Noname manuscript No.
(will be inserted by the editor)

Neural ParsCit: A Deep Learning Based Reference String Parser

Animesh Prasad · Manpreet Kaur · Min-Yen Kan

Received: date / Accepted: date

Abstract We present a deep learning approach for the core digital libraries task of parsing bibliographic reference strings. We deploy the state-of-the-art Long Short-Term Memory (LSTM) neural network architecture, a variant of a recurrent neural network (RNN) to capture long-range dependencies in reference strings. We explore word embeddings and character-based word embeddings as an alternative to hand-crafted features. We incrementally experiment with features, architectural configurations, and the diversity of the dataset. Our final model is an LSTM-based architecture, which layers a linear-chain Conditional Random Field (CRF) over the LSTM output. In extensive experiments in both English in-domain (computer science) and out-of-domain (humanities) test cases, as well as multilingual data, our results show a significant gain ($p < 0.01$) over the reported state-of-the-art CRF-only based parser.

Keywords reference string parsing · sequence labeling · CRF · LSTM

Animesh Prasad
School of Computing, National University of Singapore
Computing 1, 13 Computing Drive, Singapore 11741
Tel.: +65-90525651
E-mail: animesh@comp.nus.edu.sg

Manpreet Kaur
School of Computing, National University of Singapore
Computing 1, 13 Computing Drive, Singapore 11741
Tel.: +65-84106281
E-mail: manpkaur@comp.nus.edu.sg

Min-Yen Kan
School of Computing, National University of Singapore
Computing 1, 13 Computing Drive, Singapore 11741
Tel.: +65-6516-1885
Fax: +65-6779-4580
E-mail: kanmy@comp.nus.edu.sg

1 Introduction

Scientific papers cite scholarly work to acknowledge their past contribution though reference strings. These reference strings allow a reader to accurately conceptualize the proposed work with respect to prior work. Importantly, it facilitates the understanding of the scientific discourse surrounding a research topic by identifying key ideas and incremental advances, when coupled with textual citation analysis. Impact assessment of both individual articles and scientists, and larger aggregates, such as scientific fields and academic institutions are premised on this basic cornerstone task. Accurate extraction and parsing of reference strings to identify field tags – such as *Author*, *Journal*, *Title*, among others – is needed by many downstream tasks to avoid propagation of error. Preprocessing is first performed to obtain reference strings from the digital Portable Document Format (PDF), the *de facto* standard in many fields for transmitting scholarly discoveries. In the workflow for this task, the bibliographic section is first extracted from the files, then the reference strings recovered. Hand-compiled features are generated from the data, and a trained sequence labeling learned model is used to induce the labels (fields) per word.

Academic disciplines have adopted different styles of citing references in their research work. For instance, the APA format is commonly used in the scientific discipline, while the MLA format is employed more often in humanities. However, even with stylized conventions, authors occasionally apply these styles inconsistently or conflictingly, even within the scope of a single article, making the task of reference string parsing challenging. reference strings also contain mentions of named entities and new vocabulary are continually introduced, further exacerbating errors for approaches based on dictionary-based lookup.

Still, the reference strings largely obey conventions that greatly assist with the parsing task. Fields are mostly contiguous and separated by delimiters. However, an approach purely based on splitting the string by delimiter occurrences fails as the delimiters themselves are used by many fields. Encoding long-range dependencies such as the position of the current word with respect to the previous and future sequence of entities become important cues to model the task well. Hence, the parsing task is often modeled as sequence labeling problem, where each token of the reference string is labeled as one of the possible fields. Models such as the Hidden Markov Model (HMM) (Rabiner and Juang, 1986) and the linear chain Conditional Random Field (CRF) (Lafferty et al, 2001) are widely used for such sequential modeling tasks in natural language processing. HMM and CRF are both probabilistic graphical models which model long range dependencies by making a Markovian assumption that a limited local context is sufficient to make the correct decision. Many existing techniques for reference string tagging also use these graphical models. These are numerous examples of such implementations in both academic and commercial implementations, such as ParsCit (Councill et al, 2008), CiteSeerX (Giles et al, 1998), and Mendeley[1].

2 Related Work

Our work centers on the application of deep learning to the reference string parsing task. As such, we first review the prior work on state-of-the-art reference string parsers and their shortcomings, with particular attention to the ParsCit model, which we build upon and compare with. Afterwards, we touch upon recent developments in deep learning and how the shortcomings of the existing paradigm motivate us to use it to close the performance gap.

ParsCit (Councill et al, 2008) is an open-source CRF based implementation which labels (classifies) all words of a reference string into one of the 13 disjoint fields (classes). We focus on its primary function as a toolkit for reference string parsing. In ParsCit, 23 human-engineered features are extracted, for each word to be classified. These features include capitalization, punctuation, numeric type, the length of the word, the location of the word within the reference string, the presence of substring in the word and a pair of 1, 2, 3 and 4 character $n-$grams per word, which are the prefix and suffix portions of the word. All these features are derivable from the surface reference string. Apart from these features, ParsCit also uses external knowledge in the form of six dictionary features for publisher names, place names, surnames, female and male names, and months. These features correspond to four fields namely *Author*, *Editor*, *Location* and *Date* which are similar to named entities. These hand-engineered features are fed to a CRF model which extracts their values for the current token to be tagged as well as its neighboring, contextual words. This method incorporates neighboring signals (in a fixed context window) such as the occurrences of previous or future punctuation marks, which we have already mentioned as important in tagging the current word.

We observe that some of these features – such as the token position and word length – are derivable from the raw text itself, and could be hypothetically learned using an unsupervised approach. Similarly, features which are empirically decided – such as prefix character $n-$grams of up to 4 characters – give rise to the need of more powerful models that can extract these information from raw characters.

We also observe that such relevant reference string parsers makes extensive use of dictionary-based features. In ParsCit, these account for a significant fraction (26% of features correlating directly to four classes). Such features are fragile, being domain and language-specific, and not portable in generalizing the performance of the model to new, unseen domains and language data. We question whether the prior work's performance would generalize well to the presence of out-of-vocabulary words present in foreign language and scholarly works in unseen domains. These questions directly challenge the generalization ability of the prior legacy work. A robust model should be able to use other unambiguous signals to decide labels for such words.

Reference string processing has been well-studied by many research teams. Although the identified shortcomings belong to few systems, we feel that they are representative of the field. We find that other recent work also largely goes in the direction of feature engineering and selection – including finding more lucid, hand-crafted features for CRFs (Romanello et al, 2009; Kern and Klampfl, 2013), (Tkaczyk et al, 2015) or the use a template-based aligner (Chen et al, 2012) – without addressing the central issue of finding an optimal representation for the important surface level signals, in contrast to hand-coding it from human intuition. For example, CERMINE (Tkaczyk et al, 2015) uses extensive features fed to CRF, achieving 93.3% F1 (on a smaller set of 7 classes) over a mixed dataset including the commonly-studied Cora and PubMed reference strings. Similarly, (Kern and Klampfl, 2013) uses additional font information. All these approaches falls under the class of carefully designed features for a CRF model which we intend to move away from and introduce a model that is empirically data-driven.

None of these techniques use the vast amount of unlabeled (or noisy) data available, relying on the supervised learning paradigm. As the amount of human-annotated reference strings will always be limited, ideal techniques should be scalable to take advantage of the vast, unlabeled body of reference strings. They should also be robust when faced with limited data. BibPro (Chen et al, 2012) takes a step towards in this direction, utilizing noisy data to train and test their system, but unfortunately their results do not compare

[1] `http://www.mendeley.com/`

with standard datasets; their authors performed evaluation only on their noisy data set, leaving readers to extrapolate performance on other data.

A final shortcoming of the field is that the evaluations have been largely limited to the Cora dataset, which is homogeneous (English computer science articles) and unrepresentative of the multilingual, multidisciplinary scholastic reality. As the complexity of the problem is introduced by the variability of the citation styles, we assume a dataset with citations from various domains and styles should test the applicability of the proposed technique best. To facilitate direct comparison on annotated data that exhibit these variations, we use ParsCit as our baseline.

