frontiers in Research Metrics and Analytics

ORIGINAL RESEARCH
published: 13 July 2018
doi: 10.3389/frma.2018.00021

Deep Reference Mining From Scholarly Literature in the Arts and Humanities

Danny Rodrigues Alves[1], *Giovanni Colavizza*[1,2]* *and Frédéric Kaplan*[1]

[1] *Digital Humanities Laboratory, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland,* [2] *The Alan Turing Institute, London, United Kingdom*

We consider the task of reference mining: the detection, extraction and classification of references within the full text of scholarly publications. Reference mining brings forward specific challenges, such as the need to capture the morphology of highly abbreviated words and the dependence among the elements of a reference, both following codified reference styles. This task is particularly difficult, and little explored, with respect to the literature in the arts and humanities, where references are mostly given in footnotes. We apply a deep learning architecture for reference mining from the full text of scholarly publications. We explore and discuss three architectural components: word and character-level word embeddings, different prediction layers (Softmax and Conditional Random Fields) and multi-task over single-task learning. Our best model uses both pre-trained word embeddings and characters embeddings, and a BiLSTM-CRF architecture. We test our solution on a dataset of annotated references from the historiography on Venice and, using a linear-chain CRF classifier as a baseline, we show that this deep learning architecture improves by a considerable margin. Furthermore, multi-task learning performs almost on par with a single-task approach. We thus confirm that there are important gains to be had by adopting deep learning for the task of reference mining.

Keywords: reference mining, natural language processing, conditional random fields, deep learning, recurrent neural networks, bibliometrics, arts and humanities, history

OPEN ACCESS

Edited by:
Philipp Mayr,
Leibniz Institut für Sozialwissenschaften (GESIS), Germany

Reviewed by:
Lin Zhang,
KU Leuven, Belgium
Animesh Prasad,
National University of Singapore, Singapore

***Correspondence:**
Giovanni Colavizza
gcolavizza@turing.ac.uk

Received: *26 February 2018*
Accepted: *12 June 2018*
Published: *13 July 2018*

Citation:
Rodrigues Alves D, Colavizza G and Kaplan F (2018) Deep Reference Mining From Scholarly Literature in the Arts and Humanities. Front. Res. Metr. Anal. 3:21. doi: 10.3389/frma.2018.00021

1. INTRODUCTION

Reference mining (or parsing) is a Natural Language Processing (NLP) task focused on the detection, extraction and classification of bibliographic references and their constituent components from scholarly literature. It is a necessary step toward the creation of relational citation data, a task commonly performed in view of building citation indexes (Garfield, 1979). Compared to other NLP tasks, reference mining stands in the broader category of sequence labeling problems, which includes among others Part Of Speech (POS) tagging and Named Entity Recognition (NER). Traditional machine learning methods for sequence labeling tasks, including Hidden Markov Models (HMM) and (linear-chain) Conditional Random Fields (CRF), depend on a considerable amount of external knowledge in the form of hand-engineered features and task-specific resources like gazetteers and lexicons. However, these resources are costly to produce and are not easy to adapt to variations of a given task, especially so because they require expert human knowledge.

In recent years deep learning, or the use of deep neural network models trained on large amounts of data, has been changing the whole field of machine learning, considerably improving on most tasks (LeCun et al., 2015; Schmidhuber, 2015). Yet the openly available non-commercial tools for reference parsing still mostly rely on previous-generation techniques (Tkaczyk et al., 2018). Quite consequently, this paper contribution is to take a deep learning approach by applying current state-of-the-art architectures for sequence labeling to the specific task of reference mining.

A further motivation for the use of deep learning comes from the scholarly domain which we interest ourselves into: the arts and humanities. Where reference mining applications targeting most scientific publications need to focus on relatively uniform reference lists, scholarly publications in the arts and humanities are more varied in this respect (Sula and Miller, 2014; Colavizza et al., 2017). A set of challenges must be considered: references are made to (at least) both primary and secondary sources, and primary sources are by definition more varied than secondary ones. References can happen anywhere in the text of a publication, especially so in footnotes, and not just in reference lists. In this case, references are often given once in full form and abbreviated thereafter. It must also be noted that it is not customary to cite primary sources in reference lists. Lastly, the variety of publication venues, languages, scholarly communities in the arts and humanities is broader, making reference practices and styles less uniform. For these and other reasons, the scholarly literature from the arts and humanities is still not well indexed (Mongeon and Paul-Hus, 2016) nor studied (Ardanuy, 2013) using citation data.

We consider and compare several components of a recurrent neural network architecture for reference mining. In particular, we experiment with different approaches in the input layer, by considering both character and word-level embeddings. We also test a Conditional Random Field instead of the canonical Softmax prediction layer. Finally, we experiment with multi-task learning in order to test whether the learning our best model does is shared across different tasks. All models are built around a single BiLSTM layer, a proven key ingredient in a variety of sequence labeling tasks. We make two implementations available, one using *Keras* (Chollet et al., 2015) (relying on *TensorFlow* as back-end), and another directly in *TensorFlow* (Dean et al., 2015), in order to facilitate the reuse of results and further experimentation.[1] Our experiments are based on a published dataset of annotated references from a corpus of publications on the history of Venice (Colavizza and Romanello, 2017).

This paper is organized as follows. We briefly discuss previous work in section 2, then introduce the task of reference mining and the dataset in section 3. In the same section, a CRF baseline model is discussed. Section 4 describes the general architecture we propose and test in all its components. Section 5 contains our results, as well as the details of the best architecture and model configuration, with its validation. We finally conclude in section 6.

[1] Keras version `2.1.1` and TensorFlow version `1.4.0`.

2. RELATED WORK

In a recent survey and evaluation, several non-commercial reference parsing tools, Tkaczyk et al. (2018) found that the best three performing ones all use a CRF approach: GROBID (Lopez, 2009), CERMINE (Tkaczyk et al., 2015) and ParsCit (Councill et al., 2008). All three benefit from task-specific tuning using extra annotated data, with GROBID showing the best off-the-shelf results. Indeed seven out of the total of thirteen surveyed tools use a CRF approach, while the rest mainly adopt regular expressions. To date, all published non-commercial reference mining tools rely on these or rule-based methods[2]. Heckmann et al. (2016) attempted to tackle some of the main challenges to be found in humanities literature, namely: "multilingual citation entries, lack of data redundancy, inconsistencies, and noise from OCR input." Their knowledge-based approach relying on Markov logic networks was found to substantially outperform a CRF baseline. A useful insight for the task at hand also came from Körner et al. (2017), where a CRF is used to classify lines of text containing references in advance to considering their constituent tokens. The proposed method, RefExt, outperformed several above-mentioned state-of-the-art solutions.

As deep learning started to gain momentum in recent years, attention has been given to the use of unsupervised feature extraction techniques in a variety of NLP tasks, mainly in the form of word embeddings, which lead to state-of-the-art results when used to augment, rather than replace, hand-crafted features (Collobert et al., 2011). More recent work on sequence labeling tasks relies instead on deep learning techniques such as convolutional or recurrent neural network models (CNNs LeCun et al., 1989 and RNNs Rumelhart, 1986, respectively), without the need for any hand-crafted features (Kim, 2014; Huang et al., 2015; Zhang et al., 2015; Chiu and Nichols, 2016; Lample et al., 2016; Ma and Hovy, 2016; Yang et al., 2016; Strubell et al., 2017). RNNs in particular, typically rely on a neural network architecture built using one or more Bidirectional Long-Short Term Memory (BiLSTM) layers, as this type of neural cell provides for variable-length memory allowing the model to capture relationships within sequences of proximal words. Such architectures have achieved state-of-the-art performance for both POS and NER tasks on popular datasets (Reimers and Gurevych, 2017b). Current state-of-the-art architectures for sequence labeling include the use of a CRF prediction layer (Huang et al., 2015) and the use of character-level word embeddings to complement word embeddings, trained either with CNNs (Ma and Hovy, 2016) or BiLSTM RNNs (Lample et al., 2016). Character-level word embeddings have indeed been shown to perform well on a variety of NLP tasks (Dos Santos and Gatti de Bayser, 2014; Kim et al., 2015; Zhang et al., 2015). Attention mechanisms have also been proposed for the same tasks (Rei et al., 2016; Shen and Lee, 2016). In this paper we will apply, tune and compare two architectures (Lample et al., 2016; Ma and Hovy, 2016) to the specific task of reference mining.

