Multilingual Natural Language Parsing with BiLSTM Feature Embeddings and Transfer Learning

Student Name: C. Calder

Supervisor Name: S. Jaf

Submitted as part of the degree of M.Eng. Computer Science to the

Board of Examiners in the School of Engineering and Computing Sciences, Durham University

25 April 2018

***Abstract* —** Problems in natural language processing are becoming increasingly important, including speech recognition, text-based data mining, and natural language generation. Before many of these tasks can be approached effectively, it is necessary to understand the syntactic structures of the sentences being processed. 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 in larger language processing pipelines as a more detailed and useful data structure to analyse than raw text alone. In this paper, we explore methods to design a multi-lingual dependency parser. Using modern deep learning techniques including a BiLSTM feature embedding layer, 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 achieve state-of-the-art results on many languages in the Universal Dependencies treebanks, with a UAS score of 80.70 on the English Web Treebank, and performing at better than state-of-the-art on Kazakh. By utilising transfer learning, 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.

***Keywords* —** Natural Language Processing, Parsing, Deep Learning, BiLSTM, Dependency Parsing,

I INTRODUCTION

Natural language parsing problems involve determining the syntactic 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.

Determining a parse tree of a sentence forms the basis for many other natural language tasks, particularly semantic analysis tasks (Jurafsky and Martin, 2000), such as sentiment analysis, information extraction, and question interpretation by proving a data structure which encodes more information about a piece of text than the raw text alone.

A Dependency Parsing

Dependency parsing is based on the dependency relation formalism. In dependency relations, tokens are associated with each other directly. Each token excluding the tree's root token is

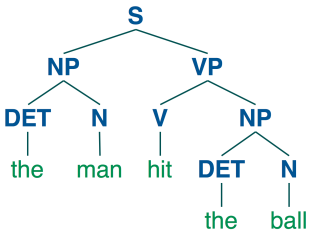

Figure 1: A constituency tree, representing a parsing of the sentence '*the man hit the ball*'.

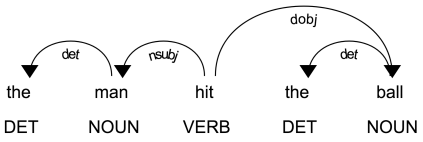

Figure 2: A dependency relation, representing a parsing of the sentence '*the man hit the ball*'.

dependent on an associated 'head' token. Dependents are associated with their head through 'dependency arcs', which can optionally be labeled to provide additional information about the relation. Associating tokens via a dependency arc is sometimes called 'head attachment'.

Dependency grammars have less rigid structural rules when compared to other representations, such as constituency grammars, as they don't depend on a set of grammar rules to determine how elements can be related. While this means that a dependency parsing of a sentence may not necessarily be grammatically correct, it does open dependency parsing up to represent languages that are less rigid, such as those with free word order or compound words.

In the dependency relation formalism, a distinction is made between 'projective' and 'non-projective' dependency trees. Projective trees are, in rough terms, dependency trees without any intersecting arcs (see Nivre, 2008, for a more rigorous definition).

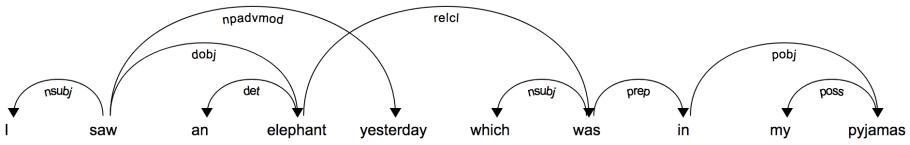

Figure 3: An example of a non-projective parse tree.

B Grammar Driven Constituency Parsing

Grammar driven parsing models language as a formal grammar, typically using probabilistic context free grammars or some extension of them. Modelling language as a formal grammar gives a hierarchical view of sentence structure where each 'constituent', represented by a node in the parse tree, is either a leaf node associated with some token in the sentence or an internal node with some constituents of its own as children. This forms the basis for constituency parsing.

That the model is a formal grammar allows common language structures to be represented. For example, in English, a common pattern is 'VERB DET NOUN', such as 'hit the ball' in Figure 1. The most common approach to grammar driven parsing is dynamic programming, in particular, the Cocke-Younger-Kasami algorithm (Jurafsky and Martin, 2000).

C Data Driven Parsing

As we discuss in Section D, ambiguity is a fundamental problem to natural language parsing; that there will be multiple valid parse trees for some sentences means that some mechanisms

must be put in place to decide which of these trees is the 'most correct' parse. To address this problem, statistical parsing was introduced, in which a probability is assigned to each of the valid parse trees. To generate these probabilities, a corpus — a large collection of text, gold-standard parse trees, or other linguistic data — is used.

