data: other papers on this field have used 10-fold cross-validation and thus average over systems which have available 90% of the training data in each run. As previously mentioned, however, the different evaluation procedures make it difficult to compare these results directly. The good results obtained are also remarkable in light of the fact that our system does not have access to many of the conventional 'bells and whistles' which other taggers offer (such as removing highly uncommon tags for certain tokens). Generally speaking, one serious drawback is that training this kind of architecture is extremely time-consuming and is in reality only feasible on GPUs, which still come with serious memory limitations.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the TITAN X used for this research. Additionally, the authors would like to thank Dr Sander Dieleman for his valuable feedback on earlier drafts of this article.

References

Baroni, M. and Lenci, A. (2010). Distributional memory: a general framework for corpus-based semantics. *Computational Linguistics* 36(4): 673–721.

Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I.J., Bergeron, A., Bouchard, N. and Bengio, Y. (2012). Theano: new features and speed improvements. In *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop: s.p*.

Beaufort, R., Roekhaut, S., Cougnon L. and Fairon, C. (2010). A Hybrid Rule/Model-based Finite-state Framework for Normalizing SMS Messages. In Hajič, J., Sandra Carberry, S., Clark, S. and Nivre, J. (eds), In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Uppsala: Association for Computational Linguistics, pp. 770–9.

Bengio, Y., Ducharme, R., Vincent, P. and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research* 3(March): 1137–55.

Bengio, Y., Courville, A. and Vincent, P. (2013). Representation learning: a review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(8), 1798–1828.

Bollmann, M. (2012). (Semi-)Automatic Normalization of Historical Texts using Distance Measures and the Norma tool. In Mambrini, F., Passarotti, M. and Sporleder, C. (eds), In *Proceedings of the Second Workshop on Annotation of Corpora for Research in the Humanities*. Lisbon, Portugal: Edições Colibri, pp. 3–14.

Bouma, G. and Hermans, B. (2012). Syllabification of Middle Dutch. In Mambrini, F., Passarotti, M. and Sporleder, C. (eds), In *Proceedings of the Second Workshop on Annotation of Corpora for Research in the Humanities*. Lisbon, Portugal: Edições Colibri, pp. 27–38.

Cahieu, C. F., Hong, H., Yamins, D. L. K., Pinto, N., Ardila, D., Solomon, E. A., Majaj, N. J. and DiCarlo, J.J. (2014). Deep neural networks rival the representation of primate IT cortex for core visual object recognition. *PloS Computational Biology* 10(12): e1003963.

Crystal, D. (2001). *Language and the Internet*. New York: Cambridge University Press.

Chrupała, G., Dinu, G., and van Genabith, J. (2008). Learning morphology with Morfette. In *Proceedings of the International Conference on Language Resources and Evaluation 2010*. Marrakech, Morocco: European Language Resources Association, pp. 2362–7.

Chrupała, G. (2014). Normalizing Tweets with Edit Scripts and Recurrent Neural Embeddings. In Toutanova, K. and Wu H. (eds), In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (*Volume 2*: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 680–6, s.p.

Daelemans, W., Groenewald, H. J., and Van Huyssteen, G. B. (2009). Prototype-based Active Learning for Lemmatization. In Angelova, G., Bontcheva, K., Mitkov, R., and Nicolov, N. (eds), In *Proceedings of the International Conference Recent Advances in Natural Language Processing (RANLP 2009)*. Borovets: Association for Computational Linguistics, pp. 65–70.

De Pauw, G. and De Schryver, G. (2008). Improving the computational morphological analysis of a Swahili corpus for lexicographic purposes. *Lexikos* 18, 303–18.

Dieleman, S. and Schrauwen B. (2014). End-to-end Learning for Music Audio. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*: Florence: IEEE, pp. 6964–8.

Ernst-Gerlach, A. and Fuhr, N. (2006). Generating Search Term Variants for Text Collections with Historic Spellings. In Lalmas M., MacFarlane A., Rüger S., Tombros, A., Tsikrika, T., Yavlinsky, A. (eds), *Advances in Information Retrieval*. *Vol.* 3936. Berlin: Springer, pp. 49–60.

