IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 26, 2019, accepted August 31, 2019, date of publication September 9, 2019, date of current version September 25, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2939687*

# Deep Learning for Natural Language Parsing

**SARDAR JAF**[1] **AND CALUM CALDER**[2]
[1]Faculty of Technology, School of Computer Science, University of Sunderland, Sunderland SR1 3SD, U.K.
[2]Department of Computer Science, Durham University, Durham DH1 3LE, U.K.

Corresponding author: Sardar Jaf (sardar.jaf@sunderland.ac.uk)

**ABSTRACT** Natural language processing problems (such as speech recognition, text-based data mining, and text or speech generation) are becoming increasingly important. Before effectively approaching many of these problems, it is necessary to process the syntactic structures of the sentences. Syntactic parsing is the task of constructing a syntactic parse tree over a sentence which describes the structure of the sentence. Parse trees are used as part of many language processing applications. In this paper, we present a multi-lingual dependency parser. Using advanced deep learning techniques, our parser architecture tackles common issues with parsing such as long-distance head attachment, while using 'architecture engineering' to adapt to each target language in order to reduce the feature engineering often required for parsing tasks. We implement a parser based on this architecture to utilize transfer learning techniques to address important issues related with limited-resourced language. We exceed the accuracy of state-of-the-art parsers on languages with limited training resources by a considerable margin. We present promising results for solving core problems in natural language parsing, while also performing at state-of-the-art accuracy on general parsing tasks.

**INDEX TERMS** BiLSTM parsing, deep learning, dependency parsing, natural language processing, parsers, shift-reduce parsing, syntactic parsing, transition-based parsing.

## I. INTRODUCTION

Ntural language parsing problems involve determining the syntactic structure (parse tree) of a sentence, which describes its grammatical structure. The two main types of parsing are dependency parsing and constituency parsing. Dependency parse trees are build over direct relations between words or other tokens in a sentence, whereas constituency parse trees are based on the parse trees of formal grammars. The different sentence structure representation by dependency parsers and constituency parsers are presented in Fig. 2a and 2b..

Determining a parse tree of a sentence forms the basis for many other natural language tasks, particularly semantic analysis tasks [1], such as sentiment analysis, information extraction and question interpretation by proving a data structure (parse tree) which encodes more information about a piece of text than the raw text alone.

One of the main types of parsing for natural language text is dependency parsing. Dependency parsing is based on the dependency relation formalism. In this formalism, tokens are associated with each other directly. Each token, excluding the tree's root token, is dependent on an associated 'head' token. Dependents are associated with their head through 'dependency arcs', which can optionally be labeled to provide additional (grammatical) information about the relation. Associating tokens via a dependency arc is sometimes called 'head attachment'.

In the dependency relation formalism, a distinction is made between 'projective' and 'non-projective' dependency trees. Projective trees are, in simple terms, dependency trees without any intersecting arcs. Fig. 1 demonstrates an example of non-projective dependency parse tree and Fig. 2 and Fig. 3 show a projective dependency parse tree.

The varying amounts of data makes designing a parser that is effective on multiple languages presents a challenge; each language has unique grammatical rules and to be effective on multiple languages a parser should be agnostic to these differences in structure. In contrast to this, it is reasonable to expect that the most accurate parser for a single language would have features designed around that language. The dependence upon data in data-driven parsing, however, does introduce with it the task of generating quality treebanks or data sources. For some languages with little, or no available data this becomes an issue. In this paper we explore the way a deep learning algorithm could learn from large data sources

The associate editor coordinating the review of this manuscript and approving it for publication was Yonghong Peng.

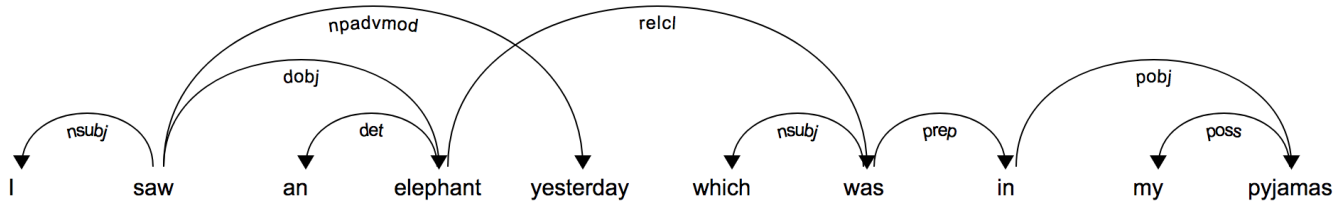
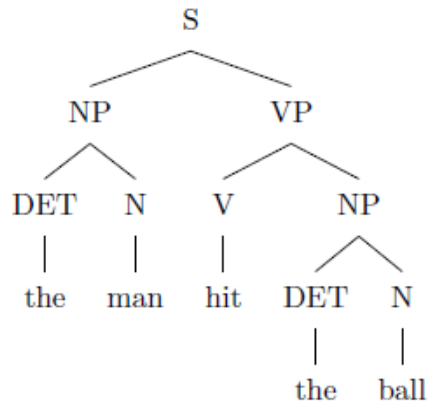
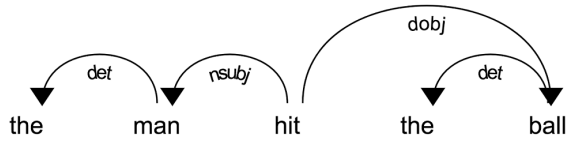



**FIGURE 1.** An example of a non-projective parse tree.




(a) A constituency tree representing the structure of the sentence 'the man hit the ball'




(b) A dependency tree representing the structure of the sentence 'the man hit the ball'.

**FIGURE 2.** Constituency and dependency structures for the sentence 'the man hit the ball'.

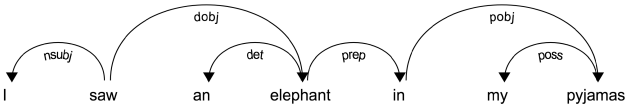
of similar languages to a target language and parse the target language with promising result.

A fundamental problem in parsing is the inherent ambiguity in language [2]–[5]; often sentences have multiple valid interpretations, and additional context is required to differentiate between these interpretations. Given that a sentence may have multiple valid parse trees, as that shown in Fig. 3a and Fig. 3b, a good parser should make some attempt at differentiating between the parse trees to select the most likely valid tree. Much of statistical parsing and data-driven parsing aims to solve this problem.

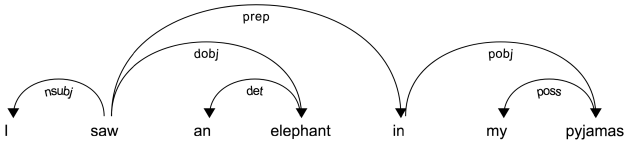
One of the main benefits of using machine learning classifiers, such as Bidirectional Long Short Term Memory (BiLSTM), is it can overcome the issue of ambiguity in natural language sentences.

The main contribution of this work are:

1) The development of a multi-lingual parser by utilizing an advanced form of Recurrent Neural Network (RNN), specifically Bidirectional Long Short Term Memory (BiLSTM).
2) We empirically demonstrate the potential of BiLSTM algorithm for treating long distance dependency relations problem in natural language parsing, which is often an issue with transition-based parsers.
3) Our proposed architecture improves language agnosticism by reducing feature engineering.
4) Our proposed approach significantly reduces the sparsity of data for predicting labels of dependency relations.
5) Our proposed parser improves the accuracy of data-driven parsers for languages with limited resources by utilizing transfer learning technique.