Taking this notion further, data driven parsing moves away from hand-crafted grammars and attempts to learn the grammar itself. In constituency parsing, chunk based parsing reduces the requirement for predefining grammars by breaking a sentence in to 'chunks' which are defined over very simple grammars, and then using a classifier to group chunks in a less rigid fashion. Dependency parsing, being mostly unrelated to formal grammars, lends itself very well to data driven parsing. By eliminating the need for hand-crafted grammars, designing a data-driven parser is much less reliant on specialist linguistic knowledge. The dependence upon data, however, does introduce with it the task of generating quality treebanks or data sources. For languages with fewer speakers this becomes an issue, and some languages have little, or no available data.

Along with varying amounts of data, designing a parser that is effective on multiple languages also 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's reasonable to expect that the most accurate parsers for a single language would have features designed around that language.

D Ambiguity

A fundamental problem in parsing is the inherent ambiguity in language; 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, 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.

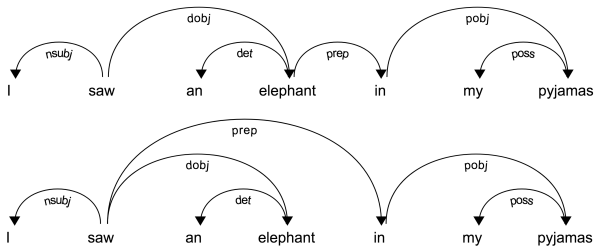

Figure 4: Two interpretations of '*I saw an elephant in my pyjamas*' In the lower sentence, the person seeing the elephant is wearing pyjamas, and in the top interpretation the elephant is wearing pyjamas.

E Current State of the Art Parsers

The two dominant approaches to dependency parsing are shift-reduce, or transition-based parsing, and graph-based parsing. In shift-reduce parsing algorithms the input is processed incrementally from left-to-right, with the parser making transitions between states according to the current configuration, generally guided by some classifier. Graph-based dependency parsing takes a very different approach, and instead operates over the connected digraph with a vertex representing each token in the sentence.

As the core component of many parsers is some data-driven model, machine learning models 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. 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.

F Project Objectives

The objectives of this project are to assess the viability of deep learning for developing a multi-lingual, data-driven parser. We introduce a BiLSTM feature embedding layer to improve long distance parsing that is often an issue with transition-based parsers, and to reduce feature engineering, which in turn improves language-agnosticism. As well as this, we make use of transfer learning in an attempt to improve the accuracy of models trained on languages with limited resources.

Using these approaches, we primarily address two questions; *How can we build a parser that is effective on multiple languages?* and *How can we train effective models for languages with limited data?*. Our results are very promising, with state-of-the-art performance on most languages, and with impressive improvements on resource-limited languages, to the extent that pre-training our parser on similar languages to the target language exceeds current state-of-the-art results by a large margin.

II RELATED WORK

A Early Research

Much of the syntactic representation of sentences in the styles we use today can be dated back to Chomsky's seminal work *Syntactic Structure* (Chomsky, 1957). Chomsky defined phrase structure grammars and phrase structure rules by developing the work of Leonard Bloomfield on Immediate Constituent analysis in *Language* (Bloomfield, 1933). Phrase structure grammars now form the foundation for constituency parsing (Jurafsky and Martin, 2000, p. 195).

The development of dependency grammars is typically associated with the work of Lucien Tesnière in *Éléments de syntaxe structurale* (Jurafsky and Martin, 2000, p. 268). While the development of these formalisms of natural language were developed around the same time, research has tended to focus around phrase structure grammars. This is beginning to change, however, with the rise in popularity of dependency grammar relations. See for instance the Depling conference, which started in 2013 and centres around computational linguistics utilising dependency structures.

B Grammar Driven Parsers

Chomsky's phrase structure grammar defines language as a formal grammar. As with other formal grammars, we can attempt to acquire the derivation and trivially construct the parse tree. Much of the early work on parsing grammars was driven by the development of the first programming languages and their compilers, such as Donald Knuth's left-to-right parser (Knuth, 1965).

While parsers for deterministic grammars form a useful basis for language parsing, work as early as Chomsky's modeled language on probabilistic grammars (Chomsky, 1957) due to

the inherent ambiguity of natural language. Early work on probabilistic context free grammars (PCFGs), such as Booth's *Probabilistic Representation of Formal Languages* (Booth, 1969) gave rise to the first data-driven parsers. *Trainable Grammars for Speech Recognition* (Baker, 1979) builds on the concepts of PCFGs, and discusses ways to train algorithms for grammar inference on arbitrary CFGs. Methods such as this, which utilise a corpus or treebank to build a statistical model of a grammar, allow parsers to approximate the probability of a given parse when given ambiguous sentences. Early examples of this include the PARSIFAL system (Marcus, 1980).