[2] An exception is Neural ParsCit https://github.com/opensourceware/Neural-ParsCit, a yet unpublished adaptation of the architecture proposed in Lample et al. (2016) for the task of reference parsing.

3. TASK DEFINITION, DATASET, AND BASELINE MODEL

A bibliographic reference is a contiguous sequence of text where all the necessary information on a citation to any primary or secondary source is contained. Most typically, previous scholarship and primary evidence such as archival documents or works of art and literature can be cited in arts and humanities scholarly literature. What constitutes necessary information is relative: usually, the first citation to a source contains all information necessary for its unambiguous identification, a substantial part of this same information can be dropped or abbreviated in subsequent citations to the same source within the same publication.

A reference is usually composed of several information components, such as the *author*, *title* or *publisher* of a cited publication, encoded in a systematic way following some editorial guidelines specific to the venue and time of publication (e.g., using double quotes or italics for the title). An example of a reference is

```
G. Ostrogorsky, History of the Byzantine
 State, Rutgers University Press, 1986.
```

This reference has four components: the author's name, title, publisher and year of publication. In this example, the components are separated by a comma and the author's name is abbreviated using initials followed by a dot. The same reference might be given elsewhere following a different *reference style*, defined as: "a specific combination of elements in a reference, such as author and title, encoded in a predefined way" (Colavizza et al., 2017, p. 4). For example, it might be given as *"Ostrogorsky, G. (1986). History of the Byzantine State, Rutgers University Press,"* where the combination of elements as well as their encoding has changed.

If we consider a text as a stream of tokens organized into lines (sequences of characters separated by white space), the goal of reference mining is to:

- **Detect** that a token is part of a reference. A token part of a reference can be anywhere, most typically in footnotes.
- **Extract** a reference, i.e., individuate its first and last tokens (begin-end).
- Optionally **classify** a full reference and its constituent components: in our case, a reference might be to a primary or secondary source (this information is useful for further processing steps such reference disambiguation to establish citations, as this step typically relies on existing catalogs look-up), and each reference might contain a variety of components (author, title, archive and record group, etc.).

In this article we consider all three actions, and use the processing unit (sequence) of the line of text. Our motivation to use sequences as lines of text is given by the need to parse the full-text of publications in order to capture footnotes, and the irregular positioning of references therein. The extraction and detection of references is done using *begin-end* token classification to mark, respectively, the beginning and end of a reference within a stream of tokens. With respect to classification, two annotation schemes (tags) are considered: *specific* and *generic*. A specific annotation identifies a component of a reference, such as author or title. A generic annotation refers to the typology of the cited source, distinguishing among primary sources, books and other contributions such as journal articles. More in detail, given the plain text of a publication, our goal is to assign the most likely tag to each token (token by token classification). We define three tasks as follows:

- Task 1: *reference components*. Each token is classified using a taxonomy of 27 specific tags, unevenly represented in the annotated dataset, which include a non-reference tag. The taxonomy is given and discussed in Colavizza and Romanello (2017) and in the accompanying code repository. The reason to have 27 tags is mainly the presence of references to archival documentation, which requires a classification on its own.
- Task 2: *reference typology*. Each token is classified according to the generic annotation scheme. As mentioned above, tags include: *primary* sources (e.g., archival documents), *secondary* sources (books), and *meta*-sources, i.e., publications contained within other publications (e.g., journal articles). Furthermore, *b*egin, *e*nd and *i*n reference tags are prepended to a generic tag, and an *o*ut of reference tag is used too. For example, *b-secondary* marks the first token of a reference to a book-form publication.
- Task 3: *reference span*. Each token is classified simply using the *b*egin, *e*nd, *i*n and *o*ut schema. For example, *e-r* marks the last token of a reference.

The different tasks are illustrated in **Figure 1**, using the example given above.

3.1. Dataset

We use a published dataset containing more than 40,000 annotated references from a corpus of publications on the historiography on Venice. The corpus includes books and journal articles published from the 19th century to 2014. It considers publications in a variety of languages: mostly Italian, followed by English, French, German, Spanish and Latin. The annotated corpus includes references taken from reference lists and footnotes, as a consequence, a considerable variety of referencing styles and referred sources is present. Annotated references for every publication are a representative sample of the total amount. For reasons of copyright, this dataset does not contain the full text of publications, but only the text lines where a reference (or part of it) appears; therefore some lines of text include out-of-reference tokens, preceding or following a reference (these tokens are important to learn to assign begin-end tags). Full details, including corpus acquisition and annotation sampling strategy and procedure, are given in Colavizza and Romanello (2017) [3].

A new export of this dataset is used here, prepared as follows. Initially, every publication with annotated references is randomly

[3]The dataset and accompanying code are available in, respectively GitHub: https://github.com/dhlab-epfl/LinkedBooksReferenceParsing and Zenodo: http://doi.org/10.5281/zenodo.579679

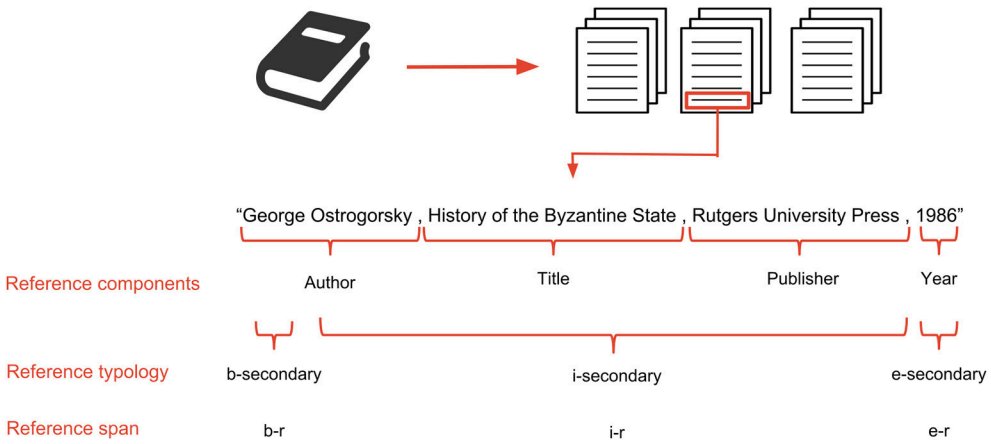

FIGURE 1 | Example of a reference annotated according to tasks 1–3. Task 1 covers reference components, task 2 considers the span of a reference plus its general typology, task 3 instead only considers the span of a reference. We use a clear-cut example for illustration purposes, yet in fact most references given in footnotes have text before and afterwards, often on the same line.

allocated in a train, test or validation set, with an 80/10/10 split respectively[4]. The number of references in each set does not precisely follow the same proportion, as different publications have a varied amount of annotated references. Nevertheless, a publication-level split is important in order to reduce reference style data snooping. Next, the annotated lines of plain text for every publication are considered independently and split into tokens using the NLTK word-punkt tokenizer (Bird et al., 2009), thus considering several punctuation symbols as a separate token. The dataset is at this point composed of a set of lines of text, which will be parsed independently, each including at least part of a reference, split into tokens and associated with the annotation schemes of the different tasks. By all means, a reference can be part of multiple lines of text. The choice of considering lines of text independently reduces the dependency window that the classification method can rely upon, and is to be considered a limitation of this study.