Recently there have been many advancements in deep learning (Schmidhuber, 2015), spearheaded by general- purpose graphics cards which speed up matrix and vector operations. Deep learning based techniques have shown success in various fields for different tasks in computer vision, speech and natural language processing (NLP). Using data to learn representations and train generic deep neural architectures by back propagation has been a central paradigm (Rumelhart et al, 1985) of the field. Such techniques rely less on handcrafted features, given sufficient data that allows the machine models to identify a good abstract internal representation of the data (LeCun et al, 2015). With recent advances in training (Rifai et al, 2011), models that better leverage structural regularities in the data (Mohamed et al, 2012; Krizhevsky et al, 2012) and a more thorough understanding of their internal workings, researchers have started testing deep neural architectures as alternative to domain-specific modeling techniques.

In the NLP domain, deep neural network based models obtained comparable results to state-of-the-art, human-engineered feature-based models on part-of-speech tagging, chunking, named entity recognition, and semantic role labeling (Collobert et al, 2011). Such tasks are all sequence labeling tasks, in which the system assigns labels to variable-length sequences. Recent research has focused on harder sequence modeling tasks such as grammatical error detection, question answering and machine translation. For NLP tasks, the advantage of eschewing task-specific features while obtaining good results have been often facilitated by pre-trained word embeddings. Such word embeddings are generic, numeric vector representation of words that capture semantic and syntactic information through distributed representation of contexts. These word embeddings can be trained on a large corpus of unlabeled data. Word2vec (Mikolov et al, 2013b) and GloVe (Pennington et al, 2014) are two such proposed techniques for obtaining task-independent vector representations for words using unlabeled text corpora. These word embeddings are used as a replacement for handcrafted features and given as input to the neural network model. The neural network then learns an optimal, task-specific transform from task-independent representations of the words.

When applying deep learning for reference string parsing, we believe training embeddings would be effective, as one key property of the scenario is that there is a vast amount of unlabeled citation strings. These can be used to generate the embeddings, thereby incorporating the knowledge of citation structures without human intervention. A second property is that scholarly reference strings contain a high proportion of named entities and domain-specific words. Due to the arduousness of obtaining such quality data, the amount of supervised data is little, and this affects the quality of learning representations. We believe that learning unsupervisedly from large unlabeled data might be generalize better to unseen cases, even if the labeling is noisy. To effectively address these two properties, we wish to exploit the unlabeled data, while limiting the number of parameters of the learned model to achieve good generalization performance, avoiding overfitting.

3 Formulation

The task of *reference string parsing* refers to transforming reference strings into a machine-readable form, e.g. BibTeX format. One possible solution is to apply sequence labeling to the string's sequence of tokens. We now formally describe the reference string parsing as a sequence labeling problem, and then illustrate the background of sequence labeling methods by reviewing the fundamental models of HMMs, CRFs and the associated deep learning based approaches.

Definition. Given an input sequence, $X = (x_1, ..., x_t, ..., x_T)$ and possible output (tag) sequences $Y = (y_1, ..., y_t, ..., y_T)$, the task is to identify corresponding optimal output sequence, $\hat{Y} = (\hat{y_1}, ..., \hat{y_t}, ..., \hat{y_T})$.

Here, x_t represent the input and y_t represents a variable which can take any of the N possible output labels at time step t respectively. During training we learn use the pair of input-optimal output sequences to learn such a function. A reasonable output can be obtained with a Markov assumption over the sequences, then learning to optimize an objective function over Y and/or X. We discuss these methods and corresponding models below in detail.

3.1 Hidden Markov Models

A basic approach is to use a generative model such as a Hidden Markov Model (HMM). The HMM can be used to obtain the optimal sequence tag by training to maximize the joint log probability as:

$$\hat{Y} = argmax \log p(X, Y) \tag{1}$$

When a generative model like an HMM is used for modeling the joint probability, Viterbi decoding is applied during inference to find the conditional likelihood and hence the most optimal sequence. This difference in training and testing objectives and the potential occurrence of label bias resulting from such maximum likelihood models has led to their replacement by CRFs in many practical implementations.

3.2 Conditional Random Fields

The conditional random field (CRF) model maximizes the conditional probability of the output labels given the observed data:

$$\hat{Y} = argmax \log p(Y|X) \tag{2}$$

Here, the conditional probability, $p(Y|X)$ for the sequence is determined by calculating the product of potential function (ϕ) at each time step as:

$$p(Y|X) = \frac{\prod_{i=1}^{N} \phi(y_{i-1}, y_i, X)}{\sum_{Y \in Y^*} \prod_{i=1}^{N} \phi(y'_{i-1}, y'_i, X)} \tag{3}$$

where N is the maximum number of classes and Y^* is set of all possible label assignments for a given input sequence. The prediction for a string using this model can be achieved by the Forward–Backward algorithm. Though HMM and CRF form a generative–discriminative model pair, CRFs are a better substitute for tasks with less supervision and high class imbalance, as the CRF model invests all of its modeling power to discriminate among classes, whereas HMMs divide their power between class discrimination and generation.

3.3 Deep Learning based Sequence Modeling

A simple feedforward deep neural network with layers $i = 1$ to L is a sequence of operations on input features resulting in non-linear transformations of the input features. For an input feature vector $\mathbf{x}$, layer i performs the operation as:

$$h^i(\mathbf{x}) = f\left(W^i \tilde{h^i}(\mathbf{x}) + b^i\right) \tag{4}$$

where,

$$\tilde{h^i}(\mathbf{x}) = \begin{cases} \mathbf{x}, & \text{if } i = 1 \\ h^{i-1}(\mathbf{x}), & \text{otherwise} \end{cases}$$

$$f(\mathbf{x}) = \begin{cases} \dfrac{\exp(x_k)}{\sum_{j=0}^{dim(\mathbf{x})} \exp(x_j)}, & \text{if } i = L \\ \tanh(\mathbf{x}) = 2\sigma(2\mathbf{x}) - 1 = \dfrac{e^{x_k} - e^{-x_k}}{e^{x_k} + e^{-x_k}}, & \text{otherwise} \end{cases}$$

The $tanh$ function introduces non-linearity on the affine transform of the hidden layer. The nonlinear operation performed in the output layer ($i = L$) is $softmax$ which enables multiclass classification. This is due to the fact that the softmax output can be treated as probability distribution over the target classes (*i.e.*, the normalized class prediction from a simple feedforward network can be treated as the conditional probability $p(y_{(n)}|\mathbf{x})$ for classes $n = 1$ to N). The final layer L uses a softmax function to implement the multiclass classification, while the other layers add flexibility in the modeling by passing their input through a non-linear function (feedforward layers).

One notable shortcoming of the standard feedforward model is that it cannot effectively encode temporal dependencies in generating the probability distribution. This implies that the conditional probabilities output by the model are independent of any previously seen and future samples. This shortcoming makes the standard model impratical for sequence labeling task such as reference string parsing that observes a sequence of tokens (a temporal series).

To address this problem, hybrid techniques have been introduced where the output of the neural model is fed to a CRF or HMM layer, in a stacked fashion. The CRF or HMM then uses the output probability of the neural model as an abstract feature and does the actual classification based on its own objective function.

However, native solutions in the neural modeling paradigm also address these same issues. Neural networks architectures such as recurrent neural networks (RNNs) take the previous and/or future (time steps) predictions into account. The power of RNNs is attributed to its potential to remember output history by connections to previous time steps and use such historical evidence to make decision for current time step as follows:

$$\mathbf{h}(\mathbf{x_t}) = tanh\Big(W_x \mathbf{x_t} + W_h \mathbf{h}(\mathbf{x_{t-1}}) + b\Big) \tag{5}$$

The training criterion (loss function) for such models is usually Cross Entropy (CE), defined as:

$$CE = -\frac{1}{T} \sum_t \sum_n \hat{y}_{t(n)} \log(y_{t(n)}) \tag{6}$$

The model minimizes CE by attempting to classify each of the $\mathbf{x_t}$ correctly, rather than trying to learn the actual distribution of tags over the sequence, as with in the case of a HMM or CRF.

