DEEP BIAFFINE ATTENTION FOR NEURAL DEPENDENCY PARSING

Timothy Dozat
Stanford University
tdozat@stanford.edu

Christopher D. Manning
Sanford University
manning@stanford.edu

ABSTRACT

While deep learning parsing approaches have proven very successful at finding the structure of sentences, most neural dependency parsers use neural networks only for feature extraction, and then use those features in traditional parsing algorithms. In contrast, this paper builds off recent work using general-purpose neural network components, training an attention mechanism over an LSTM to attend to the head of the phrase. We get state-of-the-art results for standard dependency parsing benchmarks, achieving 95.44% UAS and 93.76% LAS on the PTB dataset, 0.8% and 1.0% improvement, respectively, over Andor et al. (2016). In addition to proposing a new parsing architecture using dimensionality reduction and biaffine interactions, we examine simple hyperparameter choices that had a profound influence on the model's performance, such as reducing the value of β_2 in the *Adam* optimization algorithm.

1 INTRODUCTION

Dependency parsers—which annotate sentences in a way designed to be easy for humans and computers alike to understand—have been found to be extremely useful for a sizable number of NLP tasks, especially those involving natural language understanding in some way, such as semantic parsing (Monroe & Wang, 2014) and retrieving images based on a textual description (Socher et al., 2014). However, frequent incorrect parses can severely inhibit final performance, even completely road-blocking lines of research; for this reason, improving the quality of dependency parsers is needed for the improvement and success of these downstream tasks.

In recent years, using deep learning in dependency parsers has gained a lot of attention due to the uncanny ability of neural networks to find real statistical patterns from data. The usual approach involves essentially training a neural network feature extractor and using the neural features in place of handcrafted ones to take discrete actions in a traditional parsing algorithm. Until recently, little to no research had successfully built dependency parsers that only used components that have been used successfully in a wide variety of neural models (although Vinyals et al. (2015) build such a sequence-to-sequence constituency tree parser). There are a few reasons why one might want to build this kind of parser: neural tools have shown a great deal of promise across domains, so it would be wise to see to what extent dependency parsing can benefit from them; a parser that only uses general tools can benefit from innovations in those tools that come from other rapidly moving domains; and advances in the realm of dependency parsing have a higher chance of impacting other fields when they all share the same tools.

In this paper, we build on a recently proposed neural dependency parser that uses only "neural" components—BiLSTMs and attention—drawing on ideas proposed in the neural machine translation literature. Our model substitutes the concatenation-based attention mechanism they use with a variant of the bilinear attention proposed in the neural machine translation literature by Luong et al. (2015), augmenting it with additional MLP layers and making it parallel to traditional classification over a fixed number of classes. In addition to using bilinear transformations to predict dependency arc structures, we also show how to extend bilinear transformations to predict the dependency relations as well.

Furthermore, we explore how different hyperparameter choices—some specific to dependency parsing, others more generally applicable—impacted performance, finding that deviating from estab-

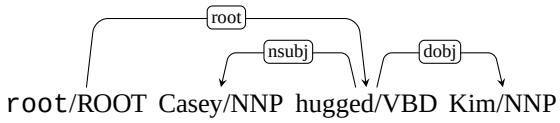

Figure 1: A dependency tree parse for *Casey hugged Kim*, including part-of-speech tags and a special `root` token. Directed edges (or arcs) with labels (or relations) connect the verb to the root and the arguments to the verb head.

lished conventions can have a significant positive effect on the quality of the final model. By using this variation on traditional attention and exploring alternative hyperparameter configurations, we achieve state-of-the-art performance on standard dependency parsing tasks by a considerable margin.

2 Background and Related Work

2.1 Transition-based neural parsing

Transition-based parsers—such as shift-reduce parsers—parse sentences from left to right, maintaining a "buffer" of words that have not yet been parsed and a "stack" of words whose head has not been seen or whose dependents have not all been fully parsed. At each step, transition-based parsers can manipulate the stack and buffer and assign arcs from one word to another. One can then train any multi-class machine learning classifier on features extracted from the stack, buffer, and previous arc actions in order to predict the next action. Here we summarize the contributions made by models that use neural networks to make these transition actions.