This reprocessed dataset is made available using the CoNLL convention: each line in a file (test, train and validation) corresponds to a token in a sequence (original line of text), and sequences are separated by a blank line. Each token line contains the token surface form followed by the corresponding tags for each task, separated by a white space. To encode the relative position of a token in a reference, the IOBE convention is used, where `i-label` stands for a token inside a reference (not begin or end), `o` outside, `b-label` if the token is the first of a reference and `e-label` the last. The IOBE is a variant of the more common IOB scheme. Using a more expressive tagging scheme like IOBE has been shown to marginally improve model performance (Ratinov and Roth, 2009; Dai et al., 2015) and ease the retrieval of references spanning across several lines.

[4]Sometimes in the literature what we refer as test dataset, to assess the results of training, is named development dataset, and the validation dataset, what we use at the end to test for generalization, is named test dataset. We will use what we call test dataset for development and what we call validation dataset for final testing.

Our example *"G. Ostrogorsky, History of the Byzantine State, Rutgers University Press, 1986,"* assuming it spans a single line (sequence), is encoded as:

```
G author b-secondary b-r
. author i-secondary i-r
Ostrogorsky author i-secondary i-r
, author i-secondary i-r
History title i-secondary i-r
of title i-secondary i-r
the title i-secondary i-r
Byzantine title i-secondary i-r
State title i-secondary i-r
, title i-secondary i-r
Rutgers publisher i-secondary i-r
University publisher i-secondary i-r
Press publisher i-secondary i-r
, publisher i-secondary i-r
1986 year e-secondary e-r
. year e-secondary e-r
```

3.2. CRF Baseline

We train and test a Conditional Random Field (Lafferty et al., 2001) baseline using the same dataset. The CRF classifier is trained over a rich set of hand-crafted features considering a size-two bi-directional window: the features for a token at position t in a sequence include features extracted for the two preceding and two following tokens too, that is positions $t-2$, $t-1$, $t+1$, $t+2$, following previous work where the specificities of applying CRF to the humanities are amply discussed (Colavizza and Romanello, 2017). This model is trained with Stochastic Gradient Descent applying both L1 and L2 regularization, using the *CRFSuite* package (Okazaki, 2007)[5]. The code and training details are given in this work's accompanying repository. The best cross-validated

[5]We used the *CRFsuite* implementation from sklearn-crfsuite, version 0.3.6 available at https://github.com/TeamHG-Memex/sklearn-crfsuite.

configuration of this model yields the following F1 validation scores for each task:[6]

Task 1 gives an F1 score of **82.63%**
(precision 82.88%, recall 82.76%).

Task 2 gives an F1 score of **71.04%**
(precision 71.32%, recall 71.1%).

Task 3 gives an F1 score of **92.50%**
(precision 92.64%, recall 92.41%).

4. MODEL

We consider a recurrent architecture organized into three layers: input (word representations), inner and prediction, following the best performing models for sequence labeling tasks (Lample et al., 2016; Ma and Hovy, 2016). The network firstly receives a sequence of (one-hot encoded) words $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, ..., \mathbf{w}^{(n)}$ as input and transforms it into a sequence of dense vectors $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(n)}$, using a combination of word and character-level word embeddings. Secondly, word representations are passed to a bidirectional LSTM composed of two layers: a forward layer where the word representations are processed starting with input representation $\mathbf{x}^{(1)}$ to $\mathbf{x}^{(n)}$, and a backward layer from $\mathbf{x}^{(n)}$ to $\mathbf{x}^{(1)}$. The outputs of these two layers are concatenated and used in the prediction layer, which outputs a sequence of predictions $\hat{\mathbf{y}}^{(1)}, \hat{\mathbf{y}}^{(2)}, ..., \hat{\mathbf{y}}^{(n)}$ for each initial input word $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, ..., \mathbf{w}^{(n)}$. We remark that we refer to words from now on, to adopt the most common terminology in the literature, where in practice generic tokens are considered.

4.1. Input Layer

The input layer combines word and character-level word embeddings for each input token in a sequence, in order to create a word representation.

4.1.1. Word Embeddings

Word embeddings are a common staple of sequence classification tasks, and are often trained over large corpora like *Wikipedia*[7] or *Reuters*[8], in order to embed richer information than just using task-specific data. Yet considering the dataset used in this project, publicly available embeddings will likely not help. Instead, word embeddings were pre-trained using Word2Vec (Mikolov et al., 2013a,b) on the full contents of all the publications from which our references were extracted. We used the Gensim Word2vec implementation (Řehůřek and Sojka, 2010), a window of 5 words and the skip-gram model. The vectors are trained for all words appearing at least five times in the dataset, while less frequent words have been regrouped under an unknown token `$UNK$` and all digits have been merged into a `$NUM$` token. We tested word embeddings with a dimensionality of 100 and 300. Word embeddings can be randomly initialized and trained with the model, pre-trained and kept fix, or pre-trained and further trained with the model. The pre-trained word embedding vocabulary comprises 727,902 words, of which 51,569 are actually used in the published dataset.

4.1.2. Character-Level Word Embeddings

Tokens part of references contain relevant information at the orthographic and morphological levels, such as prefixes and suffixes and the use of punctuation or abbreviations. Given the relative small amount of annotated data at hand, it is likely the case that these features will not be learned at the word level in a satisfactory way. Conversely, character-level word embeddings can help into learning task-specific features at this level, with fewer examples. These features have in particular found useful application to deal with out-of-vocabulary words and morphologically rich languages (Dos Santos and Zadrozny, 2014). Furthermore, character-level word embeddings can help reduce the impact of OCR errors and help deal with rare words. Character-level word embeddings are a representation of a word from the compounded representation of sequences of characters the word is composed of. They can be learned either via CNNs or BiLSTMs. The character-level word embeddings are trained by first considering randomly initialized character embeddings. In the CNN case, we then feed them to a single 1d convolution layer followed by a max pool layer, using a filter stride of 1 and various widths. Alternatively, we use a BiLSTM and concatenate its outputs.

4.1.3. Word Representation Architecture

Figure 2 describes the architecture to build a word representation input made of the concatenation of a word embedding and a character-level word embedding trained with a BiLSTM. The word embeddings consist of a lookup to the precomputed Word2Vec embeddings, or randomly initialized ones, and the character-level word embeddings are computed through additional neural network layers as described above. The final word representation is a concatenation of its word embedding and character-level word embedding.

To prevent the model from too strongly depending on word and character-level word embeddings, dropout layers are added after the BiLSTM or CNN layers (for character-level word embeddings) and after word and character-level word embeddings are concatenated. More generally, as sketched in **Figure 3**, dropout layers are applied on several components of the final model. Dropout is a regularization technique where randomly selected neurons are turned off during training. It helps to prevent overfitting and to avoid the model to depend to heavily on individual neurons (Srivastava et al., 2014).