(a) The interpretation of '*I saw an elephant in my pyjamas*' is that a person sees the elephant wearing pyjamas.




(b) This interpretation of '*I saw an elephant in my pyjamas*' is the person seeing the elephant is wearing pyjamas.

**FIGURE 3.** Example of an ambiguous sentence that results in two different parse trees.

The remainder of this paper is organized as follows. Section II presents the related work to parsing natural language text. The proposed approach and a discussion of the parser architecture is presented in Section III. Section IV shows our empirical results of the parser performance and Section V concludes the paper.

## II. RELATED WORK

The development of dependency grammars is typically associated with the work of Lucien Tesnière in *Éléments de syntaxe structurale* [1, p. 268]. Since around 2013 computational linguistics community shifted its focus away from constituency parsing and started utilizing dependency structures in data-driven parsing. The dependency structure representation for an English sentence is demonstrated in Fig. 2, 2b and 3.

An approach to natural language dependency parsing is based on shift-reduce algorithms, also known as transition-based parsing. In transition-based parsing the input is processed incrementally from left-to-right, with the parser making transitions between states according to the cur-

rent configuration of the parser, generally guided by some classifier.

As the core component of many parsers in some data-driven models, machine learning algorithms have seen a lot of use in parsing. With the recent rise of deep learning, there has been a natural trend towards experimenting with deep learning models for parsing natural language text. Deep learning has been particularly effective across natural language tasks when compared to more classic machine learning processes, and this is especially true of parsing.

Graph-based approaches have also proved effective, especially when used alongside machine learning models. From [7] we see that when using BiLSTM feature embedding the performance of graph-based models may in fact be greater than transition-based systems for long-distance parsing, such as for parsing Chinese sentences. In this paper, we elected not to explore graph-based parsing, as the computational complexity of graph-based parsers ($O(n^2)$ in the length of the sentence) results in poorer scaling than most transition-based parsers.

Transition-based algorithms were developed for natural language parsing that operate on probabilistic Context Free Grammars [8]. Similar methods were developed for incrementally parse dependency trees [9]. Transition-based parser algorithms allow parsing of a sentence in linear time with respect to the length of the sentence by making a series of transitions between parser configurations, guided by some 'oracle'.

Gold oracles — oracles that when given a perfectly parsed 'gold' tree provide a transition sequence that produces the tree — are used for producing parse tree from training data. Using the gold oracle, sequences of parser configurations and resulting transitions can be produced from a treebank to be used to train a classifier to approximate this oracle. Such an oracle is presented as [10, Table 1] for Nivre's arc-eager parser. One issue with typical gold oracles is that they depend on all previous transitions to be correct [10], [11]. An extension to this idea, therefore, is the dynamic oracle. Dynamic oracles use a gold tree to produce the optimal transition in a given configuration regardless of the previous states of the parser, and allow for 'error exploration' — allowing the parser to make mistakes during parsing allows the parser to learn to recover from mistakes made during parsing. As transition-based parsing is very dependent on previous transitions [12], this method has been used in many parsers in an attempt to improve accuracy [7], [10], [11] with varying success.

The transition-based algorithms require some way to assign a score to each transition given the configuration of the parser. Nivre and Scholz [13] and Nivre [14] describes an 'oracle' which provides the optimal transition at each configuration in order to perfectly parse a sentence. In practice, this is typically achieved by using machine learning algorithms to approximate this oracle. Following his earlier theoretical work, [13] present a concrete implementation of a dependency parser which utilizes instance-based learning through TiMBL, a framework based on the k-Nearest Neighbor algorithm [15]. In a similar effort, [16] propose a dependency parser which utilizes a Support Vector Machine to infer transitions for their own shift-reduce algorithm. Support Vector Machines (SVMs) have been used quite successfully in parsing tasks, including both chunk parsing methods for constituency parsing [17], and dependency parsing through shift-reduce algorithms [14], [16], [18]. Similary, [3], [19] utilized various tree based classifiers for dependency parsing using transition-based algorithm. A particularly noteworthy parser is MaltParser [20] which has options to utilize either an SVM, or TiMBL as its underlying learning algorithm and was regarded as state-of-the-art for a time.

With the recent rise in interest in deep learning, parsing seems to be an excellent area to apply these techniques, and transition-based parsing is classification problems. Deep learning has excelled at these tasks, and as such provides a natural fit for this class of problem.

In deep learning, recurrent neural networks operate on sequential data, and as such have been used frequently in natural language tasks [21, p. 163]. Long Short Term Memory (LSTM) in particular has been used with some success, including in parsing tasks [22], [23]. These attempts have generally looked to apply the sequential aspect of the LSTM to the sequence of transitions used in a transition-based parser, and are based around a novel kind of LSTM, the Stack-LSTM [22]. An alternative approach utilized in recent years uses BiLSTMs; the sequence of tokens in a sentence is taken as the input to a sequence-to-sequence BiLSTM model which generates a dense feature embedding to be used as the input to another model to parse the sentence. [24], for example, use the feature embedding as a pre-processing step in their graph-based parser, whereas [7] and [25] utilize a BiLSTM feature embedding as an embedding layer to transition-based parsers, achieving good results with only a simple multi-layer perceptron (MLP) to guide the parser. A parser architecture such as this means that the feature engineering required is effectively eliminated, albeit with some trade-off in the form of computational complexity, due to the complexity of a multi-layer BiLSTM.

For this paper, particularly relevant is the use of a deep learning classifier as the oracle to a transition-based parser. One of the first major implementations of a dependency parser backed by a neural network as its learning algorithm is the Stanford parser [12]. Chen and Manning in [12] were able to significantly reduce the amount of feature engineering required by utilizing few dense features, rather than 'millions of sparse indicator features' typically required by other solutions in order to increase parsing speed by eliminating the need to generate such features, increase generality, and to reduce the complexity of the parser. Much of the research in deep learning for parsing has had similar aims – [7] present a parser utilizing a BiLSTM layer, taking only the sequence of words and associated part-of-speech tags (POS) in a sentence as input, and allowing the BiLSTM to build the feature embedding using a sequence-to-sequence architecture before using a multi-layer perceptron to classify transitions.

**TABLE 1.** The transitions defined by Nivre's arc-eager parser.

| Action | State transition | Precondition |
|---|---|---|
| $Left\text{-}Arc_l$ | $(\sigma\|i, j\|\beta, A) \rightarrow (\sigma, j\|\beta, A \cup \{(j, l, i)\})$ | $\neg[i = 0] \cap \neg[\exists k \exists l' s.th. (k, l', i) \in A]$ |
| $Right\text{-}Arc_l$ | $(\sigma\|i, j\|\beta, A) \rightarrow (\sigma\|i\|j, \beta, A \cup \{(i, l, j)\})$ | $\neg[\exists k \exists l' s.th. (k, l', j) \in A]$ |
| $Shift$ | $(\sigma, i\|\beta, A) \rightarrow (\sigma\|i, \beta, A)$ | |
| $Reduce$ | $(\sigma\|i, \beta, A) \rightarrow (\sigma, \beta, A)$ | $\exists k \exists l' s.th. (k, l', i) \in A$ |