C Transition Based Parsers

Based on shift-reduce algorithms used for parsing deterministic CFGs, transition based algorithms were developed for natural language parsing that instead operate on probabilistic CFGs (Abney, 1991). More recently, similar methods were developed for incrementally parse dependency trees (Nivre, 2003, 2008). Nivre's algorithms allow parsing of a projective 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'. As well as parsing projective dependency trees, Nivre's algorithms have been extended to allow parsing of non-projective trees (Covington, 2001; Nivre, 2008, 2009).

Gold oracles — oracles that when given a perfectly parsed 'gold' tree provide a transition sequence that produces the tree — are used for producing 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 Algorithm 1 in Goldberg and Nivre (2012) for Nivre's arc-eager parser. One issue with typical gold oracles is that they depend on all previous transitions to be correct (Goldberg and Nivre, 2012; Gómez-Rodríguez and Fernández-González, 2015). 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 a mistake during parsing allows the parser to learn to recover from mistakes made during parsing. As transition based parsing is very dependent on previous transitions (Chen and Manning, 2014), this method has been used in many parsers in an attempt to improve accuracy (Goldberg and Nivre, 2012; Gómez-Rodríguez and Fernández-González, 2015; Kiperwasser and Goldberg, 2016) with varying success.

C.1 Graph Based Parsers

Along with incremental dependency parsing, based on work by McDonald et al. (2005), graph based dependency parsing has gained popularity. McDonald's method models the parse tree as a minimum spanning tree over a fully connected graph where each word is a node. This allows typical spanning tree algorithms to be used, such as Chu-Liu/Edmond's Algorithm to achieve $O(n^2)$ parsing speed in the length of the sentence (Chu and Liu, 1965; Edmonds, 1967). Alongside transition based parsing, graph based models have also been used in many recent parsing implementations, with performance comparable to state of the art transition based parsers (Pei et al., 2015; Wang and Chang, 2016).

D Data Driven Parsing with Machine Learning

Techniques such as graph based parsing and Nivre's shift-reduce algorithms require some way to assign a score to each arc or to each transition given the configuration of the parser. In Nivre's papers (Nivre and Scholz, 2004; Nivre, 2008), for example, he 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, Nivre and Scholz (2004) present a concrete implementation of a dependency parser which utilises instance-based learning through TiMBL, a framework based on the k-Nearest Neighbour algorithm (Daelemans et al., 2003). In a similar effort, Yamada and Matsumoto (2003) propose a dependency parser which utilises a Support Vector Machine to infer transitions for their own shift-reduce algorithm. SVMs have been used quite successfully in parsing tasks, including both chunk parsing methods for constituency parsing (Zhao and Zhou, 2006), and dependency parsing through shift-reduce algorithms (Nivre, 2008; Yamada and Matsumoto, 2003). A particularly noteworthy parser is MaltParser (Nivre and Hall, 2005) which has options to utilise either an SVM, or TiMBL as its underlying learning algorithm and was regarded as state-of-the-art for a time.

E Deep Learning for Parsing

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

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 (Chen and Manning, 2014). Chen and Manning were able to significantly reduce the amount of feature engineering required by utilising 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 to deep learning for parsing has had similar aims — Kiperwasser and Goldberg (2016) present a parser utilising a BiLSTM layer, taking only the sequence of words and associated POS tags 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.

E.1 LSTMs

Recurrent neural networks operate on sequential data, and as such have been used frequently in natural language tasks (Goodfellow et al., 2016, p. 163). LSTMs in particular have been used with some success, including in parsing tasks (Dyer et al., 2015; Ballesteros et al., 2015). 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 (Dyer et al., 2015). An alternative approach utilised in very recent years uses Bidirectional-LSTMs; 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. Wang and Chang (2016), for example, use the feature embedding as a preprocessing step in their graph-based parser, while Kiperwasser and Goldberg (2016) and Nguyen et al. (2017) utilise a BiLSTM feature embedding as an embedding layer to transition-based parsers, achieving good results with only a simple multi-layer perceptron 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.

III SOLUTION

For experimentation and testing, we designed our system in python. We created a parsing framework based on Nivre's arc-eager algorithm. To implement our parser model, we used tensorflow, a leading open-source deep learning framework with python bindings which is seeing a large amount of use in both industrial and research settings.

