Master semester project
Digital Humanities Laboratory

Deep reference mining from humanities publications

Rodrigues Alves Danny

Supervised by
Giovanni Colavizza

Fall 2017

Abstract

This project considers the task of reference mining: the detection, extraction and classification of references within a plain text from humanities publications. Despite its similarities to Named Entity Recognition, reference mining applied to the field of arts and humanities brings forward specific difficulties, mainly the need to capture the richness of referencing styles, referred objects, references locations and the morphology of highly abbreviated words.

A deep learning approach requiring no explicit feature engineering is proposed, outperforming a Conditional Random Field model based on hand-crafted features. Several architectures and components are explored and discussed: word and character embeddings, hidden recurrent layers and their directionality, prediction layers (Softmax and CRF), single and multi-task learning. The results highlight the importance of character-level embeddings, which allow to learn word-level encoding styles and be more robust against OCR issues, and of the CRF prediction layer, convenient to consider the structural dependence among predictions. The BiLSTM-CRF neural network with both word and character features learns the compositionality of reference styles, from the encoding of its components at word level to their ordering at the reference level. Trained and tested on an annotated dataset of references from the historiography of Venice, the best model reaches a validation F1 score of `89.13%` for reference components, `79.78%` for generic tags and `94.34%` for begin-end tags.

KEYWORDS: `Reference mining, Bibliometrics, NLP, Deep Learning, BiLSTM-CRF, Embeddings`

Contents

1 Introduction

1.1 Reference mining

Reference mining is a necessary step towards the construction of citation indexes. It enables to find sources the author cites and possible interactions between authors and documents, to understand the pattern of citations and to assert the importance of some publications. Compared to other Natural Language Processing (NLP) tasks, reference mining is similar to sequence labeling problems, such as Part-Of-Speech (POS) tagging or Named Entity Recognition (NER). Most common sequence labeling models are linear statistical ones, like Hidden Markov Models (HMM), Conditional Random Fields (CRF) or Decision Trees. To be efficient, those systems depend on a large amount of knowledge, in the form of hand-engineered features or task-specific resources like gazetteers, lexicons or dictionaries. However, acquiring this type of resource is costly and requires experts' knowledge, which may not be always available. This makes sequence labeling problems difficult to solve and to be adapted for new tasks or domains.

Recently, attention has been given to the use of unsupervised techniques in a variety of NLP tasks, to offer an alternative strategy for obtaining sequence labeling resources. Those unsupervised techniques lead to state-of-the-art performances, mainly due to the improvements in word embeddings based on massive amounts of data and neural network training algorithms[23]. However, even systems that rely heavily on unsupervised features have used these to augment, rather than replace, hand-crafted features. Using only the neural word embeddings leads to poor performance. The most recent works on sequence labeling tasks focus on creating end-to-end systems, without the need of any manually created features[20, 22, 2, 18]. Those systems rely on a neural network architecture built around a Bidirectional Long-Short Term Memory (BiLSTM) layer. This type of neural networks has a variable length memory allowing the model to capture relationships inside a sequence between words distant one from the other. Those architectures have achieved state-of-the-art performance for both POS and NER with popular and well-known data[29].

Sequence classification is not an easy problem, as sequences can vary in length, be constituted of a very large vocabulary of inputs and may require the model to learn long-term contexts or dependencies between symbols in the input sequence. It is not surprising to see deep learning methods perform well on this type of task, as one ability often cited of those models is to perform automatic features extraction from raw data, also called features learning. Deep learning methods can learn the features from natural language, no need to pre-define and extract features.

In this project, recent neural network architectures are tested for reference mining with a dataset from a new domain: the humanities. Despite its similarities with NER, reference mining brings forward specific challenges, like the need to capture the morphology of highly abbreviated words found in references or the structural dependence of the elements of a reference, which both follow

codified styles. This task is particularly difficult and exploring it in the context of the arts and humanities prompt to new challenges.

1.2 The arts and humanities

Citing in the humanities is a less standardized practice than in other sciences. Publications rarely - or never - have a reference list at the end of the document with all citations stored in a single place. Instead, citations are commonly made in footnotes. Furthermore, humanist have developed their own elaborated practices for the abbreviations and encoding of references, which also entail making a variety of usages of formatting features such as italics or variations in styles. Lastly, it is common in the humanities to refer to both primary and secondary sources. In general, a primary source is a documentary evidence used to support a claim, and a secondary source is a scholarly publication. Unfortunately, these characteristics of citing in the humanities make it difficult to reuse existing services out-of-the-box and are at the core motivation of building a dedicated machine learning model [8].

1.3 The approach taken

In this project, multiple neural networks architectures are analyzed for reference extraction. Based on state-of-the-art architectures with the CoNLL-2003 corpus for NER, all models are built around a BiLSTM layer. In a first part, several components were analyzed to assert the utility of using word embeddings or character-level embeddings, the benefits of using a CRF layer over a Softmax layer for predictions and their impact in the training time of each model. In a second phase, the parameters for the best architecture are fine-tuned and the pertinence of a multi-task approach is discussed. Finally, the best model is used on a never-touched validation dataset to compute the final F1 score. The implementation was done with *Keras*[5] using *Tensorflow*[16] as backend[1]

In section 2, the dataset and its specificities are discussed. The baseline model, done with a CRF layer and hand-crafted features is also presented. In section 3, the architecture of the best model is detailed and each layer inspected. Section 4 contains the overview of all architectures tested, the final model with its parameters and the results for each task. Sections 5 and 6 discuss results and close this project.

2 Task definition and dataset

In this section, the tasks to be solved by the model are outlined and the dataset used is presented with its characteristics and specificities. Before continuing with neural network models, the baseline, a Conditional Random Field model based on feature engineering, is presented with its results for each of the three tasks.

[1] Keras version `2.1.2` and Tensorflow version `1.4.0`.

2.1 The tasks

The tasks in this project concern the classification of references and the annotation of their components. A reference is defined as the text containing a citation from an author to another work. Extracting references from humanities publications carries more complexity than other sciences fields. Indeed, humanities references are complex considering that there is not a unique style for them. For example, there is no obligation of having a reference list at the end of documents, humanist have rather an intensive use of footnotes. Reference styles are quite varied at both the syntactic and semantic level, and change over time, author, publisher, or publication type, thus complicating the generalization.

A reference is usually composed of four components, such as *author*, *title*, *publisher* or *publication year* encoded in a systematic way (e.g. using double quotes for the title). An example of a reference is

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

As detailed in Figure 1, this reference is composed of four components: author's name first, then title, publisher, and year. In this example, components are separated by a comma and author's first-name is abbreviated with a capital letter followed by a dot. In another document, the same citation could be contained in a reference where the first-name is not abbreviated, the publication's year appears before the publisher or the title is surrounded by double quotes.

References are composed of tokens: a token is referred as a word-level entity. The goal of this project is to annotate each token in a sequence of plain text. Two types of annotations are considered: specific and generic. A specific annotation identifies a component of a reference, such as *author, title* or *publisher* in the above example. A generic annotation refers to the type of a reference (if a monograph or a primary or secondary source) and the position of the token in the reference (begin, end, inside or outside).

For plain text coming from humanities publications, the model will assign the most likely tag to each token (token by token classification). Three type of annotations are considered:

- Task 1: *Reference components.* The specific annotations of a reference will be retrieved. There are 56 different specific tags which have been consolidated into 27 classes in order to regroup similar and under-represented tags [2].
- Task 2: *Generic tags.* For each token, the model predicts its associated generic annotation. As mentioned above, those tags classify document between *primary* sources (e.g.: archival documents), *secondary* sources (e.g.: books), and *meta*-sources (e.g: journal articles)[3]. Alongside with the document's type, *begin-end* tags are assigned, as for Task 3.

[2] See Annex A for the full taxonomy

[3] This separation is required by the *LinkedBooks* project[7]

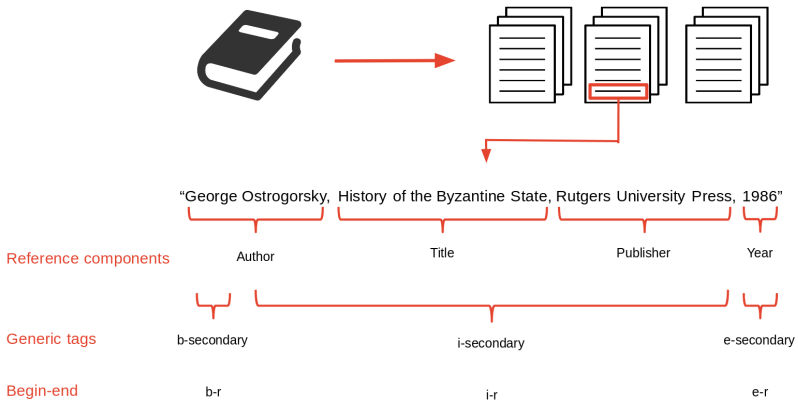


Figure 1: Example of a reference and the tags associated with each token for Task 1. As this is a reference to a secondary source, tags for Tasks 2 and 3 will all contain the appropriate IBOE tag followed by *-secondary*

.

- Task 3: *Begin-end*. For each token, the model predicts if the token is located at the beginning, end, inside or out of a reference.

Note that those tasks do not rely on the outcome of the others. This could be considered in the future, as mention in section 5.