**TABLE 2.** The transitions defined by Nivre's arc-standard parser.

| Action | State transition | Precondition |
|---|---|---|
| $Left\text{-}Arc_l$ | $(\sigma\|i, j\|\beta, A) \rightarrow (\sigma, j\|\beta, A \cup \{(j, l, i)\})$ | $\neg[i = 0] \cap \neg[\exists k \exists l' s.th. (k, l', i) \in A]$ |
| $Right\text{-}Arc_l$ | $(\sigma\|i, j\|\beta, A) \rightarrow (\sigma, i\|\beta, A \cup \{(i, l, j)\})$ | $\neg[\exists k \exists l' s.th. (k, l', j) \in A]$ |
| $Shift$ | $(\sigma, i\|\beta, A) \rightarrow (\sigma\|i, \beta, A)$ | |

**TABLE 3.** Parsing '$We_1$ $swam_2$ $at_3$ $the_4$ $beach_5$' using Nivre's arc-eager parser.

| Steps | Configuration | Action |
|---|---|---|
| 1 | $([1], [2, 3, 4, 5], A = \{\})$ | Left-$\text{Arc}_{\text{nsubj}}$ |
| 2 | $([], [2, 3, 4, 5], A)$ | Shift |
| 3 | $([2], [3, 4, 5], A)$ | Right-$\text{Arc}_{\text{prep}}$ |
| 4 | $([2, 3], [4, 5], A)$ | Shift |
| 5 | $([2, 3, 4], [5], A)$ | Left-$\text{Arc}_{\text{det}}$ |
| 6 | $([2, 3], [5], A)$ | Right-$\text{Arc}_{\text{pobj}}$ |
| 7 | $([2, 3, 5], [], A)$ | |

**TABLE 4.** Parsing '$We_1$ $swam_2$ $at_3$ $the_4$ $beach_5$' using Nivre's arc-standard parser.

| Steps | Configuration | Action |
|---|---|---|
| 1 | $([1], [2, 3, 4, 5], A = \{\})$ | Left-$\text{Arc}_{\text{nsubj}}$ |
| 2 | $([], [2, 3, 4, 5], A)$ | Shift |
| 3 | $([2], [3, 4, 5], A)$ | Shift |
| 4 | $([2, 3], [4, 5], A)$ | Shift |
| 5 | $([2, 3, 4], [5], A)$ | Left-$\text{Arc}_{\text{det}}$ |
| 6 | $([2, 3], [5], A)$ | Right-$\text{Arc}_{\text{pobj}}$ |
| 7 | $([2], [3], A)$ | Right-$\text{Arc}_{\text{prep}}$ |
| 8 | $([], [2], A)$ | Shift |
| 9 | $([2], [], A)$ | |

## III. SOLUTION

### A. TRANSITION BASED PARSING

In this paper, we elected to use a transition-based parser. We are motivated to use this algorithm because it is efficient, simple to implement given their straightforward design and many state-of-the-art parsers are based on either the arc-standard or arc-eager shift-reduce algorithms. Whilst simple to implement and understand, these algorithms remain effective as parsing algorithms, allowing design efforts to focus primarily on the neural network model. One particularly appealing aspect of shift-reduce parsers is their efficiency — with $O(n)$ run-times in the length of the sentence, few calls to the model need to be made.

We follow Nivre's notation [9], [14] to outline the arc-standard and arc-eager parsing algorithms. These are presented in Table 1 and Table 2.

Nivre's algorithms maintain a parser state, or 'configuration', which is mutated via some simple state transition functions (shift and reduce functions). Configurations are generally based around at least one 'stack' $\sigma$ which maintains an ordered list of the tokens currently being operated on, a 'buffer' $\beta$ which is an ordered list of the tokens yet to be operated on, and a set of labeled dependency arcs $A$. After reaching a terminal state (generally when $|\beta| = 0$), the parser terminates and returns the arcs made between words $A$ as a set of 3-tuples of the form $(head, label, child)$ where $head$ and $child$ are token IDs, and $label$ is a dependency relation associating the two tokens.

By utilizing a shift-reduce parser, we also gain the ability to swap between shift-reduce algorithms and can explore the effects these changes have on the parser, provided that a gold oracle for that parsing algorithm exists in order to generate the dataset. For example, we could substitute the parsing algorithm for Covington's algorithm [26], or Nivre's list-based parser [14] for an expected $O(n)$ (worst-case $O(n^2)$)-time non-projective parser.

In this paper, we use Nivre's arc-eager algorithm. The arc-eager algorithm attempts to produce shorter parse sequences than other shift-reduce parsers by connecting arcs as early as possible [14], and by connecting arcs as early as possible, the parser should have less of an issue with long-distance parsing. With the arc-standard parser, there may be cases where the parser is in a configuration with two tokens $i, j$ at the top of the stack $\sigma$ and bottom of the buffer $\beta$ respectively, with $(i, l, j)$ being an arc in the gold tree, however according to the rules of the shift-reduce parser, $(i, l, j)$ should not be added yet as some descendant of $j$ has not yet been connected to its head. In such a case, the oracle would require knowledge of which descendants of $j$ had already been assigned a head to guarantee a correct parse tree. By disallowing the model to add an arc as soon as it sees two words which should be connected, we eliminate this problem.

For an example of this behavior, we present the parsing steps in Table 3 and Table 4 for parsing the sentence '$We_1$ $swam_2$ $at_3$ $the_4$ $beach_5$'.

Notice in Table 4 that the parsing diverges at step 3, as we cannot yet form a right arc between $swam_2$ and $at_3$, when using the arc-standard parser; the rule Right-Arc $(\sigma|i, j|\beta, A) \rightarrow (\sigma, i|\beta, A)$ at configuration $([2], [3, 4, 5], A)$ would remove the token $at_3$ from the parser configuration and prevent its descendants ($beach_5$) being assigned their proper head. The Right-Arc operation joining '$swam_2$' and

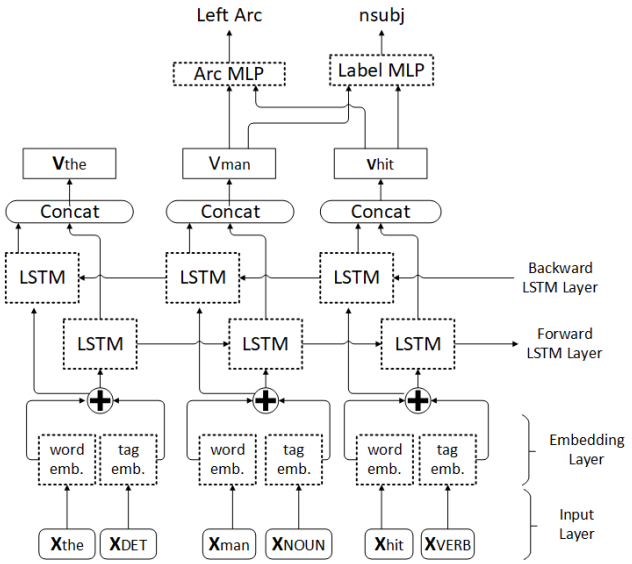



**FIGURE 4.** A simplified view of the parser architecture part way through parsing the sentence *'the man hit the ball'*: The BiLSTM layer builds a feature embedding over the word and POS embeddings $X_t$ of the input tokens $t$, and given a parser state $(\sigma|man, hit|\beta, A)$ the multi-layer perceptrons for classifying the arc and label are given the BiLSTM embedded vectors for 'man' and 'hit' as input.

```
[1.00000e+05 2.80000e+02]
  [4.50000e+01 4.50000e+01]
  [1.90000e+01 1.90000e+01]
            ...
  [9.07000e+02 9.07000e+02]
  [1.00000e+00 1.00000e+00]]
            ...
 [[1.00001e+05 1.00001e+05]
  [1.00000e+05 1.00000e+05]
  [0.00000e+00 0.00000e+00]
            ...
  [0.00000e+00 0.00000e+00]
  [0.00000e+00 0.00000e+00]]
```

(a) An example of embedding representation for word forms

```
[[ 1.   2.   9.   5. ... 16.   9. 17. 14.]
 [ 1.   7.   0.   0.   ...  0.   0.   0.   0.]
 [ 1.   7.   0.   0. ...  0.   0.   0.   0.]
            ...
 [ 1.   7.   0.   0. ...  0.   0.   0.   0.]
 [ 1.   7.   0.   0.   ...  0.   0.   0.   0.]]
```

(b) An example of embedding representation for part-of-speech tags

**FIGURE 5.** The embedding representation for word forms and part-of-speech tags.

'$at_3$' must therefore be delayed until step 7 in the arc-standard parsing.