A Transition Based Parsing

In this paper, we elected to use a transition-based parser. Nivre's algorithms are simple to implement given their straightforward design, and many state-of-the-art parsers make use of either the arc-standard or arc-eager shift-reduce algorithms. This allows us to compare the performance of our parser to that of other, similar parsers, and gain insight in to how the specific design decision made in this paper affect the performance of the parser. Whilst simple to implement and understand, Nivre's algorithms remaining effective as parsing algorithms, allowing design efforts to focus primarily on the neural network model. One particularly appealing aspect of Nivre's 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 (Nivre, 2003, 2008) to outline the arc-standard and arc-eager parsing algorithms.

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

Figure 5: The transitions defined by Nivre's arc-eager parser.

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

Figure 6: The transitions defined by Nivre's arc-standard parser.

Nivre's algorithms maintain a parser state, or 'configuration', which is mutated via some simple state transition functions. Configurations are generally based around at least one 'stack'

σ which maintains an ordered list of the tokens currently being operated on, a 'buffer' β 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 utilising 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 might substitute the parsing algorithm for Covington's algorithm (Covington, 2001), or Nivre's list-based parser (Nivre, 2008) for an expected $O(n)$ (worst-cast $O(n^2)$)-time non-projective parser.

Graph-based approaches have also proved effective when used alongside machine learning models, and from Kiperwasser and Goldberg (2016), we see that when using BiLSTM feature embeddings the performance of graph-based models may infact be greater than transition-based systems for long-distance parsing, such as for parsing Chinese sentences. Graph-based algorithms such as the MST Parser also parse non-projective trees with no modifications required. 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.

For our experiments, we used Nivre's arc-eager parser. The arc-eager algorithm attempts to produce shorter parse sequences than other shift-reduce parsers by connecting arcs as early as possible (Nivre, 2008), 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 and bottom of the buffer 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. By allowing 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 behaviour, see Figures 7 and 8 for the parsing of '*We*$_1$ *swam*$_2$ *at*$_3$ *the*$_4$ *beach*$_5$'.

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

Figure 7: Parsing '*We*$_1$ *swam*$_2$ *at*$_3$ *the*$_4$ *beach*$_5$' using Nivre's arc-eager parser.

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

Figure 8: Parsing '*We swam at the beach*' using Nivre's arc-standard parser.

Notice that the parsing diverges at step 3, as we can't yet form a right arc between *swam*$_2$

and at_3 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 being assigned their proper head. The Right-Arc operation joining '$swam_2$' and '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 Nguyen et al. (2017) and Kiperwasser and Goldberg (2016) — utilising a BiLSTM to first build a feature embedding layer, and take the output of the BiLSTM as the input to a more typical multi-layer perceptron, as in Chen and Manning (2014). The BiLSTM provides many beneficial features; primarily, it encodes positional information, and it also significantly reduces the need for feature engineering.

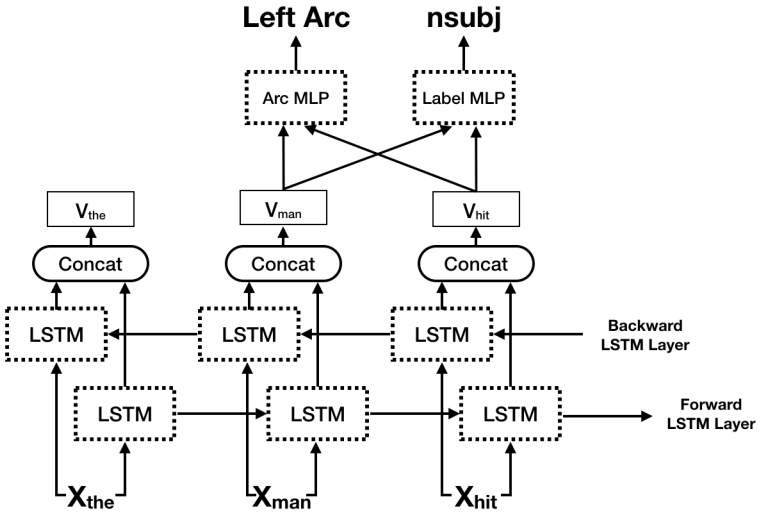

Figure 9: A simplified view of the parser architecture part way through parsing the sentence '*the man hit the ball*': The BiLSTM layer builds a feature embedding over the word and POS embeddings X_t of the input tokens t, and given a parser state $(\sigma|man,hit|\beta,A)$ the multi-layer perceptrons for classifying the arc and label are given the BiLSTM embedded vectors for 'man' and 'hit' as input.