2.2 Dataset

The dataset at the core of this project is part of the *LinkedBooks* project[4], an initiative to *develop an in-depth approach to the problem of indexing humanities' publications via citations [7]*. The documents composing the dataset come from the historiography on Venice, from the 19^{th} century to nowadays. Both monographs and journal articles were considered, giving to the dataset a variety of writing standards, styles, characteristics and specificities, all bringing complexity for the generalization of references. Besides originating from different authors, document's types or time periods, the dataset includes texts from different languages: mostly Italian, but there are also texts in English, French, German, Spanish or Latin. As humanities haven't an unique and simple *predefined* standard for references, both reference lists and footnotes can include a reference or an abbreviated reference.

The documents were digitalized, OCRed, and tokenized[5] to obtain *word by word* texts. Then, the data was manually annotated and each line containing a reference in it was taken into consideration[6]. Those lines will form the data for

[4] *LinkedBooks* is in turn part of the *Venice Time Machine* Project: `https://vtm.epfl.ch`

[5] OCR performed white *Abbyy FineReader Corporate 12*, tokenization with *scikit-learn*

[6] Annotation strategy detailed in [7].

this project: text blocs organized in *token by token* sequences.

The lines forming the dataset were taken from a public online resource [7] and split in three text files: one for training the models (80% of the documents), one for testing the models (10% of the documents), and the last one used only for computing the final score for the best model (10% of the documents)[8]. Those files follow the CoNNL convention and the IBOE tagging scheme: each line corresponds to a token in a sequence and sequences are separated by a blank line. Word lines are composed of word's surface form followed by the labels for each tag, all separated by a white space. To represent the position of a token in a reference, the IOBE format is used, where `i-label` stands for a token inside the reference (not begin or end), `o-label` for outside, `b-label` if the token is the beginning the reference and `e-label` the end. The IOBE format is a variant of the more common IOB scheme format. Using a more expressive tagging scheme like IOBE has been shown to improve marginally model performance [28] [9] and eases the retrieval of references spanning across several lines.