Chen & Manning (2014) make the first successful attempt at incorporating deep learning into a dependency parser (henceforth the CM parser). Their approach involves using a feedforward network classifier to make parsing actions; at each step, the network takes as input the concatenation of the word, tag, and label (when applicable) embeddings for words in critical positions (e.g. the top three words on the stack and the leftmost and rightmost dependents of the top two) and puts them through a multilayer perceptron (MLP) that assigns a probability to each action the parser can take. At each step, the parser takes the most probable action, updating the stack and buffer accordingly.

The principal limitation of the CM parser is that it doesn't have access to the entire sentence when making each parsing action, and a number of other approaches have modified or augmented the CM parser to address this. Dyer et al. (2015) and Ballesteros et al. (2016) replace the input to the feedforward network—which in the CM parser is a concatenation of embeddings—with the output of LSTMs over the stack, buffer, and previous actions. Weiss et al. (2015) and Andor et al. (2016) achieve state of the art performance by instead augmenting it with a beam search and a CRF loss so that the model can avoid committing to partial parses that later evidence might reveal to be incorrect.

2.2 Graph-based neural parsing

Transition-based parsing processes a sentence sequentially to build up a parse tree one arc at a time. Consequently, these parsers don't use machine learning for directly predicting edges; they use it for predicting the operations of the transition algorithm. Graph-based parsers, by contrast, use machine learning to assign a weight or probability to each possible edge and then construct a maximum spaning tree (MST) from these weighted edges. Because these general-purpose MST parsing algorithms are deterministic, they don't have access to any information about the sentence other than the weight of each arc, so interactions between words and phrases need to be captured in the process that produces the weights.

Kiperwasser & Goldberg (2016) take a graph-based approach to neural dependency parsing that is in many ways reminsicent of attention in neural machine translation as described by Bahdanau et al. (2014). In the attention model of Bahdanau et al., the recurrent output vector $\mathbf{r}_i^{(target)}$ of the current word i being generated in the target translation is concatenated with the recurrent output vector

$\mathbf{r}_j^{(source)}$ of each word in the source sentence[1], and the result is fed into an MLP with a single linear output node representing the score of target word i aligning to source word j:

$$\mathbf{h}_{ij} = \text{MLP}\left(\mathbf{r}_i^{(target)} \oplus \mathbf{r}_j^{(source)}\right) \tag{1}$$

$$s_{ij} = \mathbf{u}^\top \mathbf{h}_{ij} \tag{2}$$

Kiperwasser & Goldberg (2016) effectively apply this mechanism to dependency parsing, where the word that a given token most strongly attends to is interpreted as its head. They use a bidirectional LSTM to generate a recurrent output vector $\mathbf{r}_i$ for each word i, and for each pair of words i, j, they use the same kind of MLP to compute the score of word i being a dependent on word j:

$$\mathbf{h}_{ij}^{(arc)} = \text{MLP}^{(arc)}(\mathbf{r}_i \oplus \mathbf{r}_j) \tag{3}$$

$$s_{ij}^{(arc)} = \mathbf{u}^\top \mathbf{h}_{ij}^{(arc)} \tag{4}$$

The predicted tree structure is then the tree where each word i depends on the word j with the highest score s_{ij}. They train this model using a hinge loss to maximize the margin between the gold tree and the highest scoring incorrect tree.

Dependency relations are predicted in a similar fashion; the model concatenates $\mathbf{r}_i$ with its gold (or at test time, predicted) head word's recurrent output vector $\mathbf{r}_{y_i}$, and feeds that concatenation into another MLP (5) with an output layer that generates a score for each possible dependency relation (6), which they also train according to a hinge loss. They find that this setup gets excellent results on English and Chinese dependency parsing tasks.

$$\mathbf{h}_{i,y_i}^{(rel)} = \text{MLP}^{(rel)}(\mathbf{r}_i \oplus \mathbf{r}_{y_i}) \tag{5}$$

$$\mathbf{s}_{i,y_i}^{(rel)} = U\mathbf{h}_{i,y_i}^{(rel)} + \mathbf{b}^{(rel)} \tag{6}$$

Cheng et al. (2016) similarly propose a graph-based neural dependency parser, but in a way that attempts to circumvent the limitation of graph-based parsers being unable to condition the scores of each possible arc on previous parsing decisions. In addition to having one bidirectional recurrent network that computes a recurrent hidden vector $\mathbf{r}_i$ for each word, they have additional, unidirectional recurrent networks (left-to-right and right-to-left) that keep track of the probability of previous parsing decisions, and use these together to predict the scores for each arc.