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

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, utilising 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 $s_1 \dots s_n$ and bottom m tokens of the buffer $b_1 \dots b_m$ to obtain $v = v_{s_1}, \dots, v_{s_n}, v_{b_1}, \dots, v_{b_n}$, where m and n are hyperparameters 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 vs 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

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

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 is

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

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; Chen and Manning (2014) use only 18 input features in their model, and similarly, Pei et al. (2015) 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' (Kiperwasser and Goldberg, 2016). 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 a primary goal of this project was to design a multi-lingual parser, the reduction of feature engineering was particularly important when designing our model's 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. 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 lookahead 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 (Chen and Manning, 2014; Pei et al., 2015).

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 (Chen and Manning, 2014). The use of context windows by early parsers gives the parsing models the ability to look somewhat in to the future of the parsing sequence helps to eliminate this, however the effectiveness of a BiLSTM embedding layer for encoding long-distance relations allowed Wang and Chang (2016) 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 recognise 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 utilise 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 treebanks.

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, which takes the feature embeddings from the BiLSTM layer along with the POS tag embedding to produce a label. This 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 aim of this paper was to develop a language-agnostic, multi-lingual parser. This comes with a few challenges; firstly, while languages follow very roughly the same structure, there are significant differences in grammatical structure between languages. 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 and Chinese have many accurate datasets available, most other languages, particularly those with fewer fluent speakers, have little-to-no data with which to train a model. Quite relevant to this paper is the upcoming CoNLL 2018 Shared Task, whose focus is to continue the work of their 2017 Shared Task by constructing a parser to parse raw text over many topologically different languages, particularly those with limited resources (CoNLL, 2018). As we shall demonstrate through our results, our parser tackles this problem effectively both through the architecture of our model.

D.1 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 (Goodfellow et al., 2016, 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 Afrikaans. 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 datasets. Having this additional knowledge could improve the accuracy of the parser on the resource-limited languages when compared to training on a resource-limited language alone. Our results help to confirm this hypothesis; our parser performs with significantly higher accuracy than other state-of-the-art parsers on a variety of languages when using transfer learning.

E Model Optimisation 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-regularisation loss is minimized using the Adam optimizer (Kingma and Ba, 2015) over the set of model variables.

Layer	Parameter	Value
MLPs	Nodes in MLP_{arc} (Layer 1, Layer 2)	300, 300
	Nodes in 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	$5\text{e}{-7}$
	MLP_{arc} dropout rate	0.3

Table 1: The hyperparameters for the final model.

Following Kiperwasser and Goldberg (2016), 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. Our parser does, however, have the ability to use pre-trained word embeddings 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 hyperparameters to tune for this model; primarily the layer sizes and context window sizes. The final hyperparameter values are outilined in Table 1 Due to the

complexity of the model and size of the English Universal Dependencies treebank, which was used for most of the hyperparameter tuning, hyperparameters 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 dateset. Other model parameters that were experimented with included the L_2 regularisation 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 regularisation alone and so dropout was used only for the label MLP. We theorise 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 multi-layer perceptron, as the token embeddings will be affected by the tokens outside the context window. That it helped reduce overfitting for labelling, however, suggests that the labeling task was less sparse, and so less dependent 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 (Goldberg and Nivre, 2012). 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 Kiperwasser and Goldberg (2016) there is very little to be gained from training with error exploration when using architectures like ours; Nguyen et al. (2017) don't utilise 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 theorise that error exploration may be of use is when training on languages with smaller datasets; the artificial inflation of the dataset through error exploration could help alleviate the issues caused by having a small dataset.

F Data

For our experiments, we used the Universal Dependencies treebanks. Universal Dependencies provide treebanks for a number of languages in a common format, which was ideal for our goals. The format used by the Universal Dependecies treebanks is CoNLL-U (Universal Dependencies, 2014), which 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. Table 2 outlines an example of this format. 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 tokenisation, 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 state-of-the-art implementations.

The Universal Dependencies treebanks are becoming increasingly commonly used for evaluating parsers; there are regular shared tasks hosted by CoNLL, the most recent of which have used the Universal Dependencies relations, and the Stanford Natural Language Processing Group

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	-	-

Table 2: A CoNLL-U Formatted parse of '*The man hit the ball.*'

has migrated also away from their own Stanford Dependencies treebanks in favour of Universal Dependencies (de Marneffe and Manning, 2016). Some papers, however, still make use of the Stanford Dependencies treebanks (Chen and Manning, 2014; Kiperwasser and Goldberg, 2016). A large issue for us if we had opted to use the Stanford Dependencies is the lack of available languages with a consistent format — a problem that is readily solved by Universal Dependencies.