For example, the sequence *"G. Ostrogorsky, History of the Byzantine State, Rutgers University Press, 1986"* 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
```

with a blank line before and after.

2.3 CRF baseline

The goal of this project is to create a neural network model to perform reference mining. Before the rise of neural networks, state-of-the-art models in reference

[7] https://github.com/dhlab-epfl/LinkedBooksReferenceParsing

[8] The split on the annotated dataset is performed at the *BID* level, an unique identifier for documents. Therefore, the number of sequences in each file might not follow the same distribution.

mining were Conditional Random Fields [19]. Based on previous work with similar data [7], a CRF model was build to serve as a baseline and be compared to the neural networks models.

In this model, most effort has been put into feature engineering. The CRF classifier is trained over a rich set of hand-crafted features considering a two-token window: the features for a token at the i-position in a sequence include features of the tokens around it, from i-2 to i+2. Among the set of features, there is, for example, the shape of the token (UUU for a three uppercase word) or if the token contains a digit [9].

The CRF models were trained with SGD and L2 regularization, using *CRFSuite* and default parameters [25][10]. After training, fine-tuning parameters, explored with a random search cross-validation, were founded based on the testing dataset. The three models F1 score on the validation dataset are:

Task 1 with `c1=1.3099` and `c2=0.0773` gives a F1 score of **`82.63%`**
Task 2 with `c1=0.9298` and `c2=0.0229` gives a F1 score of **`71.04%`**
Task 3 with `c1=2.1334` and `c2=0.0142` gives a F1 score of **`92.50%`**

where c1 and c2 refer to coefficients for L1 and L2 regularization, respectively. The details of the classification for each task are given in annex B.2.

3 Model

Multiple neural network architectures have been explored and the details of this model investigation phase are reported in section 4.1. Consistent with state-of-the-art models in NER or POS, the final model turns out to be a BiLSTM-CRF neural network with word and character features. First, the network receives a sequence of words $w_1, w_2, ..., w_n$ as input and transforms it to a sequence of vectors by combining word embeddings with character-level features (3.1). These word representations are passed to a bidirectional LSTM composed of two layers: a forward layer where the word representations are processed starting with word w_1 until word w_n, and a backward layer from the word representation of w_n to the one of word w_1 (3.2). The outputs of these two layers are merged and given to a CRF layer, which predicts a sequence of tags $t_1, t_2, ..., t_n$ for each input word $w_1, w_2, ..., w_n$ (3.3).

3.1 Word's representation

The model will receive word representations based on embeddings as input. Embeddings are vector representations of individual entities, holding a notion of *distance*. Different from other approaches like bag-of-words or one-hot encoding, word embeddings can tell something about the similarity of two words: the "closest" two word embeddings are, the most similar the words are. For example,

[9] Exhaustive list of features is available in annex B.1.

[10] *CRFsuite* implementation from sklearn-crfsuite, version 0.3.6, `https://github.com/TeamHG-Memex/sklearn-crfsuite`.

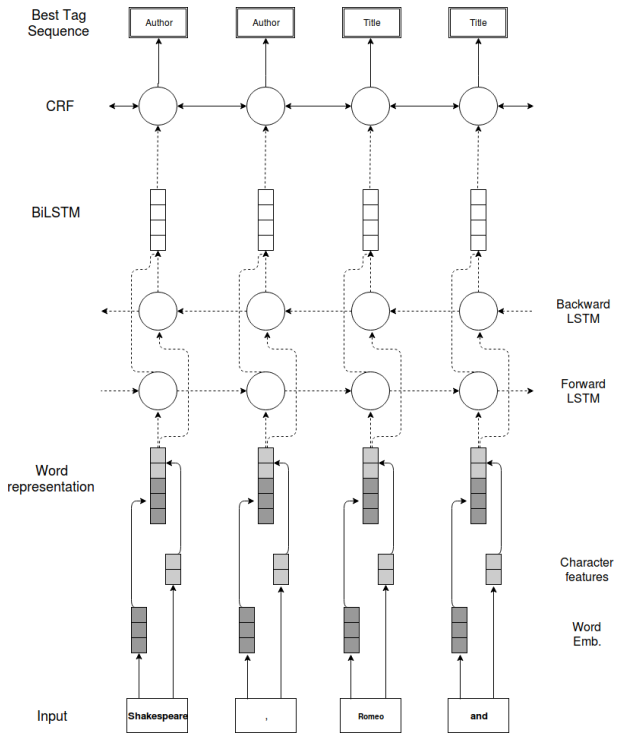


Figure 2: Sketch of the model architecture for a part of the sequence `W. Shakespeare, Romeo and Juliet, Oxford University Press, London, 1914`. Rectangles are used for inputs, double rectangles for outputs, sequences of squares for vectors, rounds for neurons and dashed lines for dropout layers.

the embeddings for words *Paris* and *Venice* are expected to be closer than the ones between *Paris* and *dog*. Two types of embeddings are used to form the word representations: word and character embeddings.

3.1.1 Word embeddings

Using word embeddings is usual for sequence classification tasks. It is common in NLP to use publicly available embeddings, trained on large corpus of text like *Wikipedia*[11] or *Reuters*[12]. Considering the uniqueness of the dataset used in this project (section 2), the assumption is made that those publicly available embeddings will not achieve the best performance for the models. Instead, word embeddings were pretrained using Word2Vec [24][23] on the entire dataset[13].

[11] `https://nlp.stanford.edu/projects/glove/`

[12] `https://www.cs.umb.edu/ smimarog/textmining/datasets/`

[13] Another option could be to use *skip-n-gram*, a variation of word2vec accounting for word order [21].

During training, default parameters were used. The data is composed of all words appearing at least five times in the dataset, while words discarded have been regrouped under an unknown token `$UNK$` and all digits have been merged into a `$NUM$` token. The word embeddings have a dimension of 300.

3.1.2 Character features

Using only word embeddings can give a good representation of the dataset. But considering the tasks of this project, word embeddings can have trouble to learn independent representations for words. Languages have important properties at the orthographic and morphological levels, like the prefix and suffix information, which are relevant to the task at hand. While using only word embeddings, character-level features, like prefix, suffix and spelling of words, will be integrated inside the embeddings, but without the same influence on the model as if they were in a dedicated part of the word representation. Constructing character-level features has the advantage to be entirely specific to the task and domain considered. These type of features have been found to be useful for out-of-vocabulary words and morphologically rich languages [11]. Character embeddings are a representation of a word from representations of the characters the word is composed of. To be sensitive to the order of characters, character-level features are learned with a bidirectional LSTM. Character-level features can in part have a similar role to the hand-engineered features used in the CRF baseline model, like the prefix and suffix features or the use of capitalized letters. Another goal with character-level features is to make the model more robust to small word deformations and resulting rare words, due to OCR issues. Rare words will have poorly trained word embeddings and considering them as a sequence of characters may result in better word representations.

3.1.3 Architecture

Figure 3 describes the complete pipeline to build the word representation input. For each word, its representation is a concatenation of its word embeddings and its character features. The word embeddings consist of a lookup to the precomputed Word2vec embeddings and the character-level representation is computed through neural network layers. For a word, each character inside it is mapped to its embedding, initialized at random. Similar to what has been done for word embeddings, each digit is mapped to the same character embedding. The character embeddings for each character are then sent to a BiLSTM to learn character interactions, both forward and backward. The output of this BiLSTM layer is concatenated with the word embeddings and give as direct input to the model inner layers. The two LSTMs have a hidden dimension of 50, for forward and backward representations. The final character-based representation of a word has dimension 100, word embeddings have a dimension of 300, which gives a word representation vector of dimension 400.

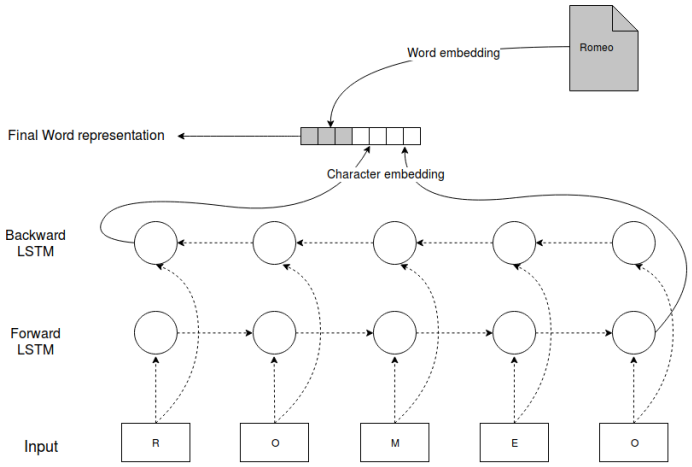


Figure 3: Model architecture for building the representation of the word "Romeo". The word representation is a concatenation of word embeddings, retrieved by a look-up to a table, and character-level features, computed with a BiLSTM layer.

To prevent the model to depend too strongly on word embeddings or character features, dropout training is applied and a dropout layer added before giving the word representation to the BiLSTM layer. As sketched in Figure **??**, dropout layers are applied on several components of the final model. Dropout is a regularization technique where randomly selected neurons are ignored during training. It helps to prevent overfitting and avoid the model to depend to heavily on an individual feature. [32].

3.2 Inner layer

The core of the model is a bidirectional LSTM layer.

3.2.1 LSTM

Long-Short Term Memory networks are part of the Recurrent Neural Networks (RNN) family [15]. This type of network is designed to recognize patterns in sequences of data. This is a different problem compared to other types of supervised learning as sequences impose an order on observations. Inside RNNs, there are sequential connections between neurons. Those connections can form loops which enable the network to have feedback and a memory over time. This is possible because recurrent networks have the ability to take their own outputs as input for subsequent steps. RNNs keep information about past inputs for an amount of time not fixed a priori, but which depends on the internal neuron weights, enabling the network to learn long-term dependencies. The RNN's memory allows networks of this type to learn and generalize across sequences

of inputs rather than local patterns. In RNN, the process of carrying memory forward is describe as:

$$\begin{aligned} \mathbf{h}_t &= \sigma(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1}) \\ \mathbf{y}_t &= softmax(\mathbf{W}_s\mathbf{h}_t) \end{aligned}$$

where $\mathbf{W}, \mathbf{W}_s, \mathbf{U}$ are connection weights computed during training.