### B. THE CORE CLASSIFIER

Our parsing model's architecture is similar to that of [25] and [7] — utilizing a BiLSTM to first build a feature embedding layer. The output of the embedding layer is used as input to the BiLSTM layers. And the output of these layers are then used as input to a more typical multi-layer perceptron (MLP), as in [12]. This process is depicted in Fig. 4. The input layer is the words and part-of-speech (POS) tags. They are fed to the embedding BiLSTM layer.The embedding layer generates the vector representation of the input word and part-of-speech tag, as show in Fig 5a and Fig 5b respectively. These vectors are concatenated and used as input to the LSTM layers (forward LSTM layer and backward LSMT layer). The output of the LSTM layers are concatenated to produce the input for the Label MLP and Arce MLP layers, which predict the parse action and the grammatical category of the dependency relations, respectively. The BiLSTM provides many beneficial features; primarily, it encodes positional information, and it significantly reduces the need for feature engineering.

The flow of a sentence through the model during parsing is as follows: each sentence token $t$ has associated with it a word form, a lemma, a universal part-of-speech tag, and a language-specific part-of-speech tag. Some combination of these decided by the model configuration are passed to an embedding layer $e$ to produce $e_{form}(t)$, $e_{lemma}(t)$, $e_{utag}(t)$, and $e_{xtag}(t)$ respectively. Fig. 5a and 5b present a snippet of word form embedding and part-of-speech tag embedding that are obtained from the input words and POS tags and fed in to the LSTM layers, as in Fig 4

The resulting vectors are then concatenated to form a token embedding $x_t = e(t)$ (with e.g. $e(t) = e_{form}(t) \circ e_{utag}(t)$). The word embeddings also include special tokens for the root node of the sentence and for words outside of the vocabulary of the word embeddings. The sequence of token embeddings $x_{t_1}, x_{t_2}, \ldots, x_{t_n}$ is then passed through the BiLSTM feature embedding layer to produce a second sequence $v_{t_1}, v_{t_2}, \ldots, v_{t_n} = BiLSTM(x_{t_1}, x_{t_2}, \ldots, x_{t_n})$.

Once the parser has generated the feature embedding for the sequence of tokens, it begins stepping through the shift-reduce algorithm it has been configured to use, utilizing the model as an oracle. For a parser configuration $C = (\sigma|s_1 \ldots s_n, b_1 \ldots b_m|\beta, A)$, a lookup is done over the BiLSTM embedding of the top $n$ tokens on the stack $s_1 \ldots s_n$ and bottom $m$ tokens of the buffer $b_1 \ldots b_m$ to obtain $v = v_{s_1}, \ldots, v_{s_n}, v_{b_1}, \ldots, v_{b_n}$, where $m$ and $n$ are hyper-parameters of the model. In cases where some $s_i$ or $b_j$ do not exists, as there are fewer than $n$ items on the stack or fewer than $m$ items on the buffer, a special token $v_{none}$ is used to represent $v_{s_i}$ or $v_{b_j}$. With the $v$s calculated, there are then two multi-layer perceptrons, $MLP_{arc}$ and $MLP_{label}$. For the arc-eager parser, with base transitions $\mathcal{T} = \{Left\text{-}Arc, Right\text{-}Arc, Shift, Reduce\}$, $MLP_{arc}$ produces a score for each $T \in \mathcal{T}$ in the form of a tuple as in Equation 1

$$MLP_{arc}(v) = (Score_{Left\text{-}Arc}(v), Score_{Right\text{-}Arc}(v), \times Score_{Shift}(v), Score_{Reduce}(v)) \quad (1)$$

Given a configuration $C$, not all transitions are necessarily allowed due to the prerequisites of each transition, and so we

also define $\mathcal{T}_a$ to denote the set of valid transitions. We have then that the transition made by the parser in configuration $C$ using Equation 2

$$\arg\max_{T \in \mathcal{T}_a} Score_T(v) \tag{2}$$

The primary difference between our parser and more traditional transition-based parsing models is the BiLSTM feature embedding. Our motivations for using the embedding layer are two-fold; positional encoding, and the near-elimination of feature engineering. The latter of these has been a core motivation for using deep learning for parsing; [12] use only 18 input features in their model, and similarly, [27] use 21 input features in their graph-based model. While these approaches are certainly an improvement on the indicator features used by early parsing models, using a BiLSTM feature embedding layer reduces this feature engineering further through 'architecture engineering' [7]. The fact that the BiLSTM passes information in both directions between cells allows the tokens' feature embedding vectors to encode information about the relationship between tokens, allowing some of the features that might be encoded by some feature engineering to be encoded directly in the model. As one of the objectives of this study was to design a multi-lingual parser, the reduction of feature engineering was particularly important when designing the model architecture.

Different languages have different structural rules, and so to build an optimal parser without some feature embedding layer would require feature engineering for each individual language. By having the model learn this feature engineering itself, we eliminate the need to construct optimal features for each language and instead approximate more complex features as part of our model to achieve our objective of developing a parser with transfer learning capability. This is not to say, however, that the requirement for feature engineering was totally eliminated. In our model, we make use of a context window to look-ahead in to the stack and buffer in each parser configuration. This look-ahead allows the model to gain additional contextual information regarding the state of the parsing algorithm, much like the context windows used in more classical approaches [12], [27].

One of the major problems in parsing is the parsing of long sequences of tokens, particularly for greedy parsers such as transition-based parsers — a mistake early on propagates throughout the entire parse, and has a knock-on effect [12]. The use of context windows gives the parsing models the ability to look somewhat in to the future of the parsing sequence helps to eliminate this problem, however the effectiveness of a BiLSTM embedding layer for encoding long-distance relations allowed [24] to completely forgo the use of context windows in their graph-based model to completely eliminate feature engineering, although in our testing we found that the quality of our model still improved when using a small context window. Partner to this issue is long-distance arc attachments; the parser must somehow recognize that a child $c$ with a distant head $h$ are associated, and 'hold off' on assigning a head to $c$ when it is presented with other candidate heads. Context windows help alleviate this issue, however as we show in Section IV, the BiLSTM layer in our solution helps to encapsulate distant relations, improving the long-distance attachment abilities of our parser.