IV RESULTS

To evaluate our parser, we use 3 common metrics for parser evaluation; labeled attachment score (LAS), unlabeled attachment score (UAS), and label accuracy. UAS assesses the accuracy of the structure of a dependency tree compared to its gold parsing, and LAS also takes in to account the labeling of the dependency arcs (Kubler et al., 2009). Label accuracy assesses the accuracy of the labels assigned to correct arcs.

To evaluate the accuracy of our parser we make use of the CoNLL-X evaluation script[1], 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.

When evaluating the parser, as is typical with natural language tasks we use training, development and testing datasets. The training set is used for training the model, and during training its performance is evaluated on the development set to allow for tuning the hyperparameters of our model. Once the optimal hyperparameters for the development set have been found, the overall performance of the model is evaluated on the testing dataset. This guarantees that the model generalises well to unseen data. The Universal Dependencies datasets provide each of these datasets for each 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 (CoNLL, 2017). Because we are using the same datasets that were used in the shared task, we can directly compare the performance of our parser to the state-of-the-art parsers in the shared task. Details of the datasets we used are outlined in Table 3.

To evaluate our parser's versatility and language independency, we experimented on languages with different grammatical complexities and dataset sizes. Choosing languages with diverse linguistic characteristics ensured that we were not optimising our parser too heavily for one language family, as a large motivation for this paper was to build a parser that performed well on any language. We segment our languages in to 'large' and 'small' treebanks. Our large treebank languages include English, Persian, Turkish and Chinese. Along side these we tested Afrikaans

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

Treebank	Language	Training Sentences	Testing Sentences
en	English	12543	2077
af	Afrikaans	1315	425
kmr	Kurmanji	20	734
fa	Persian	4798	600
tr	Turkish	3685	975
kk	Kazakh	31	1047
zh	Chinese	3997	500

Table 3: An overview of the datasets used for experimentation and testing.

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 languages with very limited resources (31 and 20 sentences, respectively). Each small language was fairly close linguistically to a larger language; Afrikaans to English, Kurmanji to Persian, and Kazakh to Turkish. We attempt to exploit these linguistic similarities when using transfer learning.

	This Work			CoNLL 2017	
Treebank	UAS	LAS	LA	UAS	LAS
en	80.70	65.95	71.16	84.74	82.23
af	66.95	51.44	61.21	-	-
kmr	53.68	35.91	47.42	54.73	47.53
fa	82.49	62.13	67.97	89.64	86.31
tr	54.72	38.05	48.83	69.62	62.79
kk	52.96	29.84	34.99	45.72	29.22
zh	70.10	46.44	55.27	72.39	68.56

Table 4: Unlabeled Attachment Scores, Labeled Attachment Scores, and Label Accuracy versus the best scoring parser in the CoNLL 2017 Shared Task for each language.

As shown in Table 4, our parser shows very promising unlabeled attachment scores across most languages, coming close to the most accurate parser submitted to the leading 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 low-resource 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 Optimisation and Hyperparameter Tuning

During training and hyperparameter tuning, we made use of tensorboard, tensorflow's built in logging and visualisation tool. This allowed us to see how changes in the architecture of the

model affected different characteristics of the overall system, such as training and parsing speed, parsing accuracy, and compare these characteristics directly.

Through hyperparameter tuning with a relatively fine-grained grid search, we found optimal parameters detailed in Table 1. Particularly interesting were the stack and buffer window sizes and experiments to optimise which features to use. The most optimal window sizes were 3 and 2 for the stack and buffer respectively. We theorise that when increasing the size of the windows, the data sparsity increased, leading to a reduced ability for the parser to generalise for larger window sizes. In a similar vein, we found that using language-specific part-of-speech tags yielded poorer accuracies on both the training and development datasets than the universal tags, and we attribute this to the sparsity of the data when using language-specific tags. That the difference between training and development accuracies is larger when using language-specific tags than universal tags (4.05% and 2.65%, respectively) indicates that when using language-specific tags more overfitting is occurring, reducing model's ability to generalise to unseen sentences.

B Long Distance Dependencies

	This Work			Stanford Parser		
Distance	Precision	Recall	F_1 Score	Precision	Recall	F_1 Score
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

Table 5: Precision, Recall, and F_1 Score of head attachments by attachment distance on the English Web Treebank Universal Dependencies dataset.

Comparing our parser on parsing distance with the Stanford parser, we see from Table 5 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 embeddings generated by the BiLSTM.