4.2. Inner Layer

Long-Short Term Memory cells (LSTM) are part of the Recurrent Neural Networks (RNN) family, designed to account for flexibly long memory dependences (Hochreiter and Schmidhuber, 1997). LSTMs overcome in part the limitations of vanilla RNNs, such as the practically short memory dependence and the tendency to suffer from vanishing or exploding gradients (Bengio et al., 1994).

[6]Here c1 and c2 refer to the model coefficients for L1 and L2 regularization, respectively. For task 1 cross validated parameters were set at `c1=1.3099` and `c2=0.0773`; `c1=0.9298` and `c2=0.0229` for task 2; `c1=2.1334` and `c2=0.0142` for task 3.

[7]https://nlp.stanford.edu/projects/glove/

[8]https://www.cs.umb.edu/~smimarog/textmining/datasets/

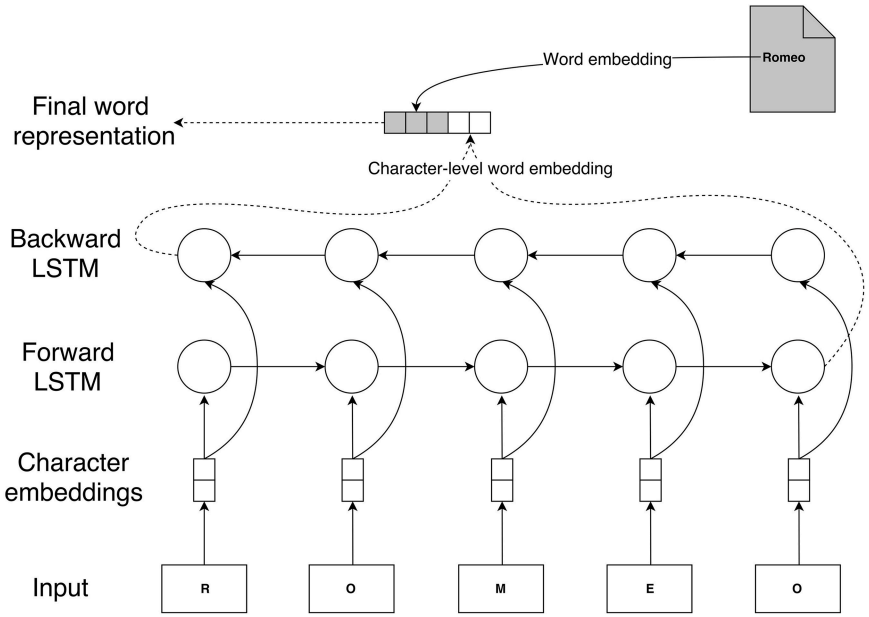

FIGURE 2 | The word representation architecture using both pre-trained word embeddings and BiLSTM character-level word embeddings, used in the example to construct the representation of the word "Romeo." Rectangles are used for inputs, sequences of squares for vectors, rounds for neuron cells and dashed lines for dropout connections.

An RNN cell with sigmoid activation and softmax prediction can be described as follows:

$$\mathbf{h}^{(t)} = \sigma(\mathbf{b} + \mathbf{W}\mathbf{x}^{(t)} + \mathbf{U}\mathbf{h}^{(t-1)})$$
$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{c} + \mathbf{V}\mathbf{h}^{(t)})$$

where $\mathbf{x}^{(t)}$ is the input word representation in position t of the current sequence, $\mathbf{h}^{(t)}$ represents the hidden state at the same position, $\mathbf{b}$ and $\mathbf{c}$ are bias vectors and $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are parameter matrices to be learned. An LSTM instead introduces three gates to the RNN configuration: an input gate $\mathbf{i}$, a forget gate $\mathbf{f}$, and an output gate $\mathbf{o}$, in order to provide the cell with a means to retain information on previous states more effectively. An LSTM cell with softmax prediction, as implemented in Keras, can be described as follows:

$$\mathbf{i}^{(t)} = \sigma(\mathbf{b}^i + \mathbf{W}^i\mathbf{x}^{(t)} + \mathbf{U}^i\mathbf{h}^{(t-1)})$$
$$\mathbf{f}^{(t)} = \sigma(\mathbf{b}^f + \mathbf{W}^f\mathbf{x}^{(t)} + \mathbf{U}^f\mathbf{h}^{(t-1)})$$
$$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot tanh(\mathbf{b}^c + \mathbf{W}^c\mathbf{x}^{(t)} + \mathbf{U}^c\mathbf{h}^{(t-1)})$$
$$\mathbf{o}^{(t)} = \sigma(\mathbf{b}^o + \mathbf{W}^o\mathbf{x}^{(t)} + \mathbf{U}^o\mathbf{h}^{(t-1)})$$
$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \odot tanh(\mathbf{c}^{(t)})$$
$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{c} + \mathbf{V}\mathbf{h}^{(t)})$$

where σ is the element-wise hard-sigmoid function and $\odot$ is the element-wise product. As before, $\mathbf{x}^{(t)}$ is the input word representation in position t of the current sequence and $\mathbf{h}^{(t)}$ represents the hidden state at the same position. $\mathbf{x}^{(t)}$ represent the current cell state, as a function of the forget gate applied to the previous step cell state, and the input gate applied to a non-linear transformation (hyperbolic tangent in this case) of a vanilla RNN internal state. The final hidden state is then given by a product of the output gate with a further non-linear transformation of the cell state. The different bias vectors $\mathbf{b}$ and $\mathbf{c}$ and matrices $\mathbf{W}$, $\mathbf{U}$, and $\mathbf{V}$ are all learned parameters. A BiLSTM is made of two LSTM layers, one being fed the input in the original order, the other in reversed order. The final hidden layer is the concatenation of the two: $\mathbf{h}^{(t)} = \left[\overrightarrow{\mathbf{h}}^{(t)}; \overleftarrow{\mathbf{h}}^{(t)}\right]$.

Since inputs are processed in temporal order, a possible shortcoming of LSTMs is their inability to make use of subsequent context (Hochreiter et al., 2001). Nevertheless, two LSTMs can be used to process the input in opposite directions, and their results concatenated. This solution, referred to as a Bidirectional LSTM (Schuster and Paliwal, 1997), has shown notable results in a variety of NLP tasks (Graves and Schmidhuber, 2005; Graves et al., 2013; Huang et al., 2015).

4.3. Prediction Layer

A widely adopted prediction layer for multi-class sequence labeling tasks relies on the softmax function. Assuming $\mathbf{z}$ to be a vector of unnormalized log probabilities from a linear layer, we have:

$$\mathbf{z} = \mathbf{c} + \mathbf{V}\mathbf{h}$$
$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1} e^{z_j}}$$

The softmax takes every classification decision independently for every input word, yet sequence labeling tasks seldom present no dependence between proximal tags. For example in our task 3,