These models can theoretically reach beyond the Markov assumption and can be used to predict each label without joint decoding. However, in practice, learning such long-ranged dependencies require proportionally more training samples, which is infeasible for our task. As a compromise, we can compose a CRF or an HMM layer on top of the RNN output to re-impose the Markov assumption, making it easier for the network to learn with fewer labeled data. Though for stacked models, the overall performance is upper bounded by the performance of the final layer; in practice, the CRF layer still helps. This happens because signals from outputs afar percolate into the Markov blanket as features, enhancing prediction. However, RNNs has their own practical shortcomings (i.e., extreme gradients) which makes it difficult to train reliably. For overcoming this issue, we employ an LSTM, a variant of RNN, which we discuss next.

4 Method

We now discuss the salient points of our proposed technique. We describe the chosen deep learning architecture, the purpose of each model component and the input features to the model.

4.1 Model: Long Short-Term Memory

We explained that long-range dependencies exist in reference strings and need to be appropriately captured. We utilize Long Short-Term Memory (LSTM) networks to do this.

From Eq. (5) we see that the recursive nature of an RNN allows the capture and modeling of information from previous time steps. While theoretically RNNs can capture historical evidence going back to the start of the sequence (*i.e.*, the first input), in practice, RNNs capture limited recent history, failing to fully exploit the benefit of information from previous time steps in long sequences. This is due to the problem of vanishing gradients (Bengio et al, 1994), in which the gradient decreases exponentially between time steps, which leads to the problem of ineffective error propagation to previous time steps (Pascanu et al, 2013). This results in long training times and its difficulty in assigning sufficient contribution to time steps far back in history. The problem of vanishing gradients can be solved by incorporating a memory cell (a temporary store for history; see Hochreiter and Schmidhuber, 1997) into the RNN. Such networks are termed long short-term memory models (LSTMs). When paired with the appropriate gates, a basic memory cell can be trained to retain relevant information from the input (Gers et al, 2000). Fig. 1 shows the schematic diagram of an LSTM cell with its flow of vectors and associated operations in Eq. (7).

$$\begin{aligned}
\mathbf{i_t} &= \sigma(\mathbf{W_{xi}x_t} + \mathbf{W_{hi}h_{t-1}} + \mathbf{W_{ci}c_{t-1}} + \mathbf{b_i}) \\
\mathbf{f_t} &= \sigma(\mathbf{W_{xf}x_t} + \mathbf{W_{hf}h_{t-1}} + \mathbf{W_{cf}c_{t-1}} + \mathbf{b_f}) \\
\tilde{\mathbf{c_t}} &= \tanh(\mathbf{W_{xc}x_t} + \mathbf{W_{hc}h_{t-1}} + \mathbf{b_c}) \\
\mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \tilde{\mathbf{c_t}} \\
\mathbf{o_t} &= \sigma(\mathbf{W_{xo}x_t} + \mathbf{W_{ho}h_{t-1}} + \mathbf{W_{co}c_t} + \mathbf{b_o}) \\
\mathbf{h_t} &= \mathbf{o_t} \odot \tanh(\mathbf{c_t})
\end{aligned} \tag{7}$$

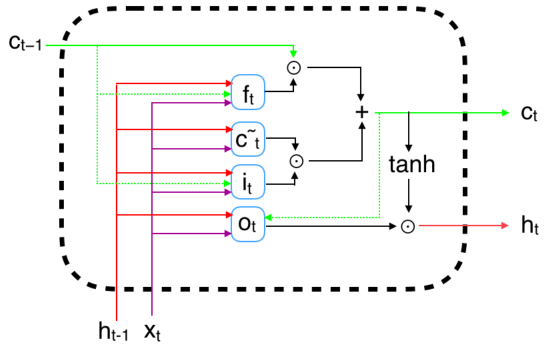

Fig. 1 Basic LSTM cell operations at time step *t*.

Here, the input gate ($\mathbf{i_t}$) and the forget gate ($\mathbf{f_t}$) vectors are calculated based on the current input ($\mathbf{x_t}$) and the output of the previous time step ($\mathbf{h_{t-1}}$). Based on these gates, the memory cell ($\mathbf{c}$) is updated by multiplying the input gate vector with the potential memory content ($\tilde{\mathbf{c_t}}$) and forget gate vector with the previous memory content ($\mathbf{c_{t-1}}$), and then adds the two resultant vectors together. The output gate vector ($\mathbf{o_t}$) is calculated by using the input, new updated memory ($\mathbf{c_t}$) and previous time step's output. Following this, the current time step output ($\mathbf{h_t}$) is calculated.

4.1.1 Bidirectional LSTM (BLSTM)

LSTM networks can encode useful past information, but not future information. Often, successive words provide important information regarding the sequence, particularly true in our task. For example, *Volume* might only come after *Booktitle*; and conversely, if the next label is *Volume*, it is highly likely that the previous label should be *Booktitle*. To capture information from future words, a Bidirectional LSTM (BLSTM) contains two parallel LSTM networks that run in opposing directions with respect to time steps. The LSTM that moves backward over the words can be said to encode information about future words. These two LSTMs are trained independently and can be later merged via a single feed forward layer.

4.1.2 Stacking CRF and BLSTM

The LSTM model can be used to model sequences without the need for joint decoding (Viterbi or Forward–Backward decoding used by HMM or CRF). Learning such parameters without priors or regularization can lead to unstable results, especially given little training data. To lessen these problems, the LSTM output can be directly fed into a CRF to maximize the conditional probability, according to the CRF objective function, re-applying a Markovian assumption. This is achieved by taking the output of the BLSTM as the input to the CRF. Feedforward layers can optionally be inserted between the BLSTM and CRF layer.

4.2 Features

We eschew domain- and language- dependent features to avoid fragility in our model, instead choosing to capture only fundamental features based on characters and domain-general word information.

To capture semantic and syntactic word information, we employ two complementary means to generate word representation:

1. Word Embeddings, also known as distributed word representations, are used as one of the word-level features in our model. We use word2vec for creating distributed word representation. Word2vec can be configured to use either Skip-Gram (SG) or Continuous Bag of Words (CBOW), along with negative sampling and hierarchical softmax to train word vectors. Fig. 2 (Mikolov et al, 2013a) shows the CBOW and SG configurations. In CBOW, given an input context, the model tries to predict the word; while in SG, the context is predicted, given an input word over a fixed vocabulary V.

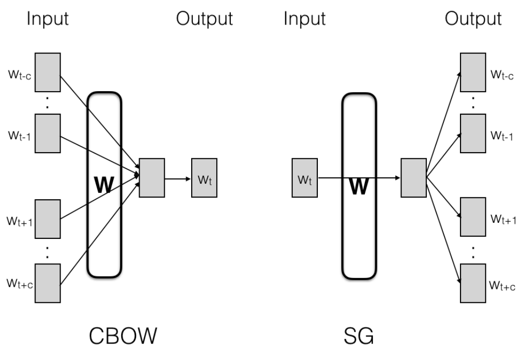

Fig. 2 The word2vec model in CBOW and SG configurations.

The objective for both SG and CBOW models is to maximize the average log probability. Eqs. (8) and (9) give the training objectives of SG and CBOW, respectively, for a given context c.

$$argmax\Big(\sum_{t=1}^{T}\sum_{-c\le j\le c, j\ne 0}\log p(w_{t+j}|w_t)\Big) \tag{8}$$

$$argmax\Big(\sum_{t=1}^{T}\sum_{-c\le j\le c, j\ne 0}\log p(w_t|w_{t+j})\Big) \tag{9}$$

The conditional probability in these training objectives captures the co-occurrence of the words and uses that to maximize the log probability. Once the model is trained, the projection matrix, shown in Fig. 2 as $\mathbf{W}$, is taken as the resultant word embedding.