3 Proposed Model

3.1 Deep biaffine parsing

The traditional attention mechanism of Bahdanau et al. (2014) is not the only one that has been proposed in the literature; Luong et al. (2015) argue for substituting the MLP in the attention mechanism with a single bilinear transformation, mapping the target recurrent output vector $\mathbf{r}_i^{(target)}$ and the source recurrent output vector $\mathbf{r}_j^{(source)}$ to a score for the alignment:

$$s_{ij} = \mathbf{r}_i^{\top(target)} U \mathbf{r}_j^{(source)} \tag{7}$$

The straightforward application of this to dependency parsing would be to define the score of a potential dependency arc as a bilinear map between the dependent and the potential head.

$$s_{ij}^{(arc)} = \mathbf{r}_i^\top U \mathbf{r}_j \tag{8}$$

However, there are at least two disadvantages of using the recurrent vectors directly. The first is that they contain much more information than is necessary for calculating the value of s_{ij}—because they're recurrent, they also contain information needed for calculating scores elsewhere in the sequence. Training on the entire vector then means training on superfluous information, which is likely

[1]In this paper we follow the convention of using lowercase italic letters for scalars and indices, lowercase bold letters for vectors, uppercase italic letters for matrices, uppercase bold letters for higher order tensors. We also maintain this notation when indexing; so row i of matrix A would be represented as $\mathbf{a}_i$.

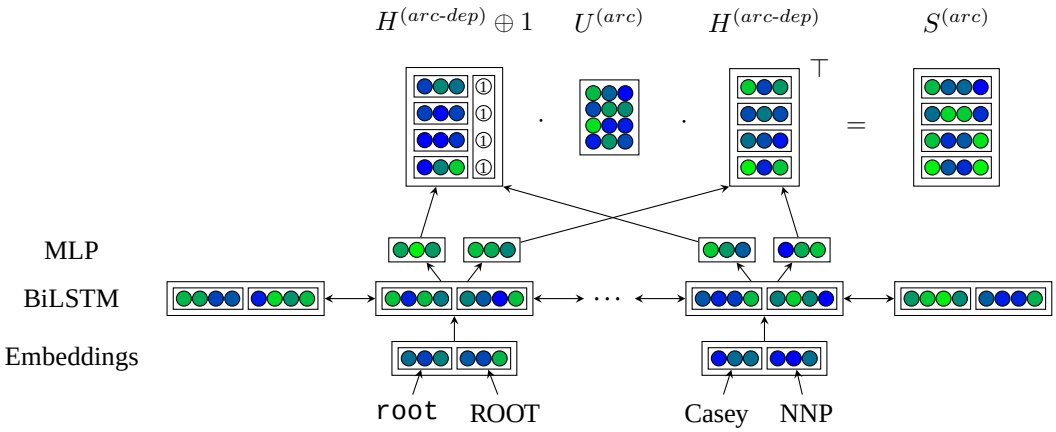

Figure 2: Deep biaffine neural dependency parser applied to the sentence "Casey hugged Kim". We concatenate a vector of ones to $H^{(arc\text{-}dep)}$ to make the scorer biaffine rather than bilinear.

to lead to overfitting. The second disadvantage is that the recurrent vector $\mathbf{r}_i$ consists of the concatenation of the left recurrent state $\overleftarrow{\mathbf{r}}_i$ and the right recurrent state $\overrightarrow{\mathbf{r}}_i$, meaning using $\mathbf{r}_i$ by itself in the bilinear transformation keeps the features learned by the two LSTMs distinct; ideally we'd like the model to learn features composed from both. We can address both of these issues simultaneously by first applying (distinct) MLP functions with a smaller hidden size to the two recurrent states $\mathbf{r}_i$ and $\mathbf{r}_j$ before the bilinear operation. This allows the model to combine the two recurrent states together while also reducing the dimensionality. We call this a *deep* bilinear attention mechanism, as opposed to *shallow* bilinear attention, which uses the recurrent states directly.