LSTM development resulted in fixing vanishing and exploding gradients[14], two issues of regular RNNs occurring in the training phase. LSTMs are designed to be able to learn order dependencies for sequence problems. The hidden layer of an LSTM layer uses gates to manage the proportion of the input to give to the memory cell, as well as the proportion from the previous state to forget [15]. Inside Keras, the LSTM layer uses the following formulas to update itself at timestep t:

$$\begin{aligned} \mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{b}_i) \\ \mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{b}_f) \\ \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_c) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{b}_o) \\ \mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t) \end{aligned}$$

where σ is the element-wise hard-sigmoid function[15] and $\odot$ is the element-wise product. $\mathbf{x}_t$ is the input vector (e.g. word representation) at timestep t, $\mathbf{h}_t$ is the hidden state vector (or output vector) storing all useful information at time **t** (including past information, *a.k.a* the RNN's memory). $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_c$, $\mathbf{W}_o$ are the weight matrices for different gates in the LSTM unit, $\mathbf{U}_i$, $\mathbf{U}_f$, $\mathbf{U}_c$, $\mathbf{U}_o$ are the hidden-state-to-hidden-state matrices (also known as transition matrices) and $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_c$, $\mathbf{b}_o$ are the bias vectors for each gate. Figure 4[16] illustrates a single LSTM memory cell.

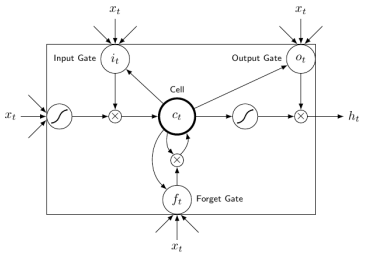


Figure 4: A LSTM Memory cell.

[14]Those issues lead RNNs to give much more importance to recent inputs in the sequence and fail to learn long dependencies.

[15]The hard-sigmoid function is a straight line approximation of the sigmoid function $\frac{1}{1-e^{-x}}$.

[16]From `https://tex.stackexchange.com/questions/332747/how-to-draw-a-diagram-of-long-short-term-memory`

3.2.2 Bidirectional LSTM

LSTM networks have trouble learning time-dependencies more than a few timesteps long. As inputs are processed in temporal order, a shortcoming of LSTMs is their ability to make use of the previous context only [14]. By considering both previous and future context, LSTMs are able to process the data in both directions with two separate hidden layers, which are then fed forward to the same output layer. This is referred as a Bidirectional LSTM [31] and has shown great results in NLP tasks [13]. Bidirectional layers get the most out of the input sequence by stepping through input in both the forward and the backward directions.

3.3 Prediction layer

The most simple way of predicting tags in a neural network is to use the output of the previous layer and make independent tagging decisions for each output, using the softmax function:

$$softmax(k, x_1, ..., x_n) = \frac{e^{x_k}}{\sum_{i=1}^{n} e^{x_i}}$$

But in the context of sequence classification, this method isn't optimal as independent classification decisions are limited when there are strong dependencies across output tags. For example, in this project's task 3, the tag *i-reference* can never be followed by the tag *b-reference*. These types of tags dependencies are not taken into account with a softmax function.

As detailed in section 4.1, the best model turned out to be one using a CRF layer for tagging predictions. Using a conditional random field for predictions enables to model tagging decisions jointly and take neighboring tags into account.

The CRF version used in NLP, called linear chain CRF, is trained to predict a vector $y = \{y_1, y_2, ..., y_n\}$ of tags given an input sequence $x = \{x_1, x_2, ..., x_n\}$. To do so, a conditional probability is computed [20, 19]:

$$\begin{aligned} Score(\mathbf{x}, \mathbf{y}) &= \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i} \\ p(\mathbf{y}|\mathbf{x}) &= \frac{e^{Score(\mathbf{x}, \mathbf{y})}}{\sum_{y'} e^{Score(\mathbf{x}, \mathbf{y})}} \end{aligned}$$

where $A_{i,j}$ is the transition probability which represents the score of transition from tag i to tag j and $P_{i,j}$ is the emission probability which represents the score of the j^{th} tag of the word i^{th}.

During training, the log probability of correct tag sequences $log(p(y|x))$ is maximized.

4 Experiments

This sections details all the neural networks architectures considered (4.1) and the experiments made to reach the final model (4.2). The results on the validation dataset are presented and discussed (4.3, 4.4). Note that for the model selection and fine-tuning parts, only the Task 1 has been considered. To speed up the experiments, early stopping was used based on the F1 score on the testing dataset, with a patience of 5 epochs and a maximum number of 25 epochs (stop training after five epochs without an increase in the testing score).

4.1 Model selection

As discussed in section 3.2, the neural network model is built around a bidirectional LSTM. Considering this architecture, there are several components that can be used before and after the BiLSTM layer. For the data coming in, three types of word representations were considered: using solely word embeddings, solely character features, or the concatenation of both. With the data coming out of the BiLSTM layer, the model can predict tags using a Softmax activation or a CRF layer. The best components were selected based on the F1 score on testing data, reported in Table 1.

Word embeddings	**Character Features**	**Inner layer**	**Output**	**F1 score**
train word2vec	BiLSTM	BiLSTM	crf	88.36
train word2vec		BiLSTM	crf	87.36
train word2vec	CNN	BiLSTM	crf	87.29
word2vec	BiLSTM	BiLSTM	crf	86.85
word2vec	CNN	BiLSTM	crf	86.16
train word2vec	BiLSTM	BiLSTM	softmax	86.12
train	BiLSTM	BiLSTM	crf	86.10
word2vec	BiLSTM	BiLSTM	softmax	85.96
train		BiLSTM	crf	85.88
train	CNN	BiLSTM	crf	85.56
train word2vec	CNN	BiLSTM	softmax	85.47
train word2vec		BiLSTM	softmax	85.41
word2vec		BiLSTM	crf	84.95
word2vec	CNN	BiLSTM	softmax	84.45
train	BiLSTM	BiLSTM	softmax	83.99
word2vec		BiLSTM	softmax	83.91
train	CNN	BiLSTM	softmax	83.61
	BiLSTM	BiLSTM	crf	83.06
train		BiLSTM	softmax	83.06
	BiLSTM	BiLSTM	softmax	82.05
	CNN	BiLSTM	crf	78.23
	CNN	BiLSTM	softmax	75.28

Table 1: F1 scores on Task 1 for all the different architectures tested. The table is sorted according to the testing score, in decreasing order. The exhaustive version of this table is available in annex D.

4.1.1 Word embeddings

Three possibilities[17]: • Train • Word2vec • **Train Word2vec**

As stated in section 3.1, word embeddings are often used as input for machine learning models solving NLP tasks. There are two ways to obtain word embeddings for a dataset: computing the embeddings based solely on the current dataset or retrieving them from a publicly available list of pretrained embeddings[18]. Embeddings are a vector representation of a given word. While computing the word embeddings, the entire goal is to retrieve the weights for each vector representation. The algorithm used in Keras Embedding layer [12] initializes vector's weights with random values, but Keras allows to manually enter values instead of the random ones. This property is used to have three possible types of word embeddings:

1. *Train*: Word embeddings initialized at random and trained
2. *Word2vec*: Word embeddings from pretrained Word2vec embeddings
3. *Train Word2vec*: Word embeddings initialized with pretrained Word2vec embeddings and trained

Those word embeddings types are also called random, static and non-static.

Using only word embeddings for the input can lead to good results. In Table 1, the second best F1 score is achieved solely with word embeddings as input to the BiLSTM. The models using only word embeddings aren't *grouped together* in the table, their results are spread, meaning that the most important factor isn't the use of word embeddings with or without character features, but the type of word embeddings used. Indeed, all models with word embeddings trained by Keras and initialized with the pre-trained Word2vec ones contribute to a better performance compared to the two other types of word embeddings. Considering the average running time for an epoch, reported in annex D, the type of word embeddings hasn't a significant influence. Even if the version using solely pretrained embeddings is faster, as it doesn't require to train embeddings, the difference represents, in average, less than `40` seconds, or `10%`, of the total running time per epoch[19]. The two other embedding's types accomplish similar running times.

4.1.2 Character features

Two possibilities: • CNN • **BiLSTM**

[17] In bold, the one used in final best model.

[18] Non-exhaustive list of available pretrained embeddings: `https://goo.gl/2Mgprh`

[19] The running time will always refer to the average running time per epoch in the training phase.

LSTM models are capable of encoding very long sequences, however, they have a representation biased towards their most recent inputs [3]. Using a bidirectional LSTM enables to counter this effect: the forward LSTM will have a good representation of the suffix of a word and the backward LSTM will have a good representation of the prefix of a word. An alternative approach is to use deep convolutional networks [22]. CNNs have been designed to discover position-invariants features of their inputs and are used in image recognition for example. Concerning NLP tasks, CNNs have shown great results to obtain character-level features [22, 18, 17, 10, 33].

In this project, character features generated with a CNN have been considered in the model exploration phase and shown one advantage and one disadvantage compared to BiLSTM character features. While training a model, the running time is almost three times inferior for the CNN version, but the F1 score is `7%` lower, in average. Where word representations composed solely of word embeddings achieve great performance, word representation composed solely of character features resulted in the worst models in this analysis. Character features aren't the most important input for the BiLSTM layer, but they help to improve model's predictions and reach state-of-the-art. Considering the two types of character features, the one generated with a BiLSTM always results in better predictions compared to CNN generated features. This is consistent with the findings of a previous study [29].