The training process is shown below. The projection matrix $\mathbf{W}$ (right hand side) converts the traditional, one-hot representation of the word to a vector (middle column vector). In other words, each row of the $\mathbf{W}$ matrix corresponds to the word vector of a specific word in the fixed, indexed vocabulary. Hence adding this as the first layer (referred to as the embedding layer) is equivalent to a lookup table.

$$\begin{bmatrix} x_{11} \; x_{12} \cdots & x_{1n} \\ x_{21} \;\cdots\cdots & \vdots \\ \vdots \;\;\cdots\cdots & \vdots \\ x_{d1} \;\cdots\cdots & x_{|V|n} \end{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}^{T} = \begin{bmatrix} x_{k1} \; x_{k2} \; \ldots \; x_{kn} \end{bmatrix} = \mathbf{W_{(k)}} (10)$$

Word embeddings are sometimes termed "pre-trained", as they are trained offline on unlabeled text, independent of the task at hand. Pre-learned embeddings trained on billions of words are available online.

2. Character-based Word Embeddings. Characters contain important information regarding morphology and word formation, relevant to many tasks. For reference string parsing, the presence and identity of punctation marks – such as commas, hyphens and numbers – can heavily influence the class of the token. Other classes of characters can also yield key evidence. For instance, a four-letter number is likely to be a *Date* and a hyphenated numeric field is likely to be *Pages*. To capture such character-level information, we use character-based word embeddings, where the characters of a word are individually represented as numeric vectors that have been optimized by a trained long short-term memory (LSTM) neural network. While the vectors start randomly initialized, the post-trained resultant embeddings encode meaningful patterns across words. For example, in part-of-speech tagging, character-based embeddings can learn similar vector representations for words ending in "-ly" that are potential adverbs.

We use a bidirectional LSTM to create character-based word embeddings. Formally, for a word $w = c_1, .., c_i, .., c_C$, the vector representation $\mathbf{x_{c_i}}$ of each character is passed sequentially through the LSTM cell, which performs a series of vector and matrix operations resulting in the output vector ($\mathbf{E_{x_{c_1},...,x_{c_i}}}$). The vector represents all the characters fed to the network up to i. Similarly, character vectors are fed in reverse order from T to 1 in another independent network. The resultant final vectors from both independent LSTMs are concatenated to form an effective character-based word embedding $[\mathbf{E_{x_{c_1},...,x_{c_T}}}, \mathbf{E'_{x_{c_T},...,x_{c_1}}}]$.

The resultant character-based word embedding for w is then concatenated with the respective word embedding from the word embeddings dictionary obtained from word2vec (projection matrix $\mathbf{W}$).

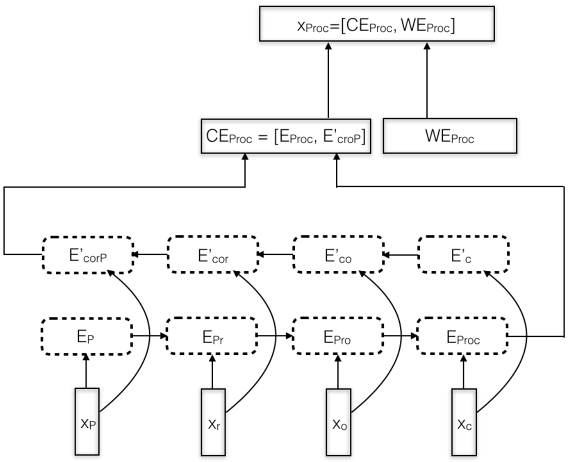

Fig. 3 An unrolled representation of bidirectional LSTM generating character-based word embedding and concatenation with the word embeddings obtained from word2vec.

The example in Fig. 3 shows the component used to generate the character-based word embeddings. This part of network is trained together with the other parts by the backpropagation algorithm using the common error function. It shows an unrolled (*i.e.*, multiple time step) bidirectional LSTM performing the described operations on the word "Proc" (often short for "Proceedings"). The output of the bidirectional LSTM (lableled CE_{Proc}) is concatenated with the word embedding obtained from the projection matrix $\mathbf{W}$ (labeled as WE_{Proc}).

If required, multiple word embeddings can be trained by using different corpora. Using multiple word embeddings trained on different sources improves the ability of the system by accounting for co-occurrences of words among different domains (*i.e.*, multiple senses). We refer to this as word vector augmentation, explored in more detail in Section 5.1.2.

5 Experiments

We first describe the experimental setup common to all our main experiments, then describe the ParsCit baseline that we compare against. We then describe the experimental results for sensitivity to word embeddings, various components of LSTM model selection, selection and robustness across domains and languages.

Setup[2]**.** We use word embeddings pre-trained by Google using the word2vec toolkit that pre-trained over a corpus of 100B words from Google News. This model produces 300-dimensional vectors for 3M unique words and phrases. We also train in-domain word embeddings using the word2vec toolkit on 4.3M reference strings extracted from the proceedings of the Association of Computing Machinery (ACM)'s Digital Library, provided by the ACM as part of a research collaboration. Numbers are changed to a canonical, numeric value by replacing all the numeric characters with zeros when applying the trained model to unseen, test references. This helps in dealing with the data sparsity of the numeric space. During testing, a similar approach is applied to capitalized words; they are checked against any available lowercase form as needed. However, to maintain the casing information, a separate capitalization feature is also fed to network. The trained embedding vectors for 440K unique words have 200 dimensions each. The training for this in-domain word embedding is done using the CBOW configuration. To combat sparsity and noise, we use a frequency filter as in standard practice, learning embeddings for words which occur at least 5 times in the corpus. For each word, we set the number of negative-sampled words to 25. The vector representation achieved shows high similarity and closer vector representation for noisy variants of the same words. For instance, the words closest to "proceedings" are "proc.", "Proc", "Proceeding", based on cosine similarity. Words such as "proceedings" and "conference" are also highly similar, as compared to similarly spelled but semantically distinct word pairs such as to same for "proceedings" and "preceding".

Our model configurations are identical to those reported implementation of LSTM for named-entity recognition (Lample et al, 2016). The implementation uses the Python-based deep learning library, Theano (Bergstra et al, 2010). All of the models are trained and tested on an NVIDIA GeForce GTX Titan X. The setup requires a few minutes of training, but inference is as fast as the CRF-only models taking only few seconds. The LSTM is trained on per-sample stochastic gradient descent, with dropout regularization, using minimum cross entropy criteria (Bengio, 2009) for parameter learning. For the CRF setup, we use complete sequence (as opposed to token) error, incorporating transition errors. The learning rate used is 0.001. Parameters are initialized randomly with a uniform distribution between –1 and 1. All out-of-vocabulary (OOV) words – *i.e.*, words which are not present in the pre-trained word embeddings – are initialized to a single, randomly chosen uniform distribution between –1 and 1. In our experiments, we found that 8–10% of words are OOV, and thus are assigned the same random word embedding. Character embeddings are similarly initialized: each unique character is initialized with a unique

[2] Code and data available at `https://github.com/WING-NUS/Neural-ParsCit`.

random vector, where the dimension of the character embedding vector is set to 25. The characters are padded with dummy characters to make the sequences of constant length which is maximum of all words in the particular reference string. For citation sequences, no padding is done and training is done sample by sample to accommodate for variable-sized sequence.

We set the dimension of BLSTM layer (which produces the character based word representation) to 25 and the dimension of the pre-final BLSTM layer (which runs over the concatenated word embeddings) to 100. To be clear, the dimension of a layer refers to the size of the output of the layer ($\mathbf{h_t}$). The values of the different hyper-parameters discussed above are selected based on general guidelines and prior experience; we did not perform an exhaustive grid search for optimal values.

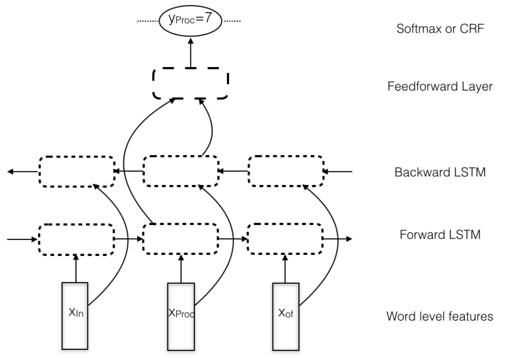

Fig. 4 An unrolled representation of our final model comprising of BLSTM using word embedding features followed by a feedforward layer and CRF.