Firth, J. R. (1957). A Synopsis of Linguistic Theory 1930-1955. In *Studies in Linguistic Analysis*. Oxford: Philological Society: pp. 1–32.

Godin, F., Vandersmissen, B., Jalalvand, A., De Neve W. and Van de Walle, R. (2014). Alleviating Manual Feature Engineering for Part-of-speech Tagging of Twitter Microposts Using Distributed Word Representations. In *Proceedings of the Workshop on Modern Machine Learning and Natural Language Processing*. Montreal, 2015, s.p.

Gysseling, M. (1977). *Corpus van Middelnederlandse teksten (tot en met het jaar 1300). Reeks I. Ambtelijke bescheiden*. 9 vols. Nijhoff, The Hague.

Gysseling, M. (1980-1987). *Corpus van Middelnederlandse teksten (tot en met het jaar 1300). Reeks II. Literaire teksten*. 6 vols. Nijhoff, The Hague.

Harris, Z. (1954). Distributional structure. *Word* 10(23): 146–62.

Hendrickx, I. and Marquilhas, R. (2011). From old texts to modern spellings: an experiment in automatic normalisation. *Journal for Language Technology and Computational Linguistics* 26(2): 65–76.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8): 1735–80.

Institute for Dutch Lexicography (1998). Cd-rom Middelnederlands. In *Woordenboek en Teksten*. Sdu, The Hague.

Jurafsky, D. and Martin, J. H. (2009). *An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, 2nd edn. New Jersey: Pearson Prentice Hall.

Kalchbrenner, N., Grefenstette, E. and Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Baltimore, MD: ACL, pp. 655–5.

Kestemont, M., Daelemans, W., and De Pauw, G. (2010). Weigh your words—memory-based lemmatization for middle Dutch. *Literary and Linguistic Computing*, 25(3): 287–301.

Knowles, G. and Mohd Don, Z. (2004). The notion of a ''lemma''. Headwords, roots and lexical sets. *International Journal of Corpus Linguistics*, 9(1): 69–81.

LeCun, Y., Boser, B., Denker, J. S., Howard, R. E., Habbard, W., Jackel, L. D. and Henderson, D. (1990). Handwritten Digit Recognition with a Back-propagation Network. In Touretzky, D. S. (ed), *Advances in Neural Information Processing Systems*. San Francisco: Morgan Kaufmann.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. *Nature* 521(7553): 436–44.

Levenshtein, V. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(9): 707–10.

Levy, O. and Golberg, Y. (2014). Linguistic Regularities in Sparse and Explicit Word Representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*. Baltimore: ACL, pp. 171–80.

Ljubešić, N., Erjavec, T. and Fišer, D. (2014). Standardizing tweets with character-level machine translation. In Gelbukh, A. (ed.), *Computational Linguistics and Intelligent Text Processing 8404*. Berlin: Springer, pp. 164–75.

Lyras, D. P., Sgarbas, K. N., and Fakotakis, N. D. (2008). Applying similarity measures for automatic lemmatization: a case-study for modern Greek and English. *International Journal on Artificial Intelligence Tools*, 17: 1043–64.

Manning, C. D., Prabhaker, P. and Schütze, H. (2008). *Introduction to Information Retrieval*. New York: Cambridge University Press.

Manning, C. D. (2015). Computational linguistics and deep learning. *Computational Linguistics* 41(4): 701–7.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean J. (2013). Distributed representations of words and phrases and their compositionality. *Neural Information Processing Systems*, 26: 3111–9.

Mitankin, P., Gerjikov, S. and Mihov, S. (2014). An approach to unsupervised historical text normalisation. In Antonacopoulos, A. and Schulz, K. (eds), In *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. New York: Association for Computing Machinery, pp. 29–34.

Nair, V. and Hinton, G. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In Fürnkranz J. and Joachims T. (eds), In *Proceedings of the 27th International Conference on Machine Learning*. Madison, WI: Omnipress, pp. 807–14.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: machine learning in python. *Journal of Machine Learning Research*, 12: 2825–30.