4.1.3 Words and characters

Instead of relying only on word embeddings or character-level features for the word representations, a model can use both. Concatenating these two vectors lead to the best results in the model selection phase, as detailed in Figure 1. Using solely word embeddings, vectors initialized with pretrained weights result in the best predictions, and when character features were used solely, the BiLSTM version gets the best predictions. Combining both inputs, the model achieves its best predictions with the same components as for individual inputs: word embeddings initialized with pretrained weights and BiLSTM character-level features. When the word representations are composed of both embeddings and character features, the running time doesn't vary much (`+5%` more compared to 4.1.2) and is dominated by the character features algorithm.

4.1.4 Prediction layer

Two possibilities: • Softmax • **CRF**

As explained in section 3.3, two predicting layers are considered: Softmax and CRF. The CRF layer is expected to result in better predictions, due to its jointly tagging decisions. This assumption is confirmed by experience as, for similar architectures, models with a CRF prediction layer always outperformed models with a softmax prediction layer (average gain of `1.8%` in the F1 score for CRF). This gain in tagging score influences the running time, as CRF models

need in average 150 more seconds to be trained. Among the seven architectures with an F1 score higher than 86%, only one of them has a softmax final layer (6^{th} position in Table 1).

4.1.5 Multi-task approach

Multi-task learning has been considered to train and predict the three tasks at one. Multi-task learning shares the hidden layer of the model between all tasks, while keeping several task-specific output layers. With the model chosen in this project, all tasks will have the same layer, except the CRF prediction layer which is individual for each task. The goal is to exploit the commonalities and information from the training signals across tasks: what is learned for a task can help other tasks be learned better. This can result in improved learning efficiency and prediction accuracy for the task-specific models when compared to training the models separately[30].

In the context of this project, the multi-task learning approach didn't lead to better results, with an average F1 score of -0.9% compare to training each model individually[20].

4.2 Fine tuning

Based on the previous section, the best model is a BiLSTM-CRF network with word embeddings and character features. All models presented in table 1 were trained with the same parameters, based on [20][21]. Now that the best model architecture is known, the inner parameters of the model are fine-tuned. Three parameters will be investigated: the batch size, the dropout rate, and the inner LSTM size. Due to time constraints, performing a random search wasn't a viable solution and the values have been arbitrarily defined.

The results reported in table 2 outline the importance of the LSTM output space dimensionality. The best predictions were achieved with a dimensionality of 100 and a high rate of dropout, 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.
The best parameters turn out to be the higher ones. Another *round* of fine-tuning has been performed, with results reported in Table 3. Table 4 report all parameters of the final model.

[20] Results available in annex F.
[21] Batch size of 50, dropout of 0.5 and LSTM's dimension of 100.

Batch	Dropout	LSTM	Testing F1 score
100	0.5	100	89.09
30	0.5	100	88.96
70	0.5	100	88.95
70	0.7	100	88.61
100	0.2	100	88.51
100	0.7	100	88.41
100	0.5	40	88.36
70	0.2	100	88.19
30	0.2	100	88.08
30	0.2	40	88.00
70	0.5	40	87.97
70	0.2	40	87.89
30	0.5	40	87.84
100	0.2	40	87.78
30	0.2	20	87.63
30	0.7	100	87.32
100	0.5	20	87.23
70	0.5	20	87.17
100	0.7	40	86.81
70	0.7	40	86.80
70	0.2	20	86.79
30	0.5	20	86.71
100	0.2	20	86.70
30	0.7	40	86.29
100	0.7	20	84.60
70	0.7	20	83.68
30	0.7	20	83.55

Table 2: F1 scores on testing data for Task 1. Experiments done with the best model from 4.1. Parameters tested are the batch size, the dropout rate and the inner LSTM dimension. Table sorted by decreasing score. The exhaustive results are reported in annex E.

Batch	Dropout	LSTM	Testing F1 score
100	0.5	200	89.56
200	0.5	200	89.24
100	0.5	300	89.13
100	0.5	150	88.99
100	0.5	100	88.89
200	0.5	150	88.61

Table 3: F1 scores on the testing data for Task 1 on the second phase of fine-tuning parameters.

Layer	Parameter	Value
Word Embeddings	dimension	300
	min word frequency	5
Character features	dimension	10
	LSTM dimension	50
BiLSTM	dimension	400
CRF	metrics	Viterbi
Early Stopping	patience	5
	max epochs	25
Model	optimizer	RMSprop
	dropout	0.5
	learning rate	0.001
	decay	0
	batch size	100

Table 4: Final model parameters for training.

4.3 Evaluation

To validate the model, it predicts the tags for the three tasks on a validation dataset. The classification reports for each of those tasks are reported below. The associated classification matrices are detailed in 4.4.

- For Task 1, the model achieves a F1 score of `89.13%` on validation data and outperforms the CRF baseline by +6.5%. The model does a very good job with the two most present tags (*title* and *author*). Combining, those two tags represent more than 2/3 of the dataset. It is interesting to see the metrics for the *publicationplace* tag with the highest precision score but a lower recall. Otherwise, all tags appearing more than 500 times in this validation dataset seem to perform quite well, exception for the *o* and *publisher* tags. The model probably needs more *o* tags examples to predict it better.
  Compare to the CRF baseline, the neural network approach performs better for the *title* and *author* tags and the vast majority of the others, especially for the *publisher* and *o* ones.

	precision	recall	f1-score	support
abbreviation	0.0635	0.0460	0.0533	87
archivalreference	0.7895	0.4573	0.5792	328
archive_lib	0.2778	0.5882	0.3774	17
attachment	0.0000	0.0000	0.0000	0
author	0.9214	0.9496	0.9353	4581
box	1.0000	1.0000	1.0000	6
cartulation	0.0000	0.0000	0.0000	10
column	0.0000	0.0000	0.0000	6
conjunction	0.4759	0.7417	0.5798	120
date	0.6667	0.3158	0.4286	19
filza	0.8333	0.2143	0.3409	70
foliation	0.0000	0.0000	0.0000	0
folder	0.0000	0.0000	0.0000	0
numbered_ref	0.0469	0.0345	0.0397	87
o	0.6669	0.5997	0.6315	1379
pagination	0.9511	0.9601	0.9556	1154
publicationnumber-year	0.8525	0.9038	0.8774	665
publicationplace	0.9628	0.8984	0.9295	1555
publicationspecifications	0.4326	0.3708	0.3993	329
publisher	0.8326	0.8282	0.8304	937
ref	0.0870	0.1111	0.0976	36
registry	1.0000	1.0000	1.0000	35
series	0.6863	0.8140	0.7447	43
title	0.9302	0.9577	0.9438	13744
tomo	0.6364	0.2121	0.3182	33
volume	0.7884	0.4448	0.5687	335
year	0.9337	0.9057	0.9195	1601
avg / total	0.8928	0.8944	0.8913	27177

Table 5: Classification report for the Task 1.

- For Task 2, the model achieves a F1 score of `79.78%` on validation data and outperforms the CRF baseline by +8.74%. The model performs overall well for the most seen tags like the *i-* tags, but it seems to have trouble with the all *primary* annotations.
- For the Task 3, the model achieves a F1 score of `94.34%` on validation data and outperform the CRF baseline by +1.84%. As for Task 2, the model outputs good predictions for the i-*r* tags, but it clearly outperforms results of Task 2 for the *begin* tags. Those results are different from the CRF baseline ones, where the *e-r* tags has a much higher F1 score.

	precision	recall	f1-score	support
b-meta-annotation	0.7378	0.7036	0.7203	280
b-primary	0.6207	0.4000	0.4865	45
b-secondary	0.7612	0.6996	0.7291	779
e-meta-annotation	0.8267	0.8089	0.8177	879
e-primary	0.4487	0.2096	0.2857	167
e-secondary	0.8425	0.7941	0.8176	1583
i-meta-annotation	0.7620	0.7917	0.7765	8457
i-primary	0.4304	0.7667	0.5513	270
i-secondary	0.8434	0.8340	0.8387	13682
o	0.6493	0.5884	0.6173	1035
avg / total	0.7997	0.7979	0.7978	27177

Table 6: Classification report for the Task 2.

	precision	recall	f1-score	support
b-r	0.8928	0.7618	0.8221	1104
e-r	0.8840	0.6323	0.7373	1145
i-r	0.9538	0.9948	0.9739	23893
o	0.9215	0.4425	0.5979	1035
avg / total	0.9472	0.9490	0.9434	27177

Table 7: Classification report for the Task 3.

4.4 Confusion Matrices

The confusion matrices for each task are reported below. A confusion matrix is a visualization of the performance of model, usually a supervised learning one[22]. The diagonal of the matrix represents the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier.

For Task 1, the tags with the higher frequency in the data have good predictions. Mistakes often occur for poorly represented tags or similar tags, like *data* and *year* or *series* and *publicationnumber-year*. The confusion matrix also reveals that a lot of tags are wrongly assigned as *title*. This is probably due to the highly skewed training data, where *title* tags represent 30% of it.

[22] In unsupervised learning it is refer as a matching matrix.

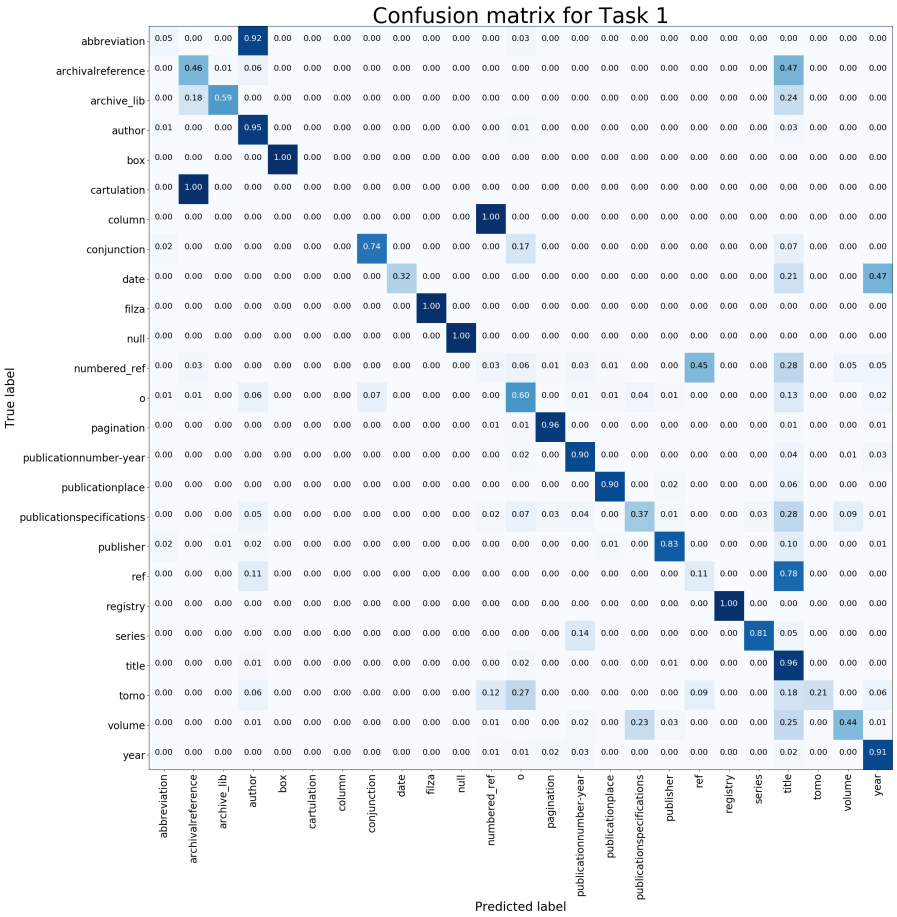


Figure 5: Validation data confusion matrix for Task 1. White cells mean zero predictions and dark blue are for all predictions.

The confusion matrix for Task 2 confirms that predictions are influences by the skewness in training dataset. Indeed, the tag *i-secondary* is often misassigned. It is obvious for tags *e-primary* and *b-primary*, two of the three less represented tags. Interestingly, *i-secondary* tags are sometimes predicted as *i-meta-annotation*, the second more present tag in the training data.