Fig. 4 shows an unrolled representation of our complete CRF–BLSTM network which runs over the concatenated word vector (concatenated word embedding and character-based word embedding). The LSTM output are fed to the feedforward layer, then through a softmax to make the multiclass decision. As we will show in our experimentation, the LSTM shows inferior performance by itself, but when used as input to a CRF (layer), improves the CRF performance, resulting in state-of-the-art performance. Fig. 4 shows the workflow for a part of the reference string “In Proc of”. In deciding the class for “Proc”, first the output of Fig. 3 is fed to the input of both forward and backward LSTMs. The forward LSTM uses the history accumulated from start of the string until “In” to produce the output vector. Similarly, the backward LSTM does the same using the history accumulated from the end of the string until “of” to produce the output vector. These two output vectors are concatenated and passed through a feedforward layer utilizing a $tanh$ activation function. Finally, the output of the feedforward layer is passed to a softmax or CRF layer, yielding the final classification. For the standard LSTM model, the output is a probability distribution, so word classes are directly predicted as output. For the variants where we stack the CRF model as the final layer, the classification is delayed; the best assignment is computed for the whole sequence as part of the CRF operation instead of individual assignments.

All proposed neural models perform I/O operations in the same fashion as the ParsCit by using the same APIs to read from tokenizer and feature extractor and to write to same output file. This makes integration or even complete migration to neural models easy. The only significant barrier to adoption constraint over the baseline ParsCit is its in-memory requirements for loading the word embeddings.

We detail our experimentation, beginning with simpler models and datasets, subsequently increasing the complexity, so as to examine the effects of scaling.

5.1 Model and Feature Selection

We examine the various components of the models, and do additive studies on the performance of the model as we introduce those components. All results presented are average performance, using 10-fold cross validation, splitting 80% for train, and 10% for validation and test, respectively. The results are reported on the test set and validation set is used for model selection, as is standard. We use the standard Cora dataset, which has 500 human annotated reference strings sampled from various sub-domains of computer science articles available in English.

5.1.1 LSTM trained with Word Embeddings

Table 1 shows the result of the LSTM configuration trained with embeddings trained on the ACM proceedings references using word2vec.

This baseline model serves as a reference point for our experimentation. In particular, note the 0.0 F_1 for classes *Editor* and *Note*, highlighting the skewed nature of the dataset. Under this model, these classes have insufficient examples for model training.

5.1.2 Effect of Augmented Word Embeddings

Table 2 shows the result of the LSTM configuration built incrementally from the model described in Section 5.1.1; *i.e.*, in addition to word embeddings trained on ACM proceedings reference, pre-trained embeddings trained on Google News (available online[3]) augmented the trained word embedding.

[3] `https://code.google.com/archive/p/word2vec/`

Class	Precision	Recall	F_1
Author	97.24	87.85	92.31
Booktitle	86.99	72.31	78.97
Date	92.65	80.19	85.97
Editor	0.0	0.0	0.0
Institution	34.12	62.05	44.03
Journal	31.8	74.36	44.55
Location	18.18	67.66	28.65
Note	0.0	0.0	0.0
Pages	83.63	68.17	75.11
Publisher	23.89	92.67	37.98
Tech	8.12	40.0	13.51
Title	94.32	82.08	87.78
Volume	85.95	82.41	84.14
Macro Average	—	—	51.77
Micro Average	—	—	80.39

Table 1 LSTM prediction performance, trained using WE trained on ACM proceedings over the Cora reference string dataset.

Class	Precision	Recall	F_1
Author	98.76	90.9	94.67
Booktitle	88.57	82.69	85.53
Date	90.97	85.03	87.9
Editor	2.65	40.0	4.97
Institution	77.17	62.42	69.01
Journal	76.62	74.53	75.56
Location	36.3	61.63	45.69
Note	0.0	0.0	0.0
Pages	81.03	76.73	78.82
Publisher	27.06	77.07	40.06
Tech	10.58	52.14	17.59
Title	95.21	88.58	91.77
Volume	84.0	82.47	83.23
Macro Average	—	—	59.6
Micro Average	—	—	85.11

Table 3 LSTM performance over Cora, trained using both Augmented and Character-based WEs. Compare against figures in Tables 2 and 1.

Class	Precision	Recall	F_1
Author	99.05	91.03	94.87
Booktitle	88.9	81.99	85.31
Date	92.18	85.37	88.64
Editor	1.42	30.0	2.72
Institution	74.75	63.53	68.68
Journal	68.59	72.63	70.55
Location	43.44	63.06	51.45
Note	0.0	0.0	0.0
Pages	81.38	81.48	81.43
Publisher	30.94	83.1	45.09
Tech	13.83	57.5	22.29
Title	95.34	87.97	91.51
Volume	84.32	82.73	83.52
Macro Average	—	—	60.47
Micro Average	—	—	85.11

Table 2 LSTM performance over Cora, trained using WE trained on ACM proceedings augmented with Google News WE. Compare against figures in Table 1.

Class	Precision	Recall	F_1
Author	98.57	91.06	94.66
Booktitle	89.08	87.15	88.1
Date	93.43	92.09	92.76
Editor	4.3	30.0	7.52
Institution	81.91	64.06	71.89
Journal	85.64	81.26	83.39
Location	65.75	73.02	69.19
Note	0.0	0.0	0.0
Pages	91.58	90.57	91.07
Publisher	56.76	78.84	66.0
Tech	29.81	79.21	43.32
Title	96.64	92.07	94.3
Volume	92.32	85.56	88.82
Macro Average	—	—	68.54
Micro Average	—	—	88.43

Table 4 Bidirectional LSTM performance over Cora, trained using Augmented and Character-based WE. *Compare against* Table 3.

A comparison between Tables 1 and 2 shows that using embeddings originating from multiple domains increases performance on almost all classes. In particular, the model gives comparatively better results on classes such as *Institution*, *Journal* and *Location*, where the entities are also used in day-to-day English. This makes sense as the word senses for such words are well captured as the embedding space is generated from Google News.

5.1.3 Effect of Character based Word Embeddings

Table 3 shows the result when the previous LSTM configuration from Section 5.1.2 additionally appends character-based word embeddings.

Though the results of adding this additional feature to the model does not yield additional gain on F_1, an analysis of results yields some insight on the source of positive contribution. As a case in point, the *Journal* class posts a large 5 F_1 point gain. Inspection of the output indicates that the model that appends the character embedding makes significantly less mistakes for patterns involving complex character sequences like “(NIPS*00)”,“IEEE” and “IEEE/ACM”. But as a consequence, the model makes mistakes in other classes, mostly due to the additional modeling effort for such prefixes and suffixes. For example, “MIT/LCS/TR-000” is classified as *Institution*, dis-preferred to the correct *Tech* class. This instance was classified correctly by the word based LSTM not using character based word embedding; the character enhanced LSTM likely overfit, seeing the prefix “MIT”.

5.1.4 Effect of Bidirectional LSTM

Table 4 shows the result when we add bidirectionality into the LSTM framework onto the previous experimental setup. Bidirectionality is applied to both word and character LSTM sequences.

Class	Precision	Recall	F_1
Author	99.72	98.48	99.1
Booktitle	93.85	95.79	94.81
Date	99.37	98.5	98.93
Editor	89.28	91.76	90.51
Institution	84.8	89.12	86.91
Journal	94.06	91.3	92.66
Location	91.4	90.26	90.82
Note	48.17	68.42	56.54
Pages	98.34	97.74	98.04
Publisher	91.64	94.96	93.27
Tech	83.89	80.97	82.41
Title	98.15	96.76	97.45
Volume	94.31	94.41	94.36
Macro Average	—	—	90.45
Micro Average	—	—	95.68

Table 5 CRF–BLSTM performance over Cora, trained using Augmented WE and Character based WE. Compare against Table 4.