C Transfer Learning

As shown in Table 6, making use of transfer learning proved very effective across a number of languages. Comparing our results to the CoNLL 2017 Shared Task, we find that our parser gains improvements of 15.24 UAS and 9.06 LAS on Kurmanji, and 15.24 UAS and 7.56 LAS on Kazakh over the current state-of-the-art. We also present one of the first results for Afrikaans, demonstrating promising parsing accuracy.

Treebank	Parser	UAS	LAS
af	This work	66.59	51.44
	This work (p.t. en)	**80.33**	**56.27**
	Best CoNLL 2017	-	-
kmr	This work	53.68	35.91
	This work (p.t. fa)	**60.24**	**56.59**
	Best CoNLL 2017	54.73	47.53
kk	This work	52.96	29.84
	This work (p.t. tr)	52.13	32.26
	This work (p.t. en)	**60.96**	**36.78**
	Best CoNLL 2017	45.72	29.22

Table 6: Unlabeled Attachment Scores and Labeled Attachment Scores on languages with and without pre-training (p.t.) on larger datasets versus the highest accuracy parser in the CoNLL 2017 Shared Task for each langauge.

With lower than expected results when pre-training the Kazakh model on Turkish, 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 pretrained model on a loosely related language can be a better choice for transfer learning than an inaccurate but related language's model.

V EVALUATION

In this paper we have presented an architecture for a natural language parser which performs with state-of-the-art accuracy on a number of languages, and have investigated the qualities of our parser that allow it to do so.

By comparing our solution to the results of the CoNLL 2017 Shared Task, we have a direct comparison with the current state-of-the-art parsers across a variety of languages. The Universal Dependencies datasets used both by us and the CoNLL Shared Tasks also provide the ability to compare our parsers performance between languages on standardised datasets, and to easily investigate the effectiveness of transfer learning.

In our experiments, using python allowed us to iterate quickly between parser architectures, both in the implementation of the parser algorithm and in the model itself. The tensorflow API followed closely with this quality of python, allowing for quick modifications to the neural network model with a relatively straightforward interface. The codebase could have been simplified somewhat by making use of a higher-level deep learning API such as Keras, which runs on top of tensorflow or other deep learning frameworks, however we found that the low-level flexibility granted by tensorflow was useful for implementing what is a slightly unorthodox neural network architecture; that the labeler and arc multilayer perceptrons take subsets of the output of the BiLSTM layer could have been difficult or impossible to implement without using the lower-level aspects of the tensorflow API.

Unfortunately, our parser shows poor performance with regards to labeling. Despite Kiperwasser and Goldberg (2016) finding success with a similar architecture on the Penn Treebank

and Chinese Treebank datasets, our split arc and labeler multi-layer perceptron architecture fell short of expectations and resulted in lower LAS scores than we would have liked.

While we have presented a highly effective parser architecture for many languages, the model optimisation procedure was somewhat limited in that it focused primarily on English datasets. Had we experimented with hyperparameter tuning on a wider range of datasets, we may have achieved a higher accuracy on some languages with different characteristics to English, and may have achieved state-of-the-art results or better. Despite this, our parser performed just below the best performing state-of-the-art parsers on many languages, and in the case of Kazakh exceeded the current state-of-the-art, demonstrating our architecture's multi-lingual capabilities. Particularly interesting 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, and showing high accuracy on Afrikaans, which was not evaluated in the CoNLL shared task.

VI CONCLUSIONS

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 specialised 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 such as Chinese. 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 transfer learning can be used 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 these resource-limited languages.

A Future work

One approach considered for this paper was to attempt to exploit the sequential nature of shift-reduce algorithms, and to utilise an 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 an 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 utilised alongside other parsing methodologies.

With the limitations of our arc labeling multi-layer perceptron holding back our parser somewhat, and while Kiperwasser and Goldberg (2016) 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 MLP_{arc} and 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

Abney, S. P. (1991), Parsing by chunks, *in* 'Principle-Based Parsing: Computation and Psycholinguistics', Kluwer, pp. 257–278.

Baker, J. K. (1979), 'Trainable grammars for speech recognition', *Acoustical Society of America Journal* **65**, S132.

Ballesteros, M., Dyer, C. and Smith, N. A. (2015), Improved transition-based parsing by modeling characters instead of words with LSTMs, *in* 'Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing', pp. 349–359.

Bloomfield, L. (1933), *Language*, Allen & Unwin London.

Booth, T. L. (1969), Probabilistic representation of formal languages, *in* '10th Annual Symposium on Switching and Automata Theory (swat 1969)', pp. 74–81.

Chen, D. and Manning, C. D. (2014), A fast and accurate dependency parser using neural networks, *in* A. Moschitti, B. Pang and W. Daelemans, eds, 'EMNLP', pp. 740–750.