When a prediction is wrong, it is often assigned to the correct IBOE tag, but the wrong reference type.

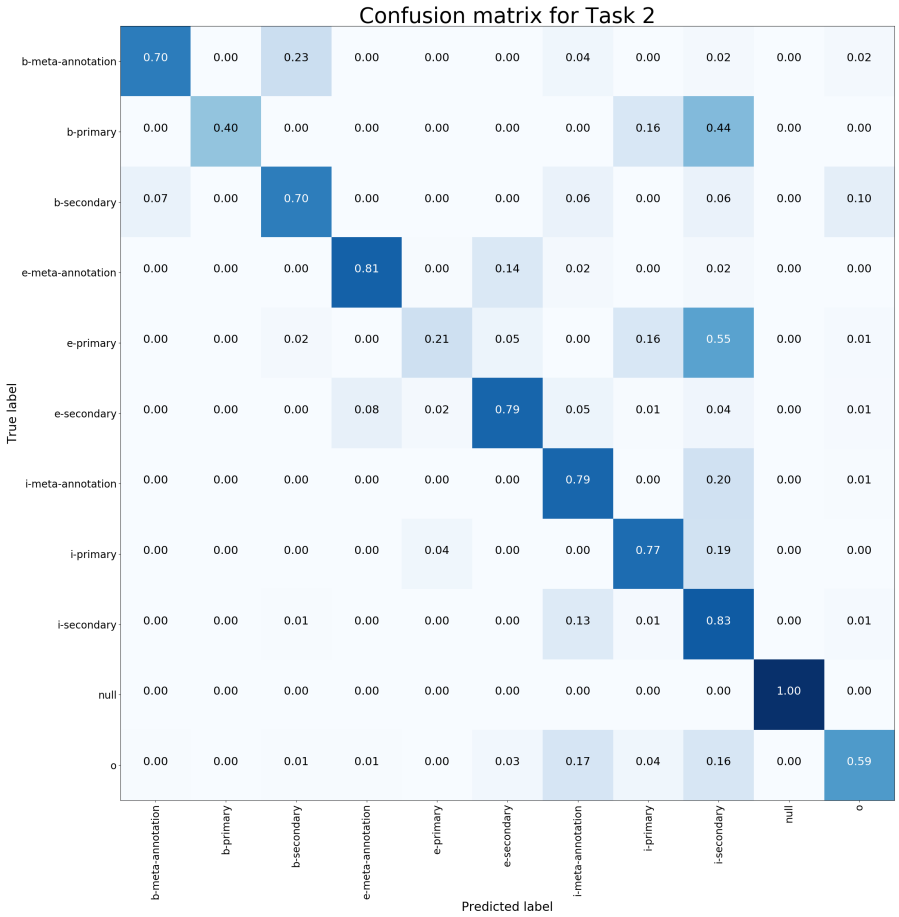


Figure 6: Validation data confusion matrix for Task 2. White cells mean zero predictions and dark blue are for all predictions.

Confusion matrix for Task 3 shows that *inside* tags are correctly predicted, but reveals a fragility in *e-r* tags predictions. Indeed, a lot of *e-r* tags are labeled as *i-r* one by the model. The model also performs poorly to predict the out-of-reference *o-r* tag, but as the inside tags are the most present in the dataset, this doesn't affect to much the F1 score reported in 4.3. This is a flaw in the model's predictions.

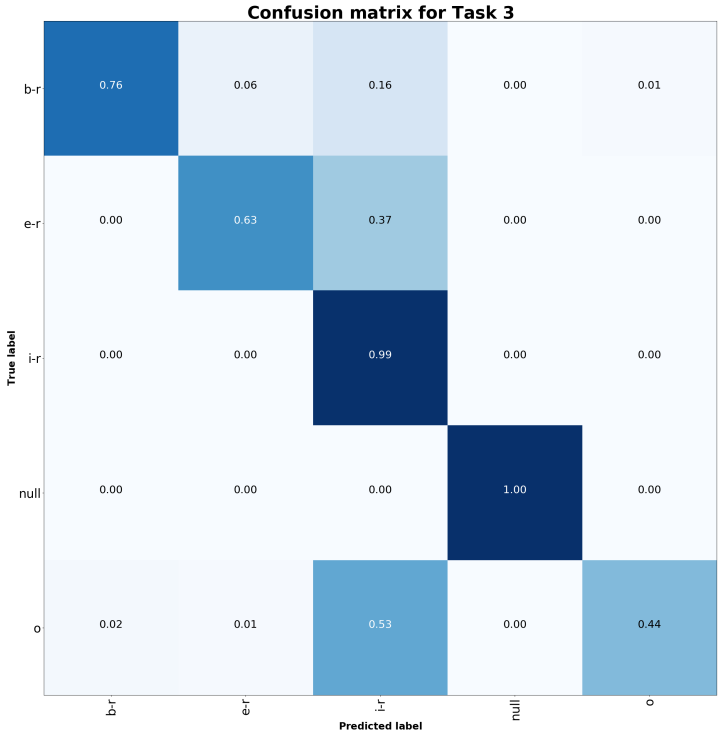


Figure 7: Validation data confusion matrix for Task 3. White cells mean zero predictions and dark blue are for all predictions.

5 Discussion

The best model reveals out to be a bidirectional LSTM with a CRF layer for jointly tagging decisions and a combination of word embeddings with character features for the word representation inputs. Those word representations are the most influent parameter to achieve good predictions. The results of the models exploration phase (4.1) clearly show that word embeddings are at the core of good results. Character-level features are also important and increase the F1 score by 1% for the best model, but the results reported in Table 1 demonstrate that CNN character features can be harmful to the model. This motivates the claim that word embeddings are the most important component of the neural network and deserve to be further explored. One of the specificities in the humanities field is that there is no large corpus of data available and there is some work to do on word embeddings. Among the possible improvements, one can try to fine-tune the Word2vec parameters, especially the minimum number of times a word must appear to be considered. Also, improvements can be done for rare words of the dataset. One idea could be to replace the `UNK` word embeddings by a linear combination of other embeddings, words close to this rare

ones. Character-level features already help to have a better representation of out-of-vocabulary words but trying to improve out-of-vocabulary word embeddings worth the try.
Without the use of task-specific resources, the proposed neural network seems to be more robust than the CRF baseline, fine-tuned with feature engineering. The dataset contains OCR issues and has other particularities, as detailed in section 2.2. According to the difference in F1 scores between the CRF model and the BiLSTM network, the latter is doing a much better job at generalizing. This can be due to the window size, which isn't fixed for LSTMs compared to the CRF, set to a size of two. The CRF features try to capture the inner-structure of a word, same as the character-level features in the neural network approach. The results show that when using solely character-level features for word representations, the F1 score is similar to the CRF baseline.
The model perform well on the three task, namely for the tags with the most important frequency in the dataset. If more time could have been devoted to this project, a more deep and comprehensive analysis of the results would have been done.

This project has been done in the context of a Master semester project at EPFL. During an entire semester, I was able to dive into this problem of reference mining and the humanities dataset linked to it. The goal of the project was to propose an artificial neural network for reference mining dealing with domain-specific texts. A two-steps approach was applied: first understand the data, the problem and the CRF model, then move to neural networks. Each phase took a half of the semester. In order to comprehend the tasks, the entire corpus of humanities publication was used: not only the lines containing references but the entire text. This data was inspected with `pandas`, a popular python data analysis library. The *Jupyter Notebooks* containing all this initial analysis phase are stored in the GitHub repository of this report[23]. For the second phase, the initial work was to assemble a basic knowledge about artificial neural networks and their applications for similar NLP tasks, namely NER and POS, through scientific publications and several blog posts [4, 27, 26, 1]. Keras was chosen for the implementation of the code and reveals to be a pleasant API to start with neural networks, with a very active community around it. After having defined all models and the different architectures to test, an EPFL's cluster was used for training and testing the neural networks.

As mentioned above, future work with this task may consider improving word representations through better word embeddings[24] or including the predictions of one model as features of another one. A deeper inspection of the model can be carried out, namely to capture differences among the random, static and non-static word embeddings or clustering the output of each inner layer. Considering the model architecture and its implementation, it would be interesting

[23]Private repository due to the use of the entire text of publications.
[24]Example of possible paths to test: `http://ruder.io/word-embeddings-2017/`.

to explore the CRF predicting layer. Indeed, while building the model, multiple CRF implementations have been tried out and the results weren't always homogeneous. At the end, the Keras's implementation was used, but fine-tuning and other implementation may be tested out[25]. In order to assess more deeply the effectiveness of multi-task learning, four models, with the same architecture, could be fine-tuned for each of the three tasks and the multi-task approach, and then compare their results. This is probably the major flaw of this project: using the same fine-tuned parameters for the three tasks and the multi-task learning. On the neural network architecture, attention mechanism could be investigated.

6 Conclusion

In this project, an end-to-end neural network model is proposed for reference mining on humanities publications. Consistent with other studies about NER and POS, the best model is a BiLSTM-CRF network with word embeddings and character-level features. This combination of word embeddings and their characters for word representations reveals to be important and robust against the specificities of the data at hand, especially OCR issues.
The model outperforms a CRF baseline, based on hand-crafted features, with a F1 score +5.7% higher on average. Combining correlated information between tasks in a multi-task learning fashion hasn't improved the prediction, but further work is required to assert it.

[25] Another CRF layer to consider: `https://github.com/Hironsan/keras-crf-layer/blob/master/crf.py`

A Specific tags

Exhaustive list of the all the specific tags with their consolidated version, based on a previous work [7]:

1. 'abbreviatedtitle': 'title', Title
2. 'abbreviation': 'abbreviation', Abbreviation such as Ivi, Ibid, Cit, Cf, etc.
3. 'appendix': 'ref', Appendix
4. 'archivalfond': 'archivalreference', Archival record group
5. 'archivalreference': 'archivalreference', Archival reference
6. 'archivalseries': 'archivalreference', Archival series or sub series (document collections)
7. 'archivalunit': 'archivalreference', Archival unit (box, register)
8. 'archive': 'archive_lib', Archive
9. 'attachment': 'attachment', Attachment
10. 'author': 'author', Author
11. 'box': 'box', Box
12. 'cartulation': 'cartulation', Cartulation such as c. 12.
13. 'cedola': 'ref', Cedola (another form of reference)
14. 'century': 'date', Century for a date, such as XIX century.
15. 'chapter': 'ref', Chapter of a book
16. 'citation': 'ref', Citation
17. 'codex': 'archivalreference', Codex
18. 'column': 'column', Column
19. 'conjunction': 'conjunction', Conjunction such as in.
20. 'curator': 'author', Curator
21. 'date': 'date', Date
22. 'editor': 'author', Editor
23. 'fascicolo': 'folder', Dossier
24. 'filza': 'filza', Gathering
25. 'folder': 'folder', Folder
26. 'foliation': 'foliation', Foliation such as f. 12.
27. 'fond': 'archivalreference', Archival record group.
28. 'library': 'archive_lib', Library
29. 'mazzo': 'ref', Group of documents
30. 'notary': 'archivalreference', Notary (name of a person)
31. 'note': 'numbered_ref', Note
32. 'numbering': 'numbered_ref', Numbering such as n. 12.
33. 'other': '', Generic placeholder
34. 'pagination': 'pagination', Pagination such as p. 12.
35. 'parchment': 'ref', Parchment
36. 'period': 'date', Period such as 1950-1995.
37. 'protocollo': 'ref', Registration of a document
38. 'parte': 'ref', Law

39. 'publicationnumber': 'publicationnumber-year', Issue and volume number such as 12(3).
40. 'publicationnumber-year': 'publicationnumber-year', Issue and volume number, with year such as 12(3), 1990.
41. 'publicationplace': 'publicationplace', Place of publication
42. 'publicationspecifications': 'publicationspecifications', Other specifications linked with a publication
43. 'publicationyear': 'year', Publication year
44. 'publisher': 'publisher', Publisher
45. 'registry': 'registry', Register
46. 'responsible': 'author', Responsible (similar to Curator)
47. 'series': 'series', Archival document series
48. 'table': 'ref', Table
49. 'title': 'title', Title
50. 'tomo': 'tomo', Volume
51. 'topicdate': 'publicationplace', Place of publication
52. 'voce': 'ref', Entry e.g. in a dictionary.
53. 'volume': 'volume', Volume
54. 'website': 'ref', Website
55. 'year': 'year', Year
56. ": " no tag!

B CRF Baseline

B.1 CRF Features

Exhaustive list of features used for each token[26]. The same set of features was used for the three tasks.

For the token `Empire`, the features are:

```
Lowercase: empire
Token shape: Ullllll
Token shape degenerated: Ul
Token type: AllLetter
Prefix 1: e
Prefix 2: em
Prefix 3: emp
Prefix 4: empi
Suffix 1: e
Suffix 2: re
Suffix 3: ire
Suffix 4: pire
Two digits: no
Four digits: no
Parentheses: no
Alphanumeric: yes
Digits and /: no
Digits and ,: no
Digits and .: no
Uppercase and .: no
Initial uppercase: yes
All upercase: no
All lowercase: no
All digit: no
All other: no
Uppercase in: yes
Lowercase in: yes
Letter in: yes
Digit in: no
Symbol in: no
Abbreviation: no
Abbreviation 2: no
Roman number: no
Roman inside: no
Interval: no
```

[26] Code available at https://github.com/dhlab-epfl/LinkedBooksReferenceParsing/.

B.2 CRF Results

Classification report for each task.

- For Task 1, the model achieves a F1 score of `82.63%` on validation data.

	precision	recall	f1-score	support
abbreviation	0.000000	0.000000	0.000000	87
archivalreference	0.430939	0.237805	0.306483	328
archive_lib	0.636364	0.823529	0.717949	17
attachment	0.000000	0.000000	0.000000	0
author	0.862993	0.878629	0.870741	4581
box	1.000000	1.000000	1.000000	6
cartulation	0.000000	0.000000	0.000000	10
column	1.000000	1.000000	1.000000	6
conjunction	0.535484	0.691667	0.603636	120
date	0.600000	0.473684	0.529412	19
filza	0.833333	0.214286	0.340909	70
numbered_ref	0.000000	0.000000	0.000000	87
o	0.422306	0.488760	0.453109	1379
pagination	0.939626	0.957539	0.948498	1154
publicationnumber-year	0.816794	0.804511	0.810606	665
publicationplace	0.908199	0.833441	0.869215	1555
publicationspecifications	0.277523	0.367781	0.316340	329
publisher	0.754875	0.578442	0.654985	937
ref	0.312500	0.416667	0.357143	36
registry	0.538462	1.000000	0.700000	35
series	0.705882	0.558140	0.623377	43
title	0.884157	0.901848	0.892915	13744
tomo	0.538462	0.424242	0.474576	33
volume	0.575290	0.444776	0.501684	335
year	0.892928	0.843848	0.867694	1601
avg / total	0.828812	0.827612	0.826328	27177

- For Task 2, the model achieves a F1 score of `71.04%` on validation data.

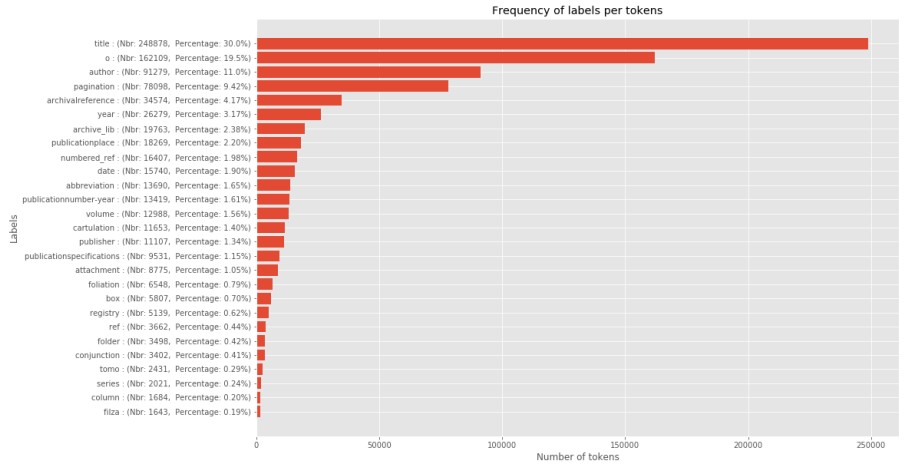
	precision	recall	f1-score	support
b-meta-annotation	0.611650	0.450000	0.518519	280
b-primary	0.285714	0.044444	0.076923	45
b-secondary	0.649073	0.584082	0.614865	779
e-meta-annotation	0.777778	0.668942	0.719266	879
e-primary	0.360000	0.053892	0.093750	167
e-secondary	0.693669	0.643714	0.667759	1583
i-meta-annotation	0.684117	0.686059	0.685087	8457
i-primary	0.608856	0.611111	0.609982	270
i-secondary	0.766575	0.772402	0.769477	13682
o	0.400000	0.570048	0.470120	1035
avg / total	0.713245	0.711042	0.710366	27177

- For the Task 3, the model achieves a F1 score of `92.50%` on validation data.

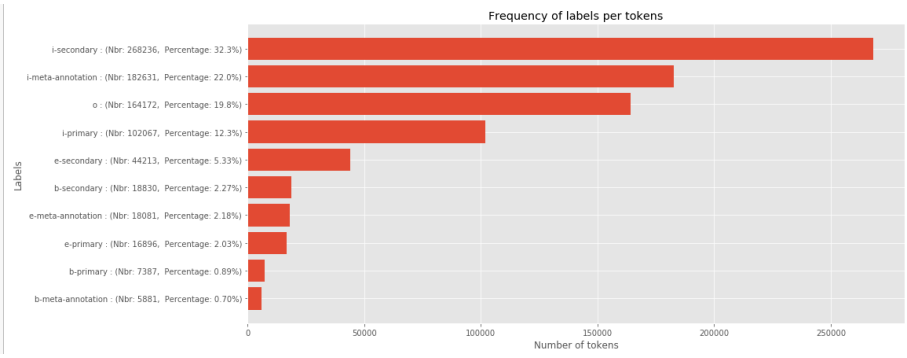
	precision	**recall**	**f1-score**	**support**
b-r	0.793782	0.693841	0.740454	1104
e-r	0.894129	0.811354	0.850733	1145
i-r	0.957633	0.961160	0.959393	23893
o	0.381711	0.439614	0.408621	1035
avg / total	0.926368	0.924127	0.924946	27177

C Training Data

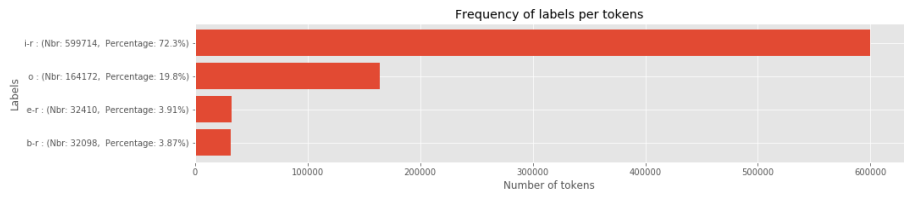
- Tags frequency on the training data for Task 1.



- Tags frequency on the training data for Task 2.



- Tags frequency on the training data for Task 3.



D Model Selection Results

Results, for task I, for all model selection phase with all columns:

Word Embeddings	Character Features	Inner layer	Output	Testing F1	Testing Accuracy	Testing Recall	Training Accuracy	Training Recall	Training F1	Avg. Epoch duration [s]	Best epoch
train word2vec	BiLSTM	BiLSTM	crf	0.88363	0.89163	0.88353	0.87534	0.87679	0.87486	1686.773063	12.0
train word2vec		BiLSTM	crf	0.87359	0.87977	0.87453	0.88189	0.88399	0.88167	564.478748	20.0
train word2vec	CNN	BiLSTM	crf	0.87298	0.87952	0.87346	0.87583	0.87782	0.87559	566.260964	16.0
word2vec	BiLSTM	BiLSTM	crf	0.86847	0.87700	0.86592	0.85467	0.85590	0.85444	1671.689170	14.0
word2vec	CNN	BiLSTM	crf	0.86157	0.86699	0.86407	0.84467	0.84829	0.84456	560.319535	23.0
train word2vec	BiLSTM	BiLSTM	softmax	0.86122	0.86889	0.86190	0.88963	0.89200	0.88933	1530.886140	6.0
train	BiLSTM	BiLSTM	crf	0.86101	0.87254	0.85918	0.88415	0.88509	0.88393	1667.915859	8.0
word2vec	BiLSTM	BiLSTM	softmax	0.85959	0.87240	0.86135	0.85730	0.86058	0.85574	1524.700370	23.0
train		BiLSTM	crf	0.85879	0.87081	0.85523	0.91237	0.91256	0.91219	552.946363	20.0
train	CNN	BiLSTM	crf	0.85560	0.87130	0.84894	0.88304	0.88471	0.88332	566.693413	8.0
train word2vec	CNN	BiLSTM	softmax	0.85466	0.86372	0.85458	0.90432	0.90510	0.90327	413.063867	10.0
train word2vec		BiLSTM	softmax	0.85410	0.86826	0.84998	0.91185	0.91230	0.91132	385.067781	12.0
word2vec		BiLSTM	crf	0.84952	0.86064	0.84778	0.83333	0.83680	0.83331	503.713275	16.0
word2vec	CNN	BiLSTM	softmax	0.84446	0.85160	0.84856	0.83219	0.83836	0.82904	399.786669	14.0
train	BiLSTM	BiLSTM	softmax	0.83996	0.85443	0.83557	0.94198	0.94183	0.94132	1527.254355	20.0
word2vec		BiLSTM	softmax	0.83908	0.85090	0.84166	0.82830	0.83385	0.82437	352.757504	23.0
train	CNN	BiLSTM	softmax	0.83609	0.85194	0.83136	0.91946	0.92001	0.91903	409.274326	7.0
	BiLSTM	BiLSTM	crf	0.83063	0.83672	0.83421	0.81775	0.82321	0.81831	1615.704910	24.0
train		BiLSTM	softmax	0.83061	0.84986	0.82167	0.93225	0.93220	0.93184	385.171326	11.0
	BiLSTM	BiLSTM	softmax	0.82054	0.82804	0.82702	0.81973	0.82591	0.81526	1459.099599	24.0
	CNN	BiLSTM	crf	0.78226	0.79636	0.78236	0.75040	0.76081	0.75091	568.872718	19.0
	CNN	BiLSTM	softmax	0.75283	0.75799	0.76804	0.73554	0.75248	0.73034	406.392566	24.0

E Model Fine-tuning Results

Results for the fine-tuning phase of the BiLSTM-CRF model with word embeddings and character-level features. The reported scores concern Task I.

Batch	Dropout	LSTM	Testing F1	Testing Accuracy	Testing Recall	Training Accuracy	Training Recall	Training F1	Avg. Epoch duration [s]	Best epoch
100	0.5	100	0.89090	0.89643	0.89147	0.89426	0.89428	0.89326	1274.908107	9.0
30	0.5	100	0.88963	0.89730	0.88830	0.88629	0.88642	0.88556	2153.929912	5.0
70	0.5	100	0.88948	0.89466	0.89140	0.90592	0.90577	0.90528	1350.567238	13.0
70	0.7	100	0.88608	0.89411	0.88376	0.87323	0.87476	0.87341	1353.856037	17.0
100	0.2	100	0.88506	0.89308	0.88525	0.89583	0.89635	0.89475	1269.877417	3.0
100	0.7	100	0.88407	0.89398	0.88279	0.87584	0.87707	0.87576	1271.983111	18.0
100	0.5	40	0.88359	0.89171	0.88123	0.88055	0.88250	0.88051	1350.769267	12.0
70	0.2	100	0.88194	0.88932	0.88308	0.91763	0.91724	0.91681	1357.579706	5.0
30	0.2	100	0.88075	0.88924	0.88159	0.88078	0.88228	0.87966	2164.874749	1.0
30	0.2	40	0.88001	0.88756	0.88146	0.90113	0.90166	0.90059	2122.470629	5.0
70	0.5	40	0.87968	0.88900	0.87884	0.88960	0.89130	0.88974	1467.639799	17.0
70	0.2	40	0.87891	0.88349	0.88136	0.87187	0.87382	0.87117	1469.420238	2.0