The results in Table 4 affirms prior work that using future information in the LSTM framework indeed improves performance. Contrasting the model's predictions with the previous ones, this effect is particularly pronounced – including parenthetical digit sequences such as "(0000)" and "0(00)" in *Booktitle* and *Title*, when they should be attributed to *Volume*; and confusing "(0000)" between *Date* and *Volume* – among others. These errors can be classified as cases where the decision boundaries occurs after long, consecutive repeating labels occur in the forward direction – such as *Title* and *Booktitle* – which can be difficult to model in the forward direction. These are easily tracked in the reverse direction, as the long-range forward dependencies become short dependencies in reverse.

5.1.5 Effect of the CRF

Table 5 shows the result of stacking a final CRF layer (CRF–BLSTM), over the previous scenario.

The resultant CRF–BLSTM trained using augmented and character-based embeddings categorically shows that keeping a Markovian blanket over non-Markovian feature interaction helps the classification with the small amount of data and skew classes. The CRF layer enables the model to identify *Notes*, and differentiate *Editors* from *Authors*, dramatically boosting the overall F_1.

In our experiments we decrease the number of parameters in the basic LSTM (having 110K parameters, 95.68 micro F_1) to No Peephole (NP–LSTM having 85K parameters, 96.06 micro F_1) and Coupled Input Forget Gate (CIFG–LSTM having 80K parameters, 96.18 micro F_1) variants (Greff et al, 2015). We use CIFG–LSTM variant in further experiments. The smaller number of parameters lessens the memory footprint of the model and reduces the sparsity and data requirements when training the model.

5.2 Comparison across different datasets

Having established a competitive deep learning architecture — CRF–BLSTM (also CIFG–LSTM) using augmented and character-based WE — for reference string parsing, we examine its performance against the baseline ParsCit model. For ease of reference, we term this final system as "Neural ParsCit", a neuralized reference string parser, that accomplishes same scope of work as the original ParsCit project. To better assess its real-world performance, it is important to validate performance over a wider variety of data, not limited to just the Cora dataset. All results presented are average 10-fold cross validation performance for both models trained on same splits. For each experiment the dataset is randomly split in 10 folds where 90% of data is used as train, and 10% for test per fold.

5.2.1 Comparison on Cora

Neural ParsCit outperforms ParsCit with statistical significance, reducing error by over 20% (11.17% reduced to 8.63%) in macro-averaged and 10% (4.34% reduced to 3.94%) in micro-averaged performance. The gain comes from the major confusion classes of *Booktitle* and *Title*. Table 7 shows the difference of all errors types for ParsCit and Neural ParsCit where only one model predicts the correct class. Each entry in the table shows relative confusion for the particular model. Most of the error from false positives on prominent, long range entities *i.e. Booktitle* and *Title* has been removed by Neural ParsCit. Interestingly, the confusion between *Booktitle* and *Title* is slightly increased for the neural model, which is indicative of strong semantic closeness of these classes.

5.2.2 Cross Domain Comparison

In addition to the collection of 500 reference strings from Cora, 300 reference strings from FLUX-CiM, 75 reference strings from ICONIP – all largely from the hard sciences in English, it includes 178 reference strings from the English humanities dataset.

Results from the cross domain comparison show conflicting performance against the ParsCit baseline; a 20% gain in error reduction on macro-averaged, but a 1% loss in micro-averaged performance. Further investigating the errors introduced by the neural model, performance degradation was primarily due to the increased confusion between *Booktitle* and *Title* (Table 9). We found that most error instances had patterns or repeating structures. One such example is the reference string "*Barklund, J., Costantini, S., Dell'Acqua, P. and Lanzarone, G. A., Integrating Reflection into SLD-Resolution, in: A. Momigliano and M. Ornaghi (eds.), Proc. Post-Conf. Ws. on Proof-Theoretical Extensions of Logic Programming, 1994*". Due to fact that *Editor* are semantically very close to *Authors*, the occurrence of *Booktitle* after such similar signals may have

Model	ParsCit			Neural ParsCit		
Class	Precision	Recall	F_1	Precision	Recall	F_1
Author	99.45	98.22	98.83	99.48	98.37	98.92
Booktitle	96.14	92.56	94.32	94.09	93.51	93.8
Date	97.64	99.66	98.64	99.22	98.54	98.88
Editor	75.68	85.56	80.32	84.51	96.37	90.05
Institution	84.47	90.79	87.52	90.69	90.76	90.73
Journal	91.6	92.96	92.28	92.68	92.24	92.46
Location	86.96	90.89	88.88	90.78	91.85	91.31
Note	49.62	72.49	58.91	58.49	70.38	63.89
Pages	97.85	97.88	97.86	98.36	97.9	98.13
Publisher	76.17	88.38	81.82	92.05	92.53	92.29
Tech	73.33	94.6	82.61	84.97	83.7	84.33
Title	98.97	96.43	97.68	98.02	98.09	98.06
Volume	95.68	94.66	95.17	95.73	94.19	94.95
Macro Average	—	—	88.83	—	—	91.37
Micro Average	—	—	95.66	—	—	96.06*

Table 6 ParsCit versus Neural ParsCit on Cora, where '*' indicate significant improvements at the $p < 0.05$ on one-tail paired Student's t-test.

	author	booktitle	date	editor	institution	journal	location	note	pages	publisher	tech	title	volume
author	—	0	0	0	0	0	-1	0	0	0	0	10	0
booktitle	0	—	-2	0	0	-6	0	0	-2	-3	0	-12	-3
date	0	2	—	1	0	1	6	0	1	0	0	0	1
editor	2	21	0	—	0	0	0	0	0	0	0	0	0
institution	2	11	0	0	—	0	-1	0	0	2	-2	14	0
journal	0	0	0	0	-1	—	-1	0	0	0	-2	9	1
location	0	7	-1	0	-3	0	—	0	0	0	-1	0	0
note	0	8	0	2	0	0	-2	—	-1	1	-2	3	-1
pages	0	0	-1	0	0	1	-1	0	—	1	0	2	-3
publisher	0	8	0	0	1	0	7	1	0	—	0	13	1
tech	0	0	0	0	3	0	0	0	1	2	—	12	0
title	-10	-9	0	0	-2	-3	-1	-3	0	2	-9	—	0
volume	0	6	-3	0	0	0	0	0	-1	0	0	3	—

Table 7 Relative confusion for ParsCit versus Neural ParsCit on Cora. Rows represent ground truth classes; columns, predicted classes. The values represents the difference of confusion ($Predict_{ParsCit} - Predict_{NeuralParsCit}$) aggregated over samples where exactly one of the model predicts the class correctly. Negative entries indicates wins for Neural ParsCit.

Model	ParsCit			Neural ParsCit		
Class	Precision	Recall	F_1	Precision	Recall	F_1
Author	98.83	98.43	98.63	98.62	98.48	98.55
Booktitle	95.34	94.4	94.87	93.15	95.78	94.44
Date	98.47	98.41	98.44	99.06	97.85	98.45
Editor	81.43	89.92	85.47	84.08	88.29	86.13
Institution	81.43	95.81	88.04	90.61	87.66	89.11
Journal	91.48	93.33	92.4	90.22	90.98	90.6
Location	93.24	91.88	92.55	93.63	93.47	93.55
Note	44.8	84.36	58.52	69.84	84.94	76.65
Pages	98.93	98.4	98.66	98.93	99.15	99.04
Publisher	81.24	92.15	86.35	91.87	93.0	92.43
Tech	65.12	95.3	77.37	84.45	87.66	86.03
Title	98.36	95.76	97.05	97.89	96.24	97.06
Volume	95.75	94.21	94.97	94.75	94.36	94.55
Macro Average	—	—	89.49	—	—	92.05
Micro Average	—	—	95.85	—	—	95.79

Table 8 ParsCit and Neural ParsCit performance on the Cross Domain dataset.

tipped the model to tag *Title*, based on training showing *Title* is often tagged after tokens semantically close to person names. As *Booktitle* also has similar semantics to *Title* – "*Proof-Theoretical Extensions of Logic Programming*", compared with "*In Proceedings of the SIGPLAN '90 Conference on Program Language Design and Implementation*" – further exacerbates the confusion.