### C. ARC LABELING

The arc-eager and arc-standard algorithms utilize operations $Left\text{-}Arc_l$ and $Right\text{-}Arc_l$, with $l$ denoting the label assigned to the arc. The set of transitions for the arc-eager algorithm is then $\mathcal{T} = (\{Left\text{-}Arc, Right\text{-}Arc\} \times \mathcal{L}) \cup \{Shift, Reduce\}$, where $\mathcal{L}$ is the set of arc labels. With 37 dependency relations in the Universal Dependency Relations, there are 76 possible transitions. Given that the distribution of these labels will be quite imbalanced — even before separating by left and right arcs there are less than 1,000 instances of the 20 least-common labels in the English Universal Dependencies Treebank out of over 200,000 arcs — and given the large number of labels, the training data gets quite sparse for some transition classes, even on larger treebank.

Our solution to this problem is to have the main classifier decide only between the transitions $\mathcal{T} = \{Left\text{-}Arc, Right\text{-}Arc, Shift, Reduce\}$ and to offload the labeling of arcs to a second multi-layer perceptron (MLP), which takes the feature embeddings from the BiLSTM layer along with the POS tag embedding to produce a label. This approach significantly reduces the sparsity of data for the main classifier and should create much more balanced data. One caveat of this approach is that using two MLPs somewhat increases the computational complexity of the model and likely increases parsing and training time. This impact may be insignificant, however, as when doing joint label and arc classification the complexity of the MLP is likely higher than the complexity of either of the individual perceptrons.

### D. DESIGNING A MULTI-LINGUAL PARSER

A core objective of this paper is to develop a language-agnostic, multi-lingual parser suitable for languages with limited-resources. Some of the main challenges are: firstly, because languages follow very roughly the same structure, there are significant differences in grammatical structure between them. These differences mean that while a given parser architecture may be very effective for one language, it may perform poorly on others. As has been discussed, we attempt to alleviate this problem by using a BiLSTM feature embedding layer to implement 'architecture engineering' as opposed to more classical feature engineering. The architecture of our model allows it to learn language-specific features and eliminate the need for hand-tuning the optimal features for each language. From our results, it is clear that our parser achieves this goal, performing with near-state-of-the-art accuracy on a wide variety of languages. Another core problem to data-driven parsing is the availability of data; whilst some languages such as English have many accurate dataset available, most other languages, particularly languages with limited-resources, have little-to-no data with

**TABLE 5. The hyperparameters for the final model.**

| Layer | Parameter | Value |
|---|---|---|
| MLPs | Nodes in $\text{MLP}_{arc}$ (Layer 1, Layer 2) | 300, 300 |
| | Nodes in $\text{MLP}_{label}$ (Layer 1, 2, 3) | 300, 400, 500 |
| MLP Input | Stack window size | 3 |
| | Buffer window size | 2 |
| BiLSTM | Nodes in each LSTM Cell (BiLSTM Layer 1, 2) | 200, 200 |
| BiLSTM Input | Word Embedding Size | 300 |
| | Tag Embedding Size | 50 |
| Misc | $L_2$ normalisation coefficient | 5e−7 |
| | $\text{MLP}_{arc}$ dropout rate | 0.3 |

which to train a machine learning classifier. As we demonstrate through our results, our parser tackles this problem effectively through transfer learning.

### E. TRANSFER LEARNING

In transfer learning, the knowledge gained by a model in one domain is used to enhance the ability of a model on a different, but related task [21, pp. 526-527]. In this paper, we explore the possibility of training a baseline model on a language with a large amount of available data, such as English, and to then re-train the model on a related language with a considerably smaller dataset, such as Kurmanji. Our hypothesis is that the model might transfer some of the knowledge gained by training on a very large dataset that is not captured by smaller dataset. Having this additional knowledge could improve the accuracy of the parser on the resource-limited language when compared to training on a resource-limited language alone. Our results help to confirm this hypothesis; our parser performance considerably improves when we pre-train it on a large treebank for parsing resource-limited language, such as Kurmanji and Kazakh.

### F. MODEL OPTIMIZATION AND TRAINING

We build our training dataset by using a gold oracle to generate a sequence of parser configurations and their corresponding optimal transitions for each sentence in our training dataset. We generate batches of sentences to be fed to the model, and for each training instance the tokens in the sentence are fed to the BiLSTM layer to generate a dynamic embedding for each token. The arc MLP uses these embeddings to evaluate probabilities for what it thinks is the optimal transition for each configuration, and the softmax cross-entropy loss is calculated over the transition probabilities versus the optimal transition. This loss, plus an $L_2$-regularization loss is minimized using the Adam optimizer [28] over the set of model variables.

Following [7], we train the BiLSTM feature embedding jointly with the parser in order to ensure the embedding is appropriate for the parsing problem. This is in contrast to more traditional methods, where an embedding layer is often built before-hand on a large text corpus using a more general model, such as a word2vec embedding [29] or GloVe embedding [30], [31]. Our parser does, however, have the ability to use pre-trained word embedding to be used in the embedding layer that is used as input to the BiLSTM, however we found the impact of this negligible.

There were a number of hyper-parameters to tune for this model. The ratios/rates for different hyper-parameters are shown in Table 5; primarily the layer sizes and context window sizes. Due to the complexity of the model and size of the English Universal Dependencies treebank, which was used for most of the hyper-parameter tuning, hyper-parameters were tuned using a grid search on a subset of the treebank — 1200 sentences, with additional tuning on 3000 sentences to verify that the parameters scale well — and then the model was trained on the entire dataset. Other model parameters that were experimented with included the $L_2$ regularization coefficient, and the droupout rate. We found that for the labeling MLP, adding dropout slowed training and did not prevent overfitting any more than using $L_2$ regularization alone and so dropout was used only for the label MLP. We theorize that this is due to the relative sparsity of the data — having two similar arc MLP input sequences is unlikely; the BiLSTM layer means that even two configurations with the same words in the context window will not have identical inputs to the multilayer perceptron, as the token embedding will be affected by the tokens outside the context window. That it helped reduce overfitting for labeling, however, suggests that the labeling task was less sparse, and so less dependent on sentence-specific context.

An approach that has been used in the past for training parsing models is error exploration, in which the parser is allowed to make incorrect decisions and attempts to build the most optimal tree given that the mistake has been made [10]. This artificially increases the data size and theoretically allows the parser to 'recover' from mistakes made earlier in the transition sequence. Error exploration requires a dynamic gold oracle, which considerably increases the complexity of the training procedure for a model, and from the ablation experiments of [7] there is very little to be gained from training with error exploration when using architectures like ours; [25] don't utilize error exploration in their joint tagger and parser, and still achieve impressive results. We therefore did not make use of error exploration when training our model. One way we theorize that error exploration may be of use is when training on languages with smaller dataset; the artificial inflation of the dataset through error exploration could help alleviate the issues caused by having a small dataset.