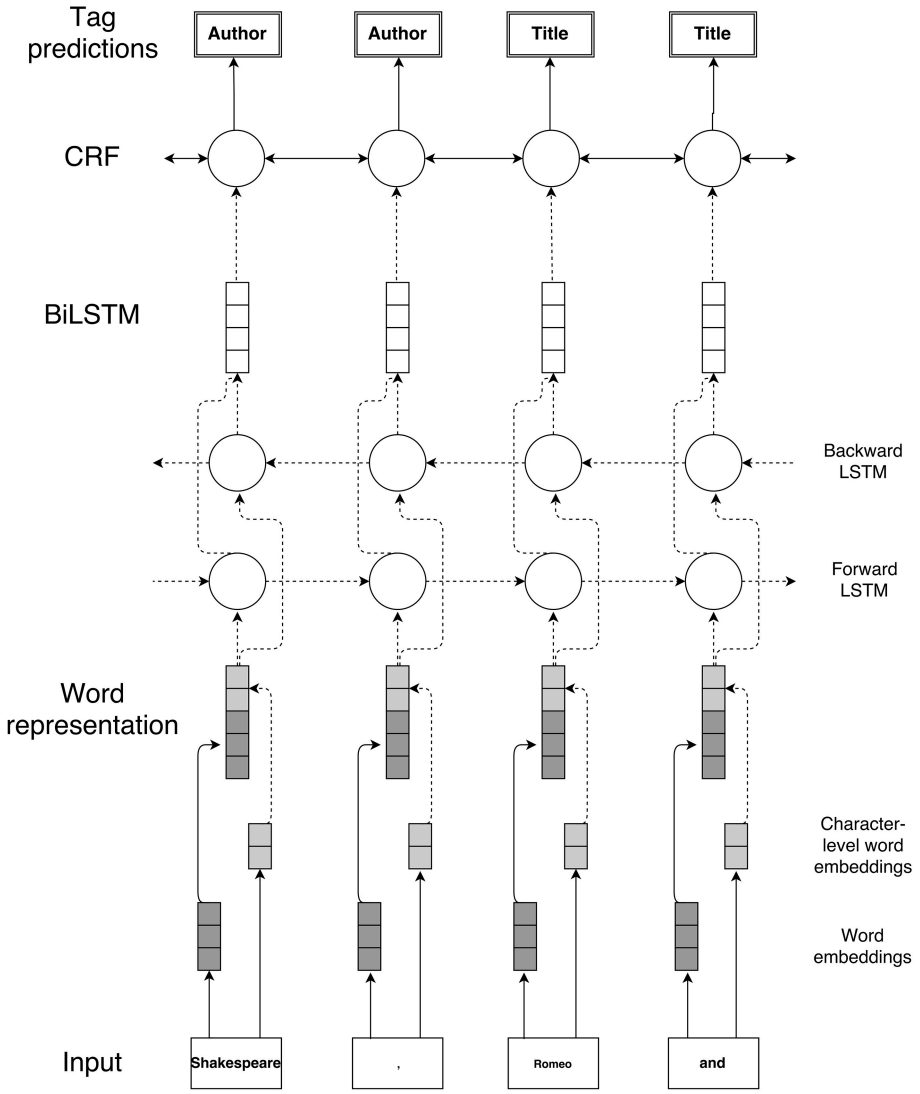

FIGURE 3 | Sketch of the model architecture for a part of the sequence `W. Shakespeare, Romeo and Juliet, Oxford University Press, London, 1914`. Rectangles are used for inputs, double rectangles for outputs, sequences of squares for vectors, rounds for neuron cells and dashed lines for dropout connections.

the tag *i-r* can never be followed by the tag *b-r*. More generally, reference styles entail that few recurring sequences of tags should be learned and predicted.

Using a CRF layer for predictions enables the model to perform classification decisions maximizing the (log) likelihood over the whole sequence of predictions (Lafferty et al., 2001; Sutton and McCallum, 2011). In the context of sequence labeling tasks, a linear-chain CRF is trained to predict a sequence $\mathbf{y} = (\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, ..., \mathbf{y}^{(n)})$ of known tags for a sequence input representation $\mathbf{X} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(n)})$. A linear-chain CRF in this setting uses a combination of unary features for state-observation pairs, and of binary features for each transition (Huang et al., 2015). We consider $\mathbf{Z}$ to be the $n \times k$ matrix of unnormalized scores from the inner BiLSTM layer, where n is the number of words in the sequence, k the number of possible tags (e.g., 27 for task 1). We then consider a square matrix $\mathbf{A}$ of new parameters, such that $\mathbf{A}_{i,j}$ represents the probability of transitioning from tag i to j in a sequence of predictions. In HMM terminology, $\mathbf{Z}$ is referred to as the emission matrix and $\mathbf{A}$ as the transition matrix. The score for the given sequence of tag assignments $\mathbf{y}$ is then calculated, and its probability over the space of possible tag prediction sequences $\hat{\mathbf{Y}}_\mathbf{X}$ taken with softmax:

$$score(\mathbf{X}, \mathbf{y}) = \sum_{i=0}^{n} \mathbf{A}_{y_i, y_{i+1}} + \sum_{i=1}^{n} \mathbf{Z}_{i, y_i}$$

$$\mathbb{P}(\mathbf{y}|\mathbf{X}) = \frac{e^{score(\mathbf{x}, \mathbf{y})}}{\sum_{\hat{\mathbf{y}}' \in \hat{\mathbf{Y}}_\mathbf{X}} e^{score(\mathbf{x}, \hat{\mathbf{y}}')}}$$

During training, the score of the correct tag sequence is maximized using dynamic programming. The best (maximum a posteriori) tag sequence assignment for a new input sequence can be computed using the Viterbi algorithm.

4.4. Multi-Task Learning

Multi-task learning has been considered to train and predict the three tasks at once, relying on the same architecture. This technique has proved useful to reach results comparable to single-task architectures, at a great reduced computational cost obtained by sharing most of the trained parameters across multiple tasks (Ruder, 2017). In some instances, multi-task learning can even improve single-task results. With respect to reference classification, we expect the inner layers of the network to learn quite similarly across different tasks, therefore it makes sense to attempt a multi-task approach.

Our multi-task architecture is identical to a single-task one up to the hidden layer outputs included. Afterwards, a separate prediction layer is created for each task. The loss function to be optimized is the sum of the losses of each task layer. Considering a softmax prediction layer, and the output $\mathbf{h}^{(t)}$ of the hidden layer at step t, we have:

$$\hat{\mathbf{y}}_1^{(t)} = softmax\left(\mathbf{c}_1 + \mathbf{V}_1\mathbf{h}_1^{(t)}\right)$$
$$\hat{\mathbf{y}}_2^{(t)} = softmax\left(\mathbf{c}_2 + \mathbf{V}_2\mathbf{h}_2^{(t)}\right)$$
$$\hat{\mathbf{y}}_3^{(t)} = softmax\left(\mathbf{c}_3 + \mathbf{V}_3\mathbf{h}_3^{(t)}\right)$$

The model thus has few extra parameters to learn, namely bias vectors $\mathbf{c}$ and matrices $\mathbf{V}$.

5. EXPERIMENTS

In this section we detail the experiments conducted on variants of the neural network architecture under consideration, as well as the fine tuning of our best final model (5.1). We then validate and discuss the results (5.2). For both model selection and fine tuning, task 1 has been considered. Furthermore we used early epoch stopping on the F1 test score with a waiting window of 5 epochs without improvements, and a maximum number of 25 epochs. Both code and dataset are released publicly (see data availability statement).

5.1. Architecture

Three main variants of the architecture were considered in turn: (1) word embeddings (presence or absence, pre-trained or not, further trained or not); (2) character-level word embeddings (presence or absence, BiLSTM or CNN), (3) prediction layer (softmax or CRF). The best components were selected based on the F1 score on testing data[9]. Results reported in **Table 1** indicate that the best architecture uses pre-trained word embeddings which are further trained on the specific task, BiLSMT character-level word embeddings and a CRF prediction layer. The experiments on the architecture of the model always used the configuration given in **Table 2**, following Lample et al. (2016).

Word embeddings can be integrated in a model architecture in three ways:

1. *Train*: Word embeddings initialized at random and trained. This configuration is also known in the literature as random initialization.

[9]The F1 score is the harmonic mean of precision and recall calculated considering every classification action independently.

TABLE 1 | Results of the experiments on the model architecture.