Such pattern-based errors can be mitigated in production systems by simply adding error-specific postprocessing to the output; or by selectively adding more training samples with such variations. We show in the following section, that in the confines of our dataset, since most of such instances come from the English humanities dataset (approximately 15 instances), when we add multilingual humanities reference strings to the dataset, the number of such instances increases and as a result, the model does correct such mistakes on its own.

5.2.3 Comparison on Multilingual Data

We also compared both systems on multilingual data. In addition to the English reference strings from the cross-domain dataset, we added 77 instances of Italian and 67 instances of French and German reference strings.

In our neural model, strings in foreign languages are translated to English using Google Translate. The training and testing is done on this canonical dataset containing only-English reference strings. After the classes are predicted on the translated string, we map back the predicted class to the original foreign language. This is done by mapping the substrings using the presence of delimiters and assigning the class values to the substring. This method assumes that the translation process maintains the order among the classes within the reference string.

The results show that both models make similar mistakes on foreign words, implying that neither approach works optimally for multilingual datasets. From Table 11, one important observation is that confusion between *Booktitle* and *Title* is reduced by the current model. This implies as Neural ParsCit does well on the mixed dataset, where no postprocessing was applied (Table 9), which is also part of this test setting, covering the margin from –67 to 39. Further analysis shows that the poor results of ParsCit was due to its fundamental limitation of using character $n-$grams as features. Character $n-$grams are highly dependent on language. In absence of semantic knowledge, the uniform treatment of prefixes caused the baseline CRF-based model to fail. A representative error example (made by the ParsCit baseline and not by Neural ParsCit) – "*A. Camerotto, 'Storie cretesi, ovvero altre storie: tra Idomeneus e i suoi 'parenti', in Tradizioni locali e generi letterari nella Grecia arcaica: epos minore, lirica ed elegia, storiografia, Atti del Convegno di Studi (Venezia, 21-22 settembre 2006), a c. di E. Cingano, Roma-Pisa 2008 [c.d.s.]*". Here "*in Tradizioni locali e generi letterari nella Grecia arcaica: epos minore, lirica ed elegia, storiografia, Atti del Convegno di Studi*" *Booktitle* is continued to be reported as *Title* since no obvious character $n-$gram signals are displayed. Unfortunately, such errors cannot be processed during post processing as identifying boundary in such cases is difficult. In contrast, the errors made by Neural ParsCit seem more semantically plausible from our point of view — for example, predicting "High" and "School" as *Institution* instead of *Title* in cases where those tokens should have been correctly tagged as *Title*.

6 Discussion

We now discuss particular issues that connect our claims against the empirical results.

Issue 1: How well does the neural model handle long range dependencies? We find that longer dependencies that occur towards the end of citation strings are the single largest source of error – commonly instances of *Title* and *Booktitle*. For the results reported in Tables 6 and 7, we analyze the location of occurrence of all *Booktitle* incorrectly identified by both the models, on average, is 16 words from the beginning, with a standard deviation of 8. In comparison, for correct instances, the average location is just 10 with an s.d. of 5 — much less. The spans of the differential performance validates our claim that the Neural ParsCit works better on such longer-range dependencies. Separately, our neural model learns to identify long-range dependencies for classes which are semantically distinct from each other *Publisher*, *Author* and *Title*. This niche ability gives it the performance edge over the CRF only baseline which not use any semantic representation of words.

Issue 2: How do word embeddings perform as features? The word embeddings trained by using word2vec do capture useful semantics of the words within reference strings, purely based on their co-occurrence. The embedding algorithms do manage to group words belonging to same class together, separating them in the high-dimensional vector space. For example, words that indicate (conference) publication venues, such as "COLING", "MobiSys" and "INTERSPEECH", which occur in similar context windows of the *Journal*, *Booktitle* and *Publisher* fields are indeed brought closer in their vector representation by the unsupervised algorithm. Our inspection of the unsupervised, learned embeddings suggests that they are more appropriate than the former, human-engineered features in the baseline ParsCit system, and are more robust. This claim is further validated backed by the results on CRF–BLSTM model (Table 6), which uses word embeddings and gives an average micro F_1 of 95.9% as compared to ParsCit, which uses handcrafted features and gives a micro F_1 of 95.7%. For most of the

	author	booktitle	date	editor	institution	journal	location	note	pages	publisher	tech	title	volume
author	—	0	0	-18	0	0	-1	0	0	-1	0	10	0
booktitle	0	—	1	-4	-6	-32	1	-3	0	-3	0	-67	-5
date	0	-1	—	0	0	0	10	-1	2	0	0	-1	0
editor	1	10	0	—	0	0	2	0	0	0	0	-4	0
institution	0	4	0	0	—	0	6	0	0	3	-1	20	0
journal	8	-8	0	0	0	—	-1	-1	0	0	-2	-8	0
location	0	15	-2	0	-8	0	—	0	0	-4	0	0	0
note	0	13	6	3	1	0	3	—	0	0	-1	20	0
pages	0	0	1	0	0	0	0	0	—	1	0	2	-4
publisher	8	13	0	1	-3	1	9	1	0	—	0	7	1
tech	0	0	-2	0	2	8	1	0	1	1	—	26	0
title	-11	1	0	-7	-6	-10	-8	-2	1	2	0	—	0
volume	0	3	-2	0	0	-2	0	-2	1	0	-2	-1	—

Table 9 Relative confusion matrix for ParsCit versus Neural ParsCit on the Cross Domain dataset. Rows represent ground truth classes; columns, predicted classes. The values represents the difference of confusion $(Predict_{ParsCit} - Predict_{NeuralParsCit})$ aggregated over samples where exactly one of the model predicts the class correctly. Negative entries indicate a win for the neural version.

Model	ParsCit			Neural ParsCit		
Class	Precision	Recall	F_1	Precision	Recall	F_1
Author	98.78	98.94	98.86	99.32	98.28	98.8
Booktitle	94.19	93.13	93.66	94.41	95.89	95.15
Date	97.94	97.92	97.93	98.87	98.23	98.55
Editor	90.67	92.86	91.75	88.01	92.02	89.97
Institution	78.29	95.77	86.15	91.44	90.97	91.2
Journal	91.92	91.45	91.68	90.88	91.86	91.37
Location	93.01	92.23	92.62	94.95	92.71	93.81
Note	58.69	87.54	70.27	65.62	93.61	77.15
Pages	98.59	98.39	98.49	99.27	98.56	98.91
Publisher	78.49	92.32	84.85	93.87	92.82	93.35
Tech	68.19	92.77	78.61	83.16	86.16	84.63
Title	97.85	95.5	96.66	98.34	97.16	97.74
Volume	95.24	94.26	94.75	95.49	95.37	95.43
Macro Average	—	—	90.48	—	—	92.77
Micro Average	—	—	95.48	—	—	96.47**

Table 10 ParsCit versus Neural ParsCit on multilingual data, where '**' indicate significant improvements at the $p < 0.01$ on a one-tail paired Student's t-test.

	author	booktitle	date	editor	institution	journal	location	note	pages	publisher	tech	title	volume
author	—	0	1	3	0	0	0	0	0	0	0	26	0
booktitle	0	—	0	-11	-1	-1	-9	0	-1	-4	0	39	-4
date	0	0	—	0	0	1	11	2	0	0	0	-1	1
editor	-29	18	0	—	0	0	-1	0	0	0	0	0	0
institution	1	24	0	0	—	0	5	0	0	3	-1	15	0
journal	4	3	0	0	0	—	0	1	0	0	-3	-15	0
location	0	15	1	0	-3	2	—	0	1	-1	0	1	0
note	0	7	11	4	2	0	0	—	-2	0	1	6	-1
pages	0	-2	0	0	0	1	-1	0	—	1	0	2	6
publisher	3	15	1	2	0	1	5	1	0	—	0	22	1
tech	0	4	0	0	2	6	1	0	2	1	—	19	0
title	-9	72	1	-3	-9	-4	3	-5	1	2	-2	—	0
volume	0	8	-3	0	0	-2	0	1	0	0	-2	0	—

Table 11 Relative confusion for ParsCit versus Neural ParsCit on multilingual data. Rows represent ground truth classes; columns, predicted classes. The values represents the difference of confusion $(Predict_{ParsCit} - Predict_{NeuralParsCit})$ aggregated over samples where exactly one of the model predicts the class correctly.

classes, the CRF–BLSTM based models outperform the baseline ParsCit, with minority class exceptions in minority classes such as *Date*. However, as our comparison extends to more complex dataset variations, the gap between our unsupervised neural model and the baseline widens. Eventually, on the complete dataset (Table 10) CRF–BLSTM outperforms all ParsCit variants by a large margin (4.52% token error rate reduced to 3.53%, a relative 21% reduction in token error rate). The final results show a 1% absolute micro F_1 gain, quite significant given the high performance of the baseline. The relative gains (0.34 on 94.01 F_1 score baseline) are much better than that achieved by high order semi-CRFs Cuong et al (2015) which are difficult to train.