**TABLE 6.** A CoNLL-U formatted parse of *'The man hit the ball.'*

| ID | Token | Lemma | UPOS | XPOS | Features | Head | Relation type | DEPS | Misc |
|---|---|---|---|---|---|---|---|---|---|
| 1 | The | the | DET | DT | - | 2 | det | - | - |
| 2 | man | man | NOUN | NN | - | 3 | nsubj | - | - |
| 3 | hit | hit | VERB | VBD | - | 0 | root | - | - |
| 4 | the | the | DET | DT | - | 5 | det | - | - |
| 5 | ball | ball | NOUN | NN | - | 3 | dobj | - | - |
| 6 | . | . | PUNCT | . | - | 3 | punct | - | - |

### G. DATA

For our experiments, we used the Universal Dependencies treebanks.[1] Universal Dependencies provide treebanks for a number of languages in a common format. The format used by the Universal Dependencies treebanks is CoNLL-U [32], as shown in Table 6. The treebank provides a number of fields for each token in a sentence. These fields are: a numerical token id; the token itself; the lemma or stem of the token; universal part-of-speech tags; language specific part-of-speech tags; additional features of the token; the head of the token; the dependency relation; additional dependency graph detail; and a miscellaneous information field. Most important to our work is the token, lemma, part-of-speech tags, and of course the token's head and dependency relations. Typically, a language processing pipeline would be provided with raw text requiring tokenization, stemming, and tagging before parsing. By using the Universal Dependencies treebanks we are provided with part-of-speech tags, and the lemma field provides the equivalent of stemmed words. This allows us to abstract away the details of acquiring these features and also eliminates the error introduced by imperfect taggers and stemmers, allowing a more direct comparison of our parser with other parser implementations. There are different versions of the treebank available, where each new version introduces data for new languages. The universal dependency treebank has been used extensively in recent years for parsing and part-of-speech tagging of natural language text [33]–[36]. For this study we have used version 2.0 [37]

## IV. RESULTS

To evaluate our baseline parser, we use common metrics for parser evaluation; unlabeled attachment score (UAS), and labeled attachment score (LAS). UAS takes in to account only the head-child dependency relation between two tokens. LAS also considers the head-child dependency relation as well as the labeling of the dependency arcs [38].

To evaluate the accuracy of our parser we make use of the CoNLL-X evaluation script,[2] with some minor modifications to provide more detailed information about long-distance head attachment. It provides a number of useful metrics, such as head attachment accuracy according to dependency distance, and analysis of the most common errors made by the parser.

[1] Available at: http://universaldependencies.org/

[2] Available at: https://github.com/elikip/bist-parser/blob/master/barchybrid/src/utils/eval.pl

**TABLE 7.** An overview of the datasets used for experimentation and testing.

| Treebank | Language | Training Sentences | Testing Sentences |
|---|---|---|---|
| af | Afrikaans | 1315 | 425 |
| en | English | 12543 | 2077 |
| fa_seraji | Farsi | 4798 | 600 |
| kk_ktb | Kazakh | 31 | 1047 |
| kmr_mg | Kurmanji | 20 | 734 |
| tr_imst | Turkish | 3685 | 975 |
| zh_gsd | Chinese | 3997 | 500 |

When evaluating the parser, as is typical with natural language tasks we use three dataset: training set, development set, and test set. The training set is used for training the machine learning classifier, and during training its performance is evaluated on the development set to allow for hyper-parameter tuning of the neural network. Once the optimal hyper-parameters for the development set have been found, the overall performance of the model is evaluated on the testing dataset. This guarantees that the model generalizes well to unseen data (such as test data). The Universal Dependencies dataset provide each of these dataset for many language to ensure consistency when evaluating multiple parsers.

For a point of comparison, we look at the performance of various parsers submitted to the CoNLL 2017 Shared Task[3] for two reasons: (i) because we are using the same dataset that are used by the parsers in that shared task, and (ii) we are evaluating our parser using the same standard evaluation metrics used in that shared task as opposed to the new evaluation metrics used in the CoNLL Shared Task 2018 [39]. Thus we can report direct comparison of our paser with the the top performing parsers in the shared task. The detail of the dataset size for different languages is shown in Table 7.

To evaluate our parser's versatility and language independent feature, we experimented on languages with different grammatical complexities and dataset sizes. Choosing languages with diverse linguistic characteristics ensured that we were not optimizing our parser too heavily for one language family, as a large motivation for this paper was to build a parser that performs well on any language. We segment our languages in to 'large', 'medium' and 'small' treebanks. Our large treebank languages include English, Persian, Turkish and Chinese. Along side these we tested Afrikaans to gain an idea of the applicability of transfer learning to medium-resource languages and chose Kazakh and Kurmanji to experiment with transfer learning on less-resource languages,

[3] http://universaldependencies.org/conll17/

**TABLE 8.** Unlabeled attachment scores and labeled attachment scores on English, Afrikaans, Kurmanji, Persian, Turkish, Kazakh and Chinese versus the best scoring parser in the CoNLL 2017 shared task for different languages.

| | This Work | | CoNLL 2017 | |
|---|---|---|---|---|
| Treebank | UAS | LAS | UAS | LAS |
| af | 66.95 | 51.44 | - | - |
| en | 80.70 | 65.95 | 84.74 | 82.23 |
| fa_seraji | 82.49 | 62.13 | 89.64 | 86.31 |
| kk_ktb | 52.96 | 29.84 | 45.72 | 29.22 |
| kmr_mg | 53.68 | 35.91 | 54.73 | 47.53 |
| tr_imst | 54.72 | 38.05 | 69.62 | 62.79 |
| zh_gsd | 70.10 | 46.44 | 72.39 | 68.56 |

**TABLE 9.** Precision, Recall, and $F_1$ score of head attachments by attachment distance on the English web Treebank universal dependencies dataset.

| | This Work | | | Stanford Parser | | |
|---|---|---|---|---|---|---|
| Distance | Precision | Recall | $F_1$ | Precision | Recall | $F_1$ |
| To root | 91.12 | 88.09 | **89.58** | 84.68 | 84.52 | 84.60 |
| 1 | 89.99 | 94.52 | **92.20** | 91.43 | 92.19 | 91.81 |
| 2 | 81.80 | 74.30 | 77.87 | 85.87 | 88.34 | **87.09** |
| 3 | 78.36 | 73.45 | 75.83 | 80.69 | 81.15 | **80.92** |
| 4 | 71.34 | 71.44 | 71.39 | 73.98 | 71.97 | **72.96** |
| 5 | 68.08 | 61.85 | **64.82** | 64.26 | 61.90 | 63.06 |
| 6 | 56.36 | 56.78 | 56.59 | 61.68 | 57.02 | **60.42** |
| 7+ | 61.70 | 66.71 | **64.12** | 64.45 | 58.15 | 61.14 |

where the training set is between 20 and 31 sentences. Each resource-limited language was fairly close linguistically to a larger language; Afrikaans to English, Kurmanji to Persian, and Kazakh to Turkish. We attempt to exploit linguistic similarities among the selected languages when using transfer learning.

As shown in Table 8, our parser shows very promising unlabeled attachment scores across most languages, coming close to the most accurate parser submitted to the shared task on parsing in English, Kurmanji, and exceeding the best parser in Kazakh, demonstrating our parser's versatility in parsing a variety of language families with good accuracy. While our parser shows good head attachment accuracy, labeling accuracy was lower than the top state-of-the-art parsers, for all languages other than Kazakh, however on some treebanks, such as Kurmanji, we still would have placed in the top ten parsers for LAS, out of 33 submitted to the shared task.

Interestingly, our parser appears to perform very well on very resource-limited languages, performing better than state-of-the-art on Kazakh and coming very close on Kurmanji, the two smallest treebanks we used to evaluate our parser.

### A. MODEL OPTIMIZATION AND HYPERPARAMETER TUNING

Through hyper-parameter tuning with a relatively fine-grained grid search, we found optimal parameters detailed in Table 5. Particularly interesting were the stack and buffer window sizes and experiments to optimize which features to use. The most optimal window sizes were 3 and 2 for the stack and buffer respectively. We theorize that when increasing the size of the windows, the data sparsity increased, leading to a reduced ability for the parser to generalize for larger window sizes. In a similar vein, we found that using language-specific part-of-speech tags yielded considerably poorer accuracy on both the training and development datasets than the universal part-of-speech tags, and we attribute this to the sparsity of the data when using language-specific tags. The difference between training and development accuracy is larger when using language-specific tags than universal tags (4.05% and 2.65%, respectively). We found that when using language-specific tags more overfitting is occurring, reducing model's ability to generalize to unseen sentences.