Word embeddings	Character features	Output	F1 score
Txrain word2vec	BiLSTM	crf	88.36
Train word2vec		crf	87.36
Train word2vec	CNN	crf	87.29
Word2vec	BiLSTM	crf	86.85
Word2vec	CNN	crf	86.16
Train word2vec	BiLSTM	softmax	86.12
Train	BiLSTM	crf	86.10
Word2vec	BiLSTM	softmax	85.96
Train		crf	85.88
Train	CNN	crf	85.56
Train word2vec	CNN	Softmax	85.47
Train word2vec		Softmax	85.41
Word2vec		crf	84.95
word2vec	CNN	Softmax	84.45
Train	BiLSTM	softmax	83.99
Word2vec		Softmax	83.91
Train	CNN	Softmax	83.61
	BiLSTM	crf	83.06
Train		Softmax	83.06
	BiLSTM	Softmax	82.05
	CNN	crf	78.23
	CNN	Softmax	75.28

Configurations are sorted according to the F1 testing score, in decreasing order. A blank cell indicates that the specific component was not included.

TABLE 2 | Configuration for the experiments on model architecture.

Layer	Parameter	Value
Word embeddings	Dimensionality	300
	Min word frequency	5
Character-level word embeddings	Embedding dimensionality	100
	BiLSTM dimensionality	100
BiLSTM	Dimensionality	64
CRF	Metric	Viterbi
Early stopping	Max waiting	5
	Max number of epochs	25
Model	Optimizer—CRF prediction	RMSprop
	Optimizer—Softmax prediction	Adam
	Dropout	0.5
	Learning rate	0.001
	Decay	0
	Batch size	50

TABLE 3 | Results of the fine-tuning of the best multi-task architecture, over the batch size, the dimensionality of the inner BiLSTM and the rate of dropout.

Batch size	Dropout	BiLSTM size	Training Task I	Validation Task I	Training Task II	Validation Task II	Training Task III	Validation Task III
100	0.5	200	0.8597	0.8613	0.8010	0.7990	0.9396	0.9316
30	0.5	200	**0.8840**	**0.8952**	**0.8236**	0.8077	0.9073	0.9053
70	0.5	200	0.8804	0.8885	0.8166	0.8005	0.9455	**0.9391**
100	0.7	200	0.8693	0.8837	0.8044	0.8052	**0.9460**	0.9359
30	0.7	200	0.8701	0.8776	0.8051	0.8015	0.9039	0.8998
100	0.5	100	0.8817	0.8882	0.8157	**0.8107**	0.9386	0.9323
100	0.5	30	0.8606	0.8671	0.8021	0.7778	0.9113	0.9076

2. *Word2vec*: pre-trained Word2vec embeddings without further task-specific tuning. Also known as static word embeddings.
3. *Train Word2vec*: Word embeddings initialized with pre-trained Word2vec embeddings and further tuned on the specific task during training. Also known as non-static word embeddings.

Our results strongly support the use of word embeddings, and also indicate that the pre-trained word embeddings carry useful information for the task at hand.

The contribution of character-level word embeddings is instead less impactful, especially as there seems to be a substantial overlap with the contribution of word embeddings: there is only a 1% gain in the best model using both word and BiLSTM character-level word embeddings. Notably, the CNN approach appears to perform less well then the BiLSTM, despite the fact that the gain in speed of a CNN architecture is considerable (3 times faster training, on average). We therefore confirm the relatively low impact of character-level word embeddings, as previously discussed in the literature (Reimers and Gurevych, 2017b), but find that a BiLSTM slightly outperforms a CNN approach for our task.

With respect to the prediction layer, as expected the CRF approach consistently outperforms the softmax, yielding a gain of above 2% when compared with an identical architecture using non-static word embeddings and BiLSTM character-level word embeddings. This result follows from the intuition that tag predictions are not independent in a reference. We eventually tested a multi-task architecture where all layers are shared across the three tasks, besides for the prediction one. Our results are quite encouraging, with performances lowering on average less than 0.5% from the equivalent single-task architecture (**Table 3**). It follows that the input and inner layers learn a set of parameters which are to a large degree shared across tasks.

As discussed in the previous section, the best model is a BiLSTM-CRF network with word embeddings and character-level word embeddings. We fine-tuned this architecture over a set of parameter ranges using grid search, with results presented in **Table 4**.

The results reported in **Table 4** outline the importance of the BiLSTM dimensionality. The best predictions were achieved with a dimensionality of 100 and a medium rate of dropout (0.5), without affecting the running time. The batch size is the parameter with the most influence on the training time: the smaller the batch, the longer the training. A second round of fine-tuning on the best model yielded some further minor improvements, given in **Table 5**. Eventually, **Table 6** reports the final configuration of our best model.

TABLE 4 | Results of the fine-tuning of the best architecture, over the batch size, the dimensionality of the inner BiLSTM and the rate of dropout.

Batch	Dropout	BiLSTM	Testing F1 score
100	0.5	200	89.09
30	0.5	200	88.96
70	0.5	200	88.95
70	0.7	200	88.61
100	0.2	200	88.51
100	0.7	200	88.41
100	0.5	80	88.36
70	0.2	200	88.19
30	0.2	200	88.08
30	0.2	80	88.00
70	0.5	80	87.97
70	0.2	80	87.89
30	0.5	80	87.84
100	0.2	80	87.78
30	0.2	40	87.63
30	0.7	200	87.32
100	0.5	40	87.23
70	0.5	40	87.17
100	0.7	80	86.81
70	0.7	80	86.80
70	0.2	40	86.79
30	0.5	40	86.71
100	0.2	40	86.70
30	0.7	80	86.29
100	0.7	40	84.60
70	0.7	40	83.68
30	0.7	40	83.55

5.2. Evaluation

We report in what follows the validation of the best model, and a discussion of the errors. Some figures and tables are given in the Appendix.

TABLE 5 | Results of the further fine-tuning of the best architecture, over the batch size, the dimensionality of the inner BiLSTM and the rate of dropout.

Batch	Dropout	BiLSTM	Testing F1 score
100	0.5	400	89.56
200	0.5	400	89.24
100	0.5	600	89.13
100	0.5	300	88.99
100	0.5	200	88.89
200	0.5	300	88.61

TABLE 6 | Configuration of the final best model.

Layer	Parameter	Value
Word embeddings	Dimensionality	300
	Min word frequency	5
Character-level word embeddings	Embedding dimensionality	100
	BiLSTM dimensionality	100
BiLSTM	Dimensionality	400
CRF	Metric	Viterbi
Early stopping	Max waiting	5
	Max number of epochs	25
Model	Optimizer	RMSprop
	Dropout	0.5
	Learning rate	0.001
	Decay	0
	Batch size	100

- On Task 1 (**Table 7**), the model achieves an F1 score of 89.66% on the validation dataset, outperforming our CRF baseline by +7.03%. The model performs particularly well on the two most represented tags (*title* and *author*): these two tags combined account for more than the 2/3 of the dataset. All tags with 500 or more examples in the validation dataset perform quite well, at the exception of the *o* and *publisher* tags. The *o* tags are probably both not well represented and difficult to grasp (too generic). When compared with the CRF baseline, in Table S1 (Appendix), the neural network approach performs better for the *title* and *author* tags, and the vast majority of the rest, especially so for the *publisher* and *o* tags.
- On Task 2 (**Table 8**), the model achieves an F1 score of 81.51% on the validation dataset, and outperforms the CRF baseline by +10.47% (Table S2 in Appendix). The model performs well overall for the most represented tags in the dataset, such as the *i-* tags, but it shows issues with the begin and end *primary* annotations, that are often difficult to capture. The lower results of the model on this task, if compared with tasks 1 and 3, suggests that distinguishing between primary or secondary references might not be a sequence labeling problem but a classification one, over the entire line/reference.
- On Task 3 (**Table 9**), the model achieves an F1 score of 95.09% on the validation dataset, and outperforms the CRF baseline by +2.59% (Table S3 in Appendix). In particular, the model improves on the *o*, begin and in tags, while lowering its performance on the end tag.