Piotrowski, M. (2012). *Natural Language Processing for Historical Texts*. California: Morgan & Claypool Publishers.

Reynaert, M., Hendrickx, I. and Marquilhas, R. (2012). Historical spelling normalization. A comparison of two statistical methods: TICCL and VARD2. In Mambrini, F., Passarotti, M. and Sporleder, C. (eds), In *Proceedings of the Second Workshop on Annotation of Corpora for Research in the Humanities*. Lisbon, Portugal: Edições Colibri, pp. 87–98.

Scherrer, Y. and Erjavec, T. (2013). Modernizing historical Slovene words with character-based SMT. In: Piskorski, J., Pivovarova, L., Tanev, H. and Yangarber, R. (eds), In *Proceedings of the 4th Biennial Workshop on Balto-Slavic Natural Language Processing*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 58–62.

Schulz, S., De Pauw, G., De Clerq, O., Desmet, B., Hoste, V., Daelemans, W. and Macken, L. (2016). Multi-modular text normalization of dutch user-generated content. *ACM Transactions on Intelligent Systems and Technology*, forthcoming, NewYork, NY, USA: ACM, 7(4):1–22.

Souvay, G. and Pierrel, J.-M. (2009). LGeRM. Lemmatisation des Mots en Moyen Français. *Traitement Automatique des Langues*, 50(2): 149–72.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15: 1929–58.

Toutanova, K., Klein, D., Manning, C. and Singer, Y. (2003). Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, Vol. 1. Stroudsburg: Association for Computational Linguistics, pp. 173–80.

Van Dalen-Oskam, K. and Van Zundert, J. (2007). Delta for Middle Dutch – author and copyist distinction in Walewein. *Literary and Linguistic Computing*, 22(3): 345–62.

Van der Maaten, L. and Hinton, G. (2008). Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 92579–605.

Van der Voort van der Kleij, J. (2005). Reverse Lemmatization of the Dictionary of Middle Dutch (1885-1929) Using Pattern Matching. In Kiefer, F., Kiss, G., and Pajzs, J. (eds), *Papers in Computational Lexicography*. Budapest: Hungarian Academy of Sciences, pp. 203–10.

Van Halteren, H. and Rem, M. (2013). Dealing with orthographic variation in a tagger-lemmatizer for fourteenth century Dutch charters. *Language Resources and Evaluation*, 47(4): 1233–59.

Van Kerckvoorde, C. (1992). *An Introduction to Middle Dutch*. The Hague: Mouton de Gruyter.

Zavrel, J. and Daelemans, W. (1999). Recent Advances in Memory-Based Part-of-Speech Tagging. In *Actas del VI Simposio Internacional de Comunicacion Social*. Centro de lingüística Aplicada, Santiago de Cuba, pp. 590–7.

Zeiler, M.D. (2012). Adadelta: an adaptive learning rate method. ArXiv, 1212.5701v1.

Zhang, X., Zhao, J. and LeCun, Y. (2015). Character-level convolutional networks for text classification. *Neural Information Processing Systems*, 28: s.p.

Notes

1 An example of over-eager expansion might for instance create confusion in the case of minimal pairs: *hoer* and *haer* are both allowed under the lemma 'haar' ('her'), but applying the same vowel transition ('oe' > 'ae') to the token *maer* under the lemma 'maar' ('but') would yield the token *moer*, which is primarily attested under lemmas such as 'muur' ('wall'), 'moeder' ('mother'), or 'modder' ('mud').

2 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 clear beneficial effect, nor did the introduction of stacked convolutions.

3 https://code.google.com/archive/p/word2vec/.

4 https://radimrehurek.com/gensim/.

5 https://github.com/mikekestemont/tag.

6 http://keras.io/.

7 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).

8 Online at: http://gtb.inl.nl/ (last accessed 6 November 2014).

9 See: https://www.nederlab.nl/home (last accessed 6 November 2014).

10 Based on personal communication with the authors.