### B. LONG DISTANCE DEPENDENCIES

Comparing our parser on parsing long distance dependencies with the Stanford parser (the best performing parser in the CoNLL 2017 Shared Task), it can be noted from Table 9 that despite being slightly less accurate overall, our parser excels at long-distance parsing; for attaching arcs a distance greater than 6 tokens away, and for tokens a distance of 5 away, our parser out-performs the Stanford parser with regards to head attachment $F_1$ score. This reinforces our hypothesis that the BiLSTM embedding layer we use allows for more accurate long-distance head attachment by encoding information about these long-distance dependencies in the token embedding generated by the BiLSTM.

### C. TRANSFER LEARNING

Making use of Deep Learning for transfer learning proved very effective across different languages. Comparing our results to the CoNLL 2017 Shared Task, we find that our parser performs with accuracy above the state-of-the-art for languages with small dataset when pre-training the model on a language with more available data.

With lower than expected results when pre-training the Kazakh model on Turkish (where Kazakh and Turkish very related to each other), we anticipated that this was a result of our parser's poor accuracy on Turkish. By pre-training the Kazakh model on the English dataset, we achieved much higher UAS and LAS scores, indicating that an accurate pre-trained model on a language from the same language family can be a better choice for transfer learning than an inaccurate but related language's model. As shown in Table 10. The parser accuracy improved by 13.84% and 4.83% for UAS and LAS, respectively. Moreover, pre-training our parser on Farsi dataset for parsing Kurmanji we achieve an improved UAS of 6.56% and LAS by 20.68%. Finally, pre-training our parser on English dataset for parsing Kazakh, the parser's accuracy is improved by 8% and 6.94% for UAS and LAS. respectively.

Table 11 show the parser performance compared to the top performing parser in the CoNLL Shared Task 2017 for Kurmanji and Kazakh. Our parser achieves higher accuracy than the stat-of-the-art parser. Our proposed model achieve a

**TABLE 10. Unlabeled attachment scores and labeled attachment scores on languages with and without pre-training (p.t.) on larger dataset. en = English, fa_seraji = Farsi, and tr_imst = Turkish.**

| Treebank | Parser | UAS | LAS |
|---|---|---|---|
| af | This work | 66.59 | 51.44 |
| | This work (p.t. en) | **80.33** | **56.27** |
| kk_ktb | This work | 52.96 | 29.84 |
| | This work (p.t. tr_imst) | 52.13 | **32.26** |
| | This work (p.t. en) | **60.96** | **36.78** |
| kmr_mg | This work | 53.68 | 35.91 |
| | This work (p.t. fa_seraji) | **60.24** | **56.59** |

**TABLE 11. Unlabeled attachment scores and labeled attachment scores on languages with pre-training (p.t.) on larger dataset versus the highest accuracy parser in the CoNLL 2017 shared task for two of the most resource-limited languages.**

| Treebank | Parser | UAS | LAS |
|---|---|---|---|
| kmr_mg | Best CoNLL 2017 | 54.73 | 47.53 |
| | This work (p.t. fa_seraji) | **60.24** | **56.59** |
| kk_ktb | Best CoNLL 2017 | 45.72 | 29.22 |
| | This work (p.t. en) | **60.96** | **36.78** |

difference of 5.51% for UAS and 9.06% for LAS on Kurmanji and 15.24% for UAS and 7.56% for LAS on Kazakh.

## V. CONCLUSION

In this paper we have presented a deep learning based architecture for a natural language parser, and have investigated the qualities of our parser that allow it to do so.

The Universal Dependencies dataset used both by us and the CoNLL Shared Task 2017 provide the ability to compare our parsers performance between languages on standardized dataset, and to easily investigate the effectiveness of transfer learning to natural natural text parsing.

A Particularly interesting result is our parser's performance on resource-limited languages, consistently performing at, or better than the level of the best known parsers. We further improved on our advances in this critical problem for natural language processing tasks by demonstrating highly effective transfer learning techniques to achieve results far above the state-of-the-art in Kurmanji and Kazakh.

That our parsing architecture performs comparably to state-of-the-art parsers across a range of language families demonstrates that BiLSTM feature embeddings allow for effective multi-lingual parsing, setting our parser out from other state-of-the-art parsers which are often more specialized for certain language families. Effectively eliminating feature engineering, the BiLSTM allows the parser to adapt to the target language through architecture engineering. The BiLSTM layer also helps to solve issues such as long-distance head attachments which are common problems when parsing languages. Our architecture also shows very promising results when parsing languages with very limited resources.

As a particularly important challenge in data-driven parsing, we have also shown that BiLSTM based parser can utilize transfer learning to further improve parsing performance on resource-limited languages. By pre-training our model on related languages, we instill more general linguistic features of the language family not captured by a smaller dataset, transfer learning proves to be a very effective method for enhancing parsing performance, achieving better than state-of-the-art parsing accuracy on two topologically different resource-limited languages.

One approach considered for this paper was to attempt to exploit the sequential nature of shift-reduce algorithms, and to utilize a Long Short Term Memory (LSTM) to classify which transition to make. Our theory is that it may be possible that common patterns in sentence structure, such as 'DET NOUN VERB DET NOUN', as in '*The man hit the ball*', may be able to be captured by the LSTM. An LSTM, however, is significantly more computationally complex than a Multi-layer Perceptron (MLP), and a large motivation for us using a BiLSTM was to encode this type of structural information in to the token embeddings; using an LSTM for parsing may be better utilized alongside other parsing methodologies.

With the limitations of our arc labeling multi-layer perceptron holding back our parser somewhat, and while [7] found success with a separate MLP for arc labeling on the Penn Treebank and Chinese Treebank datasets, research in to how to better label arcs given a feature embedding layer could prove valuable. This split $\text{MLP}_{arc}$ and $\text{MLP}_{label}$ architecture may be useful in other parsing architectures, and may prove more effective if the embedding is trained jointly on both the arc and labeler MLPs.

## REFERENCES

[1] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2000.

[2] D. Jurafsky, and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. London, U.K.: Pearson Education, 2016.

[3] S. Jaf, "The application of constraint rules to data-driven parsing," Ph.D. dissertation, Dept. School Comput. Sci., Univ. of Manchester, England, U.K. 2015.

[4] A. Farghaly and K. Shaalan, "Arabic natural language processing: Challenges and solutions," *ACM Trans. Asian Lang. Inf. Process.*, vol. 8, no. 4, p. 14, 2009.

[5] M. Collins, "Head-driven statistical models for natural language parsing," *Comput. Linguistics*, vol. 29, no. 4, pp. 589–637, 2003. doi: 10.1162/089120103322753356.

[6] L. Tesnière, *Éléments de Syntaxe Structurale*, E. Klincksieck, Eds. Paris, France, Klincksieck, 1959.

[7] E. Kiperwasser and Y. Goldberg, "Simple and accurate dependency parsing using bidirectional lstm feature representations," *Trans. Assoc. Comput. Linguistics*, vol. 4, no. 1, pp. 313–327, 2016. doi: 10.1162/tacl_a_00101

[8] S. P. Abney, "Parsing by chunks," in *Principle-Based Parsing Computation and Psycholinguistics*. Norwell, MA, USA: Kluwer, 1991, pp. 257–278.