TABLE 7 | Classification report for Task 1.

	Precision	Recall	f1-score	Support
Abbreviation	0.1333	0.0460	0.0684	87
Archivalreference	0.8163	0.4878	0.6107	328
Archive_lib	0.2857	0.8235	0.4242	17
Attachment	0.0000	0.0000	0.0000	0
Author	0.8928	0.9742	0.9317	4581
Box	1.0000	1.0000	1.0000	6
Cartulation	0.0000	0.0000	0.0000	10
Column	1.0000	1.0000	1.0000	6
Conjunction	0.4778	0.7167	0.5733	120
Date	0.6667	0.3158	0.4286	19
Filza	0.8333	0.2143	0.3409	70
Folder	0.0000	0.0000	0.0000	0
Foliation	0.0000	0.0000	0.0000	0
Numbered_ref	0.0000	0.0000	0.0000	87
o	0.8066	0.4445	0.5732	1379
Pagination	0.9504	0.9801	0.9650	1154
Publicationnumber-year	0.8874	0.8767	0.8820	665
Publicationplace	0.9569	0.9421	0.9494	1555
Publicationspecifications	0.4068	0.3982	0.4025	329
Publisher	0.8941	0.8196	0.8552	937
Ref	0.2576	0.4722	0.3333	36
Registry	0.7447	1.0000	0.8537	35
Series	0.7949	0.7209	0.7561	43
Title	0.9390	0.9651	0.9519	13744
Tomo	0.3030	0.3030	0.3030	33
Volume	0.7822	0.5254	0.6286	335
Year	0.9088	0.9582	0.9328	1601
Avg/total	0.9006	0.9022	0.8966	27177

TABLE 8 | Classification report for Task 2.

	Precision	Recall	f1-score	Support
b-meta-annotation	0.7473	0.7500	0.7487	280
b-primary	0.6957	0.3556	0.4706	45
b-secondary	0.7737	0.7022	0.7362	779
e-meta-annotation	0.7970	0.8532	0.8242	879
e-primary	0.4382	0.2335	0.3047	167
e-secondary	0.8399	0.7789	0.8083	1583
i-meta-annotation	0.7594	0.8269	0.7917	8457
i-primary	0.5772	0.8444	0.6857	270
i-secondary	0.8687	0.8475	0.8580	13682
o	0.7950	0.5507	0.6507	1035
Avg/total	0.8181	0.8162	0.8151	27177

We further discuss the error confusion matrices over the validation dataset, in order to compare the proportion of

TABLE 9 | Classification report for Task 3.

	Precision	Recall	f1-score	Support
b-r	0.8498	0.7636	0.8044	1104
e-r	0.8963	0.7703	0.8286	1145
i-r	0.9633	0.9904	0.9767	23893
o	0.8491	0.5217	0.6463	1035
Avg / total	0.9515	0.9541	0.9509	27177

classifications gone well or wrong, for each tag. Starting with Task 1 (Figure S1 in Appendix), we can see how systematic errors tend to be caused by two reasons: under-represented tags or very similar encoding styles or contents for different styles. Examples are the *date* tag mistaken for a *year*, or an *abbreviation* mistaken for an *author* (initials). The confusion matrix for Task 2 (Figure S2 in Appendix), broadly follows along the same lines, further highlighting how most frequent tags tend to ac as attractors of wrong classification actions. Indeed, the tag *i-secondary* is often misassigned. Interestingly, *i-secondary* tags are sometimes predicted as *i-meta-annotation*, the second most frequent tag in the training dataset: indeed, their contents are often very similar. Quite crucially, when a prediction is wrong it is often assigned to the correct IBOE tag, but the wrong reference type. This would allow to adopt a voting system to refine a classification at a further stage. The confusion matrix for Task 3 (Figure S3 in Appendix), shows that the inside tag is correctly predicted, but reveals a fragility in the *e-r* tag predictions. Indeed, a lot of *e-r* tags are labeled as *i-r* by the model. The model also performs poorly in predicting the out-of-reference *o* tag.

In conclusion, the neural network model substantially outperformed the CRF baseline in all tasks, with minor downgrade of performance on some infrequent tags, but an important gain on most of the rest. All systematic errors can be explained either by the important imbalance in the amount of training examples per tag, or by the similarity in either contents or referencing styles between some tags.

6. CONCLUSION

In this work, we applied a state-of-the-art deep learning architecture to the task of reference mining, with a focus on applications in the arts and humanities. In particular, the model is trained to extract and parse references within the full text of publications, such as in footnotes, yet it can be applied more generally. The final architecture follows previous work in sequence labeling tasks, by integrating word embeddings and character-level word embeddings into word representations as inputs, an inner BiLSTM layer and a CRF prediction layer. As was shown for a variety of similar tasks, important components of the network result to be pre-trained word embeddings, which integrate information on the use of words within a broader textual corpus, and the CRF prediction layer, which accounts for the dependency among tag predictions (Reimers and Gurevych, 2017a). Furthermore, for the specific task at hand, we showed the relative positive contribution of character-level word embeddings. Given the importance of morphological and orthographical features in references, and the lack of large quantities of annotated data to learn word representations from, character-level features proved to be a minor yet positive addition. This model was tested on a dataset of annotated references extracted from a corpus of scholarly literature on the history of Venice, and it improved considerably over a CRF baseline using a rich set of hand-crafted features, with F1 gains going from +2.59% to +10.47% on different tasks. Furthermore, a multi-task architecture was found to perform almost on par on all tasks combined. We released two implementations of the architecture, in Keras and TensorFlow, along with all the data we used to train and test it. These results strongly support the adoption of deep learning methods for the general task of reference mining.

This work used a relatively small dataset with some limitations, reflecting the current situation with respect to reference mining and, more broadly, citation indexing in the arts and humanities. The dataset contains several sources of noise, including OCR errors, referencing errors or inconsistencies, annotation errors. In part for this reason, we consider as the most important next step for future work to explore how active learning or semi-supervised learning techniques might be used in order to maximize the model gain while at the same time minimizing the costly process of manual annotation (Peters et al., 2017; Shen et al., 2018). At the same time, we plan to explore how to align and use existing annotated datasets with coverage in the arts and humanities (Anzaroot and McCallum, 2013). Furthermore, it remains to be tested to what extend reference parsers trained on scientific publications could be adapted for the literature in the arts and humanities.

AUTHOR CONTRIBUTIONS

GC conceived the research project, DR and GC developed code and experiments and wrote the paper, FK contributed with advice.

FUNDING

GC was supported by the Swiss National Fund with grant number 205121_159961 for part of this research.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frma.2018.00021/full#supplementary-material

Supplementary Data

The dataset and code for this article is available at the following address on GitHub: https://github.com/dhlab-epfl/LinkedBooksDeepReferenceParsing, the pre-trained word vectors are available on Zenodo at the following address: http://doi.org/10.5281/zenodo.1175212.

REFERENCES

Anzaroot, S. and McCallum, A. (2013). *A New Dataset for Fine-Grained Citation Field Extraction.* Atlanta, GA: JMLR: W&CP.