Issue 3: Why is the CRF layer crucial? The deployment of a BLSTM without the use of a final stacked CRF layer yields poor performance, as evidenced by the difference in performance between Tables 4 and 5. On closer analysis, we find that in the case where CRF is absent, the output as seemingly randomly-assigned for some words. This means that even though LSTMs are powerful enough to identify long-range dependencies, they can fail to leverage even close contextual history information – such as in the case where the labels of the past and future words might be the same as the current word. We believe this may occur due to the sparsity of the training data. To test this hypothesis, we used unlabeled reference strings from the larger ACM dataset – previously used to train word embeddings – extracting correct (but often noisy) labels from its associated BibTeX, achieved by acquiring the paper metadata by crawling associated Digital Object Identifiers (DOIs[4]). We re-train both models on 14K samples. The BLSTM model F_1 score increased significantly, resulting in a final 95.46% F_1 micro (cf CRF-BLSTM's 95.78%). This validates our intuition that BLSTM can learn these dependencies with sufficient training (noisy) samples. This also opens avenue for building enterprise solutions by crawling lot of noisy annotations from multiple sources. Empirically speaking, the CRF–BLSTM architecture proves to be the best, most robust configuration in our experiments. A closer look at the CRF objective function reveals that the model requires less data to learn and disfavors frequent changes in labels. This implies that CRF makes the model more resilient to noise from the OOV's word embedding based features. By comparing the outputs from Sections 5.1.5 and 5.1.4, we see that adding the CRF layer restricts the switching of predicted class labels by approximately 12% (from 256 down to 225).

Issue 4: Could word embeddings tackle the multilingual task directly? The current model adopts an interlingua approach for multilingual reference string parsing; *i.e.*, it translates any multilingual string to English and subsequently does the parsing of the English string. While this can be effective, it requires the support of an (online API) translation system. With the availability of multilingual word embeddings and better access to raw multilingual (unlabeled) reference strings data, unifying the embedding space for multiple languages can still be a challenging task. This direction is promising to address the multilingual nature of knowledge in future work.

[4] `https://www.doi.org/`

7 Conclusion

We apply deep learning towards the task of reference string parsing. Importantly, the deep learning architecture allows the unsupervised induction of features tuned towards the reference string parsing task. Our work demonstrates the superiority of abstract numeric representations of the words learned from unlabeled data as compared to handcrafted and dictionary features. Our work shows how the generic long short-term memory (LSTM) model can be imbued with more power by using both general domain pre-trained and in-domain learned word embeddings. For multilingual datasets, we explored a unified approach, where we translated the reference string to English, parsed them using the proposed technique and propagated the parses and labels back into the original language string.

Our work highlights the importance of using a conditional random field (CRF) layer to increase the robustness of the model, attaining par performance with the handcrafted, state-of-the-art systems. In all our proposed models, we get consistent significant macro and micro F_1 gains over the existing ParsCit version, already acknowledged as a strong baseline on this task.

Acknowledgments

We would like to acknowledge the support of the NExT research grant funds, supported by the National Research Foundation, Prime Minister's Office, Singapore under its IRC @ SG Funding Initiative. We would also like to gratefully acknowledge the support of NVIDIA Corporation with the donation of the GeForce GTX Titan X GPU used for this research.

We also acknowledge Muthu Kumar Chandrasekaran for insightful feedback on data handling and editing, along with Kishaloy Halder and Wenqiang Lei for their help in the ongoing integration with the current ParsCit pipeline.

References

Bengio Y (2009) Learning deep architectures for AI. Foundations and trends® in Machine Learning 2(1):1–127

Bengio Y, Simard P, Frasconi P (1994) Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks 5(2):157–166

Bergstra J, Breuleux O, Bastien F, Lamblin P, Pascanu R, Desjardins G, Turian J, Warde-Farley D, Bengio Y (2010) Theano: A CPU and GPU math compiler in python. In: Proc. 9th Python in Science Conf, pp 1–7

Chen CC, Yang KH, Chen CL, Ho JM (2012) Bibpro: A citation parser based on sequence alignment. IEEE Transactions on Knowledge and Data Engineering 24(2):236–250

Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P (2011) Natural language processing (almost) from scratch. Journal of Machine Learning Research 12(Aug):2493–2537

Councill IG, Giles CL, Kan MY (2008) Parscit: an open-source CRF reference string parsing package. In: LREC, vol 8, pp 661–667

Cuong NV, Chandrasekaran MK, Kan MY, Lee WS (2015) Scholarly document information extraction using extensible features for efficient higher order semi-CRFs. In: Proceedings of the 15th ACM/IEEE-CS Joint Conference on Digital Libraries, ACM, pp 61–64

Gers FA, Schmidhuber J, Cummins F (2000) Learning to forget: Continual prediction with LSTM. Neural computation 12(10):2451–2471

Giles CL, Bollacker KD, Lawrence S (1998) Citeseer: An automatic citation indexing system. In: Proceedings of the third ACM conference on Digital libraries, ACM, pp 89–98

Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J (2015) LSTM: A search space odyssey. arXiv preprint arXiv:150304069

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural computation 9(8):1735–1780

Kern R, Klampfl S (2013) Extraction of references using layout and formatting information from scientific articles. D-Lib Magazine 19(9/10)

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp 1097–1105

Lafferty J, McCallum A, Pereira F (2001) Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of the eighteenth international conference on machine learning, ICML, vol 1, pp 282–289

Lample G, Ballesteros M, Subramanian S, Kawakami K, Dyer C (2016) Neural architectures for named entity recognition. arXiv preprint arXiv:160301360

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444

Mikolov T, Chen K, Corrado G, Dean J (2013a) Efficient estimation of word representations in vector space. arXiv preprint arXiv:13013781

Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013b) Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems, pp 3111–3119

Mohamed AR, Dahl GE, Hinton G (2012) Acoustic modeling using deep belief networks. IEEE Transactions on Audio, Speech, and Language Processing 20(1):14–22

Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. ICML (3) 28:1310–1318

Pennington J, Socher R, Manning CD (2014) Glove: Global vectors for word representation. In: EMNLP, vol 14, pp 1532–43

Rabiner L, Juang B (1986) An introduction to hidden markov models. IEEE ASSP magazine 3(1):4–16

Rifai S, Vincent P, Muller X, Glorot X, Bengio Y (2011) Contractive auto-encoders: Explicit invariance during feature extraction. In: Proceedings of the 28th international conference on machine learning (ICML-11), pp 833–840

Romanello M, Boschetti F, Crane G (2009) Citations in the digital library of classics: extracting canonical references by using conditional random fields. In: Proceedings of the 2009 Workshop on Text and Citation Analysis for Scholarly Digital Libraries, Association for Computational Linguistics, pp 80–87

Rumelhart DE, Hinton GE, Williams RJ (1985) Learning internal representations by error propagation. Tech. rep., DTIC Document

Schmidhuber J (2015) Deep learning in neural networks: An overview. Neural Networks 61:85–117

Tkaczyk D, Szostek P, Fedoryszak M, Dendek PJ, Bolikowski Ł (2015) Cermine: automatic extraction of structured metadata from scientific literature. International Journal on Document Analysis and Recognition (IJDAR) 18(4):317–335