[9] J. Nivre, "An efficient algorithm for projective dependency parsing," in *Proc. 8th Int. Workshop Parsing Technol.*, 2003, pp. 149–160.

[10] Y. Goldberg and J. Nivre, "A dynamic oracle for arc-eager dependency parsing," in *Proc. COLING*, 2012, pp. 959–976.

[11] C. Gómez-Rodríguez and D. Fernández-González, "An efficient dynamic oracle for unrestricted non-projective parsing," in *Proc. 53rd Annu. Meeting Assoc. Comput. Linguistics 7th Int. Joint Conf. Natural Lang. Process.*, 2015, pp. 256–261.

[12] D. Chen and C. D. Manning, "A fast and accurate dependency parser using neural networks," in *Proc. EMNLP*, A. Moschitti, B. Pang, and W. Daelemans, Eds., 2014, pp. 740–750.
[13] J. Nivre and M. Scholz, "Deterministic dependency parsing of English text," in *Proc. 20th Int. Conf. Comput. Linguistics*, 2004, p. 64.
[14] J. Nivre, "Algorithms for deterministic incremental dependency parsing," *Comput. Linguistics*, vol. 34, no. 4, pp. 513–553, Dec. 2008. doi: 10.1162/coli.07-056-R1-07-027
[15] W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch, "TiMBL: Tilburg memory based learner, version 5.0, reference guide," Tilburg Univ., Tilburg, The Netherlands, Tech. Rep., 03-10, 2003.
[16] H. Yamada and Y. Matsumoto, "Statistical dependency analysis with support vector machines," in *Proc. IWPT*, 2003, pp. 195–206.
[17] Y. Zhao and Q. Zhou, "A SVM-based model for Chinese functional chunk parsing," in *Proc. 5th SIGHAN Workshop Chin. Lang. Process.*, 2006, pp. 94–101.
[18] A. V. M. Barone and G. Attardi, "Dependency parsing domain adaptation using transductive SVM," in *Proc. Joint Workshop Unsupervised Semi-Supervised Learn. NLP*, 2012, pp. 55–59.
[19] S. Jaf and A. Ramsay, "The selection a classifier for data-driven parsing," in *Proc. 12th Int. Workshop Natural Lang. Process. Cogn. Sci.*, Kraków, Poland, Libreria Editrice Cafoscarina, 2015, pp. 39–49.
[20] J. Nivre, J. Hall, J. Nilsson, A. Chanev, G. Eryigit, S. Kübler, S. Marinov, and E. Marsi, "MaltParser: A language independent system for data-driven dependency parsing," *Natural Lang. Eng.*, vol. 13, pp. 95–135, 2007.
[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[22] C. Dyer, M. Ballesteros, W. Ling, A. Matthews, and N. A. Smith, "Transition-based dependency parsing with stack long short-term memory," in *Proc. 53rd Annu. Meeting Assoc. Comput. Linguistics 7th Int. Joint Conf. Natural Lang. Process.*, 2015, pp. 334–343.
[23] M. Ballesteros, C. Dyer, and N. A. Smith, "Improved transition-based parsing by modeling characters instead of words with LSTMs," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2015, pp. 349–359.
[24] W. Wang and B. Chang, "Graph-based dependency parsing with bidirectional LSTM," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 2306–2315.
[25] D. Q. Nguyen, M. Dras, and M. Johnson, "A novel neural network model for joint POS tagging and graph-based dependency parsing," in *Proc. CoNLL Shared Task Multilingual Parsing From Raw Text Univ. Dependencies*, 2017, pp. 134–142.
[26] M. A. Covington, "A fundamental algorithm for dependency parsing," in *Proc. 39th Annu. Southeast Conf.*, 2001, pp. 95–102.
[27] W. Pei, T. Ge, and B. Chang, "An effective neural network model for graph-based dependency parsing," in *Proc. 53rd Annu. Meeting Assoc. Comput. Linguistics 7th Int. Joint Conf. Natural Lang. Process.*, 2015, pp. 313–322.
[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," Dec. 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980
[29] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, Lake Tahoe, NV, USA, vol. 2, 2013, pp. 3111–3119. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999792.2999959
[30] E. Grave, P. Bojanowski, P. Gupta, A. Joulin, and T. Mikolov, "Learning word vectors for 157 languages," in *Proc. 11th Int. Conf. Lang. Resour. Eval. (LREC)*, 2018, pp. 3483–3487.
[31] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Doha, Qatar, Oct. 2014, pp. 1532–1543. [Online]. Available: https://www.aclweb.org/anthology/D14-1162
[32] D. ependencies. (2014). *CoNLL-U Format*. Accessed: Feb. 21, 2019. [Online]. Available: http://universaldependencies.org/format.html
[33] T. Boros, S. D. Dumitrescu, and R. Burtica, "NLP-Cube: End-to-end raw text processing with neural networks," in *CoNLL Shared Task Multilingual Parsing From Raw Text Universal Dependencies*. Brussels, Belgium, 2018, pp. 171–179. [Online]. Available: https://www.aclweb.org/anthology/K18-2017
[34] Z. Li, S. He, Z. Zhang, and H. Zhao, "Joint learning of pos and dependencies for multilingual universal dependency parsing," in *CoNLL Shared Task Multilingual Parsing From Raw Text Universal Dependencies*. Brussels, Belgium, Oct. 2018, pp. 65–73. [Online]. Available: https://www.aclweb.org/anthology/K18-2006
[35] D. Q. Nguyen and K. Verspoor, "An improved neural network model for joint POS tagging and dependency parsing," in *CoNLL Shared Task Multilingual Parsing From Raw Text Universal Dependencies*. Brussels, Belgium Association for Computational Linguistics, Oct. 2018, pp. 81–91. [Online]. Available: https://www.aclweb.org/anthology/K18-2008
[36] G. Arakelyan, K. Hambardzumyan, and H. Khachatrian, "Towards jointUD: Part-of-speech tagging and lemmatization using recurrent neural networks," in *CoNLL Shared Task Multilingual Parsing From Raw Text Universal Dependencies*. Brussels, Belgium, Association for Computational Linguistics, 2018, pp. 180–186. [Online]. Available: https://www.aclweb.org/anthology/K18-2018
[37] J. Nivre *et al.*, "Universal dependencies 2.0," *LINDAT/CLARIN Digital Library Institute Formal and Applied Linguistics (ÚFAL)*. Staré Město, Czechia: Charles Univ., 2017. [Online]. Available: http://hdl.handle.net/11234/1-1983
[38] S. Kubler, R. McDonald, J. Nivre, and G. Hirst, *Dependency Parsing*. Morgan and Claypool Publishers, San Rafael, CA, USA, 2009.
[39] D. Zeman and J. Hajič, Eds., *CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Brussels, Belgium: Association for Computational Linguistics, 2018. [Online]. Available: http://www.aclweb.org/anthology/K18-2

**SARDAR JAF** received the Ph.D. degree from The University of Manchester, U.K. He is currently a Senior Lecturer of computer science with the University of Sunderland, U.K. His research interests include artificial intelligence, data science, and cybersecurity, especially natural language processing, including parsing, textual entailment, information retrieval, and language identification.

**CALUM CALDER** received the master's degree in computer science from Durham University, U.K. He is currently a Software Engineer with Bloomberg L.P. His research interests include data driven natural language processing and the acceleration of random forest classifiers using GPU computing. He has a strong passion for machine learning and data analytics. He has explored the application shallow and deep learning algorithms to various classification tasks.