Ardanuy, J. (2013). Sixty years of citation analysis studies in the humanities (1951-2010). *J. Am. Soc. Inform. Sci. Technol.* 64, 1751–1755. doi: 10.1002/asi.22835

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* 5, 157–166. doi: 10.1109/72.279181

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. Sebastopol, CA: O'Reilly.

Chiu, J. P., and Nichols, E. (2016). Named entity recognition with bidirectional LSTM-CNNs. *Trans. Assoc. Comput. Linguist.* 4, 357–370.

Chollet, F. et al. (2015). *Keras*. Availbale online at: https://github.com/keras-team/keras.

Colavizza, G., and Romanello, M., (2017). Annotated References in the Historiography on Venice: 19th-21st centuries. *J. Open Human. Data.* 3:2. doi: 10.5334/johd.9

Colavizza, G., Romanello, M., and Kaplan, F. (2017). The references of references: a method to enrich humanities library catalogs with citation data. *Int. J. Digit. Libr.* 18, 1–11. doi: 10.1007/s00799-017-0210-1

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* 12, 2493–2537.

Councill, I. G., Giles, C. L., and Kan, M.-Y. (2008). "ParsCit: an open-source CRF reference string parsing package," in *Proceedings of the Language Resources and Evaluation Conference (LREC 2008)* (Marrakesh).

Dai, H.-J., Lai, P.-T., Chang, Y.-C., and Tsai, R. T.-H. (2015). Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization. *J. Cheminform.* 7:S14. doi: 10.1186/1758-2946-7-S1-S14

Dean, J., Monga, R., and Google Research (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Available online at: https://www.tensorflow.org/.

Dos Santos, C., and Gatti de Bayser, M. (2014). "Deep convolutional neural networks for sentiment analysis of short texts," in *Proceedings of the 25th International Conference on Computational Linguistics: Technical Papers* (Santa Fe), 69–78.

Dos Santos, C., and Zadrozny, B. (2014). "Learning character-level representations for part-of-speech tagging," in *ICML, Vol. 32 of JMLR Workshop and Conference Proceedings* (JMLR.org), 1818–1826.

Garfield, E. (1979). *Citation Indexing: Its Theory and Application in Science, Technology, and Humanities*. New York, NY: John Wiley & Sons.

Graves, A., Mohamed, A.-R., and Hinton, G. (2013). "Speech recognition with deep recurrent neural networks," in *Acoustics, Speech and Signal Processing (icassp), 2013 IEEE International Conference on IEEE* (Budapest), 6645–6649.

Graves, A., and Schmidhuber, J. (2005). "Framewise phoneme classification with bidirectional lstm networks," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005*, *Vol. 4*, 2047–2052.

Heckmann, D., Frank, A., Arnold, M., Gietz, P., and Roth, C. (2016). Citation segmentation from sparse and noisy data: a joint inference approach with Markov logic networks. *Digit. Schol. Hum.* 31, 333–356. doi: 10.1093/llc/fqu061

Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*, eds, S. C. Kremer and J. F. Kolen (IEEE Press).

Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.* 9, 1735–1780. doi: 10.1162/neco.1997.9.8.1735

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional LSTM-CRF models for sequence tagging. *arXiv preprint*.

Kim, Y. (2014). "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (Doha), 1746–1751.

Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. (2015). "Character-aware neural language models," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)* (Arizona), 2741–2749.

Körner, M., Ghavimi, B., Mayr, P., Hartmann, H., and Staab, S. (2017). Evaluating reference string extraction using line-based conditional random fields: a case study with German language publications. *New Trends in Databases and Information Systems*, Vol. 767, eds M. Kirikova, K. Nørvåg, G. A. Papadopoulos, J. Gamper, R. Wrembel, J. Darmont, and S. Rizzi (Cham: Springer International Publishing), 137–145.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). "Conditional random fields: probabilistic models for segmenting and labeling sequence data," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, (San Francisco, CA: Morgan Kaufmann Publishers Inc.), 282–289.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR* abs/1603.01360. doi: 10.18653/v1/N16-1030

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1, 541–551. doi: 10.1162/neco.1989.1.4.541

Lopez, P. (2009). "GROBID: combining automatic bibliographic data recognition and term extraction for scholarship publications," in *Research and Advanced Technology for Digital Libraries* (Springer), 473–474.

Ma, X., and Hovy, E. H. (2016). End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354. doi: 10.18653/v1/P16-1101

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

Mongeon, P., and Paul-Hus, A. (2016). The journal coverage of Web of Science and Scopus: a comparative analysis. *Scientometrics* 106, 213–228. doi: 10.1007/s11192-015-1765-5

Okazaki, N. (2007). *Crfsuite: A Fast Implementation of Conditional Random Fields*. Available online at: http://www.chokkan.org/software/crfsuite/

Peters, M. E., Ammar, W., Bhagavatula, C., and Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. *arXiv preprint arXiv:1705.00108*

Ratinov, L., and Roth, D. (2009). "Design challenges and misconceptions in named entity recognition," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, (Stroudsburg, PA: Association for Computational Linguistics), 147–155.

Řehůřek, R., and Sojka, P. (2010). "Software Framework for Topic Modelling with Large Corpora" in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (ELRA: Valletta), 45–50.

Rei, M., Crichton, G. K., and Pyysalo, S. (2016). Attending to characters in neural sequence labeling models. *arXiv preprint arXiv:1611.04361*.

Reimers, N., and Gurevych, I. (2017a). Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799*.

Reimers, N., and Gurevych, I. (2017b). Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. *CoRR*, abs/1707.09861.

Ruder, S. (2017). An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098.

Rumelhart, D. E. (1986). Learning internal representations by error propagation. *Nature* 323, 533–536. doi: 10.1038/323533a0

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Netw.* 61, 85–117. doi: 10.1016/j.neunet.2014.09.003

Schuster, M., and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.* 45, 2673–2681. doi: 10.1109/78.650093

Shen, S.-S., and Lee, H.-Y. (2016). Neural attention models for sequence classification: Analysis and application to key term extraction and dialogue act detection. *arXiv preprint arXiv:1604.00077*.

Shen, Y., Yun, H., Lipton, Z., Kronrod, Y., and Anandkumar, A. (2018). "Deep active learning for named entity recognition," in *Proceedings of the 2nd Workshop on Representation Learning for NLP* (Vancouver, BC), 252–256.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1929–1958.

Strubell, E., Verga, P., Belanger, D., and McCallum, A. (2017). "Fast and accurate entity recognition with iterated dilated convolutions," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (Copenhagen), 2660–2670.

Sula, C. A., and Miller, M. (2014). Citations, contexts, and humanistic discourse: toward automatic extraction and classification. *Liter. Linguist. Comput.* 29, 452–464. doi: 10.1093/llc/fqu019

Sutton, C., and McCallum, A. (2011). An introduction to conditional random fields. *Found. Trends Mach. Learn* 4, 267–373. doi: 10.1561/2200000013

Tkaczyk, D., Collins, A., Sheridan, P., and Beel, J. (2018). Evaluation and comparison of open source bibliographic reference parsers: a business use case. *arXiv preprint arXiv:1802.01168*.

Tkaczyk, D., Szostek, P., Fedoryszak, M., Dendek, P. J., and Bolikowski, L. (2015). CERMINE: automatic extraction of structured metadata from scientific literature. *Int. J. Doc. Anal. Recogn.* 18, 317–335. doi: 10.1007/s10032-015-0249-8

Yang, Z., Salakhutdinov, R., and Cohen, W. (2016). Multi-Task Cross-Lingual Sequence Tagging from Scratch. arXiv:1603.06270 [cs]

Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. *ArXiv e-prints*.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.