Chomsky, N. (1957), *Syntactic Structures*, Mouton and Co.

Chu, Y. J. and Liu, T. H. (1965), 'On the shortest arborescence of a directed graph', *Science Sinica* **14**.

CoNLL (2017), 'CoNLL 2017 shared task', *http://universaldependencies.org/conll17/*. Accessed: 2018-04-24.

CoNLL (2018), 'CoNLL 2018 shared task', *http://universaldependencies.org/conll18/*. Accessed: 2018-04-24.

Covington, M. A. (2001), A fundamental algorithm for dependency parsing, *in* 'In Proceedings of the 39th Annual ACM Southeast Conference', pp. 95–102.

Daelemans, W., Zavrel, J., van der Sloot, K. and van den Bosch, A. (2003), TiMBL: Tilburg memory based learner, version 5.0, reference guide, Technical Report ILK 03-10, Tilburg University, ILK.

de Marneffe, M.-C. and Manning, C. D. (2016), Stanford typed dependencies manual, Technical report, Stanford University Natural Language Processing Group.
URL: *https://nlp.stanford.edu/software/dependencies_manual.pdf*

Dyer, C., Ballesteros, M., Ling, W., Matthews, A. and Smith, N. A. (2015), Transition-based dependency parsing with stack long short-term memory, *in* 'Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)', pp. 334–343.

Edmonds, J. (1967), 'Optimum Branchings', *Journal of Research of the National Bureau of Standards* **71B**, 233–240.

Goldberg, Y. and Nivre, J. (2012), A dynamic oracle for arc-eager dependency parsing, *in* 'Proceedings of COLING 2012', The COLING 2012 Organizing Committee, pp. 959–976.

Gómez-Rodríguez, C. and Fernández-González, D. (2015), An efficient dynamic oracle for unrestricted non-projective parsing, *in* 'Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)', pp. 256–261.

Goodfellow, I., Bengio, Y. and Courville, A. (2016), *Deep Learning*, The MIT Press.

Jurafsky, D. and Martin, J. H. (2000), *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 1st edn, Prentice Hall PTR.

Kingma, D. P. and Ba, J. (2015), Adam: A method for stochastic optimization, *in* 'Proceedings of the 3rd International Conference for Learning Representations'.

Kiperwasser, E. and Goldberg, Y. (2016), 'Simple and accurate dependency parsing using bidirectional LSTM feature representations', **4**.

Knuth, D. E. (1965), 'On the translation of languages from left to right', *Information and Control* **8**(6), 607–639.

Kubler, S., McDonald, R., Nivre, J. and Hirst, G. (2009), *Dependency Parsing*, Morgan and Claypool Publishers.

Marcus, M. P. (1980), *Theory of Syntactic Recognition for Natural Languages*, MIT Press.

McDonald, R., Pereira, F., Ribarov, K. and Hajic, J. (2005), Non-projective dependency parsing using spanning tree algorithms, *in* 'Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing'.

Nguyen, D. Q., Dras, M. and Johnson, M. (2017), A novel neural network model for joint POS tagging and graph-based dependency parsing, *in* 'Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies', pp. 134–142.

Nivre, J. (2003), An efficient algorithm for projective dependency parsing, *in* 'Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)', pp. 149–160.

Nivre, J. (2008), 'Algorithms for deterministic incremental dependency parsing', *Comput. Linguist.* **34**(4), 513–553.

Nivre, J. (2009), Non-projective dependency parsing in expected linear time, *in* 'Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1', pp. 351–359.

Nivre, J. and Hall, J. (2005), Maltparser: A language-independent system for data-driven dependency parsing, *in* 'Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories', pp. 13–95.

Nivre, J. and Scholz, M. (2004), Deterministic dependency parsing of english text, *in* 'Proceedings of the 20th International Conference on Computational Linguistics', COLING '04.

Pei, W., Ge, T. and Chang, B. (2015), An effective neural network model for graph-based dependency parsing, *in* 'Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)', pp. 313–322.

Tesnière, L. (1959), *Éléments de syntaxe structurale*, Editions Klincksieck.

Universal Dependencies (2014), 'CoNLL-U format', *http://universaldependencies.org/format.html*. Accessed: 2018-04-09.

Wang, W. and Chang, B. (2016), Graph-based dependency parsing with bidirectional LSTM, *in* 'Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)', pp. 2306–2315.

Yamada, H. and Matsumoto, Y. (2003), Statistical dependency analysis with support vector machines, *in* 'In Proceedings of IWPT', pp. 195–206.

Zhao, Y. and Zhou, Q. (2006), A SVM-based model for chinese functional chunk parsing, *in* 'Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing', pp. 94–101.