30	0.5	40	0.87839	0.89039	0.87424	0.87555	0.87705	0.87519	2062.517841	10.0
100	0.2	40	0.87783	0.88749	0.87835	0.88781	0.88848	0.88719	1348.441558	4.0
30	0.2	20	0.87630	0.88369	0.87537	0.90289	0.90249	0.90216	2099.655616	8.0
30	0.7	100	0.87323	0.88223	0.87077	0.85971	0.86081	0.85906	2155.571821	9.0
100	0.5	20	0.87234	0.87945	0.87275	0.87036	0.87133	0.86938	1330.585033	16.0
70	0.5	20	0.87170	0.88170	0.86983	0.87025	0.87275	0.87067	1462.542336	16.0
100	0.7	40	0.86808	0.87654	0.86815	0.85105	0.85308	0.85022	1357.834269	23.0
70	0.7	40	0.86800	0.87618	0.86695	0.85047	0.85499	0.85122	1483.728887	21.0
70	0.2	20	0.86787	0.87838	0.86575	0.88982	0.89144	0.89003	1459.812313	6.0
30	0.5	20	0.86709	0.87336	0.86715	0.86216	0.86536	0.86263	2050.451974	14.0
100	0.2	20	0.86702	0.87638	0.86549	0.87705	0.87877	0.87624	1349.298403	5.0
30	0.7	40	0.86289	0.86878	0.86420	0.83810	0.84138	0.83756	2058.818475	14.0
100	0.7	20	0.84599	0.85077	0.85024	0.82544	0.83172	0.82582	1356.890259	21.0
70	0.7	20	0.83684	0.85008	0.83456	0.82762	0.83334	0.82851	1465.242993	23.0
30	0.7	20	0.83547	0.84633	0.83579	0.81393	0.82183	0.81518	2118.303878	19.0

F Multi-task learning

Classification report for each task when training in a multi-task learning architecture.

- For Task 1, the model achieves a F1 score of `87.56%` on testing data.

	precision	recall	f1-score	support
foliation	0.6000	1.0000	0.7500	12
o	0.2500	0.3790	0.3013	752
filza	1.0000	1.0000	1.0000	3
box	0.8793	0.4636	0.6071	110
date	0.3500	0.7241	0.4719	58
publicationspecifications	0.4939	0.2177	0.3022	372
registry	0.8889	0.1039	0.1860	154
ref	0.4000	0.6667	0.5000	3
publicationnumber-year	0.6355	0.8096	0.7120	646
tomo	0.9153	0.2411	0.3816	224
folder	0.3830	0.6429	0.4800	28
attachment	0.0000	0.0000	0.0000	0
conjunction	0.5444	0.7313	0.6242	134
archivalreference	0.8098	0.8535	0.8311	853
year	0.9250	0.7932	0.8540	2036
publicationplace	0.9294	0.9014	0.9152	1592
column	0.0000	0.0000	0.0000	0
volume	0.5535	0.6354	0.5916	277
pagination	0.9367	0.9614	0.9489	1139
publisher	0.9345	0.8108	0.8683	1443
series	0.9646	0.7124	0.8195	153
archive_lib	0.8857	0.9029	0.8942	103
title	0.9243	0.9504	0.9371	15560
numbered_ref	0.2353	0.1667	0.1951	120
abbreviation	0.9535	0.5541	0.7009	148
cartulation	0.5000	1.0000	0.6667	8
author	0.9430	0.9101	0.9263	4948
avg / total	0.8880	0.8747	0.8756	30876

- For Task 2, the model achieves a F1 score of `80.10%` on testing data.

	precision	recall	f1-score	support
e-secondary	0.8648	0.8305	0.8473	1717
i-meta-annotation	0.7487	0.8634	0.8020	9981
o	0.1618	0.5831	0.2533	427
e-primary	0.6334	0.6567	0.6448	300
e-meta-annotation	0.8369	0.8124	0.8245	1061
b-meta-annotation	0.6250	0.6609	0.6425	348
i-secondary	0.9128	0.7570	0.8276	14357
b-secondary	0.7493	0.6279	0.6833	852
i-primary	0.7632	0.7778	0.7704	1728
b-primary	0.8161	0.6762	0.7396	105
avg / total	0.8249	0.7903	0.8010	30876

- For the Task 3, the model achieves a F1 score of `92.97%` on testing data.

	precision	**recall**	**f1-score**	**support**
e-r	0.9155	0.5458	0.6839	1310
i-r	0.9659	0.9548	0.9604	27834
o	0.1780	0.5995	0.2745	427
b-r	0.7900	0.6920	0.7377	1305
avg / total	0.9455	0.9215	0.9297	30876

References

[1] Josh Patterson Adam Gibson, Chris Nicholson. Deep learning for java, 2017.

[2] L. T. Anh, M. Y. Arkhipov, and M. S. Burtsev. Application of a Hybrid Bi-LSTM-CRF model to the task of Russian Named Entity Recognition. *ArXiv e-prints*, September 2017.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994.

[4] Jason Brownlee. Machine learning mastery with keras, 2017.

[5] François Chollet et al. Keras. `https://github.com/keras-team/keras`, 2015.

[6] Giovanni Colavizza and Frédéric Kaplan. On Mining Citations to Primary and Secondary Sources in Historiography, 2015.

[7] Giovanni Colavizza and Matteo Romanello. Annotated References in the Historiography on Venice: 19th–21st centuries, 2017.

[8] Giovanni Colavizza, Matteo Romanello, and Frédéric Kaplan. The references of references: a method to enrich humanities library catalogs with citation data. *International Journal on Digital Libraries*, Mar 2017.

[9] Hong-Jie Dai, Po-Ting Lai, Yung-Chun Chang, and Richard Tzong-Han Tsai. Enhancing of chemical compound and drug name recognition using representative tag scheme and fine-grained tokenization. *Journal of Cheminformatics*, 7(1):S14, Jan 2015.

[10] Cicero Dos Santos and Maira Gatti de Bayser. Deep convolutional neural networks for sentiment analysis of short texts. 08 2014.

[11] Cícero Nogueira dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1818–1826. JMLR.org, 2014.

[12] Y. Gal and Z. Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *ArXiv e-prints*, December 2015.

[13] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm networks. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 4, pages 2047–2052 vol. 4, July 2005.

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

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[16] Rajat Monga Jeffrey Dean and Google Research. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[17] Y. Kim. Convolutional Neural Networks for Sentence Classification. *ArXiv e-prints*, August 2014.

[18] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-Aware Neural Language Models. *ArXiv e-prints*, August 2015.

[19] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[20] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *CoRR*, abs/1603.01360, 2016.

[21] Wang Ling, Chris Dyer, Alan W. Black, and Isabel Trancoso. Two/too simple adaptations of word2vec for syntax problems. In *HLT-NAACL*, pages 1299–1304. The Association for Computational Linguistics, 2015.

[22] Xuezhe Ma and Eduard H. Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *CoRR*, abs/1603.01354, 2016.

[23] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[24] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[25] Naoaki Okazaki. Crfsuite: a fast implementation of conditional random fields (crfs), 2007.

[26] Christopher Olah. Nlp town - company blog, 2017.

[27] Yves Peirsman. Nlp town - company blog, 2017.

[28] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, CoNLL '09, pages 147–155, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[29] Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. *CoRR*, abs/1707.09861, 2017.

[30] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

[31] M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, Nov 1997.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[33] X. Zhang, J. Zhao, and Y. LeCun. Character-level Convolutional Networks for Text Classification. *ArXiv e-prints*, September 2015.