$$\mathbf{h}_i^{(arc\text{-}dep)} = \text{MLP}^{(arc\text{-}dep)}(\mathbf{r}_i) \tag{9}$$

$$\mathbf{h}_j^{(arc\text{-}head)} = \text{MLP}^{(arc\text{-}head)}(\mathbf{r}_j) \tag{10}$$

We also make a smaller change to the bilinear attention mechanism. In a traditional classification task, the distribution of classes is often uneven, so the output layer of the model normally includes a bias term designed to capture the prior probability $P(y_i = c)$ of each class, with the rest of the model focusing on learning the likelihood of each class given the data $P(y_i = c|\mathbf{x}_i)$. In dependency parsing, the distribution of dependents is similarly uneven—many words have a global tendency to attract dependents (e.g. verbs, which frequently take many dependents) and others have a global tendency to deter them (e.g. function words, which generally have no dependents). In order to capture the prior probability $P(y_i = j|\mathbf{r}_j)$ of a word taking any dependent (rather than the likelihood $P(y_i = j|\mathbf{r_i}, \mathbf{r_j})$ of a word taking a dependent given what the potential dependent is), we include a bias term linear in $\mathbf{h}_j^{(arc\text{-}head)}$, making it a bi*affine* transformation rather than a bi*linear* one.

$$s_{ij}^{(arc)} = \mathbf{h}_i^{\top(arc\text{-}dep)} U^{(arc)} \mathbf{h}_j^{(arc\text{-}head)} \tag{11}$$

$$+ \mathbf{w}^{\top(arc)} \mathbf{h}_j^{(arc\text{-}head)} \tag{12}$$

As with Kiperwasser & Goldberg (2016), the predicted tree is the one where each word is a dependent of its highest scoring head. This model can be trained with a hinge-loss or a cross-entropy objective; here, we use cross-entropy. Figure 2 shows one configuration of the proposed model.

3.2 Deep biaffine classification

In order to predict the labels, we use a parallel mechanism to deep biaffine attention. We want the label that the model predicts for a given word to be conditioned on that word's head (e.g. we want a word like "fast" be classified as an adverbial modifier when it depends on a verb, but not when it depends on a noun). So again, we use MLPs to transform the recurrent state of the current word $\mathbf{r}_i$ and its gold or predicted head y_i's recurrent state $\mathbf{r}_{y_i}$, but this time we let the model predict an

array of scores—one for each possible label—by allowing the biaffine transformation to map the two vectors to a third vector rather than a scalar. This can then also be trained under a hinge loss or cross-entropy loss objective, and we use the latter.

$$\mathbf{h}_i^{(rel\text{-}dep)} = \text{MLP}^{(rel\text{-}dep)}(\mathbf{r}_i) \tag{13}$$

$$\mathbf{h}_{y_i^{(arc)}}^{(rel\text{-}head)} = \text{MLP}^{(rel\text{-}head)}(\mathbf{r}_{y_i^{(arc)}}) \tag{14}$$

$$\mathbf{s}_i^{(rel)} = \mathbf{h}_i^{(rel\text{-}dep)} \mathbf{U}^{(rel)} \mathbf{h}_{y_i^{(arc)}}^{(rel\text{-}head)} \tag{15}$$

$$+ W^{(rel)} \left(\mathbf{h}_i^{(rel\text{-}dep)} \oplus \mathbf{h}_{y_i^{(arc)}}^{(rel\text{-}head)} \right) \tag{16}$$

$$+ \mathbf{b}^{(rel)} \tag{17}$$

Again, in our model, the MLPs serve to reduce dimensionality and build features from the two halves of the last BiLSTM output state. In this part of the model, the term in line (16) captures global preferences for the kinds of labels word i can take, as well as global preferences for the kinds of dependents word i's head can take (e.g. articles will have a strong preference to take a determiner label and a noun will have a strongly prefer dependents to take determiner or adjectival labels).

3.3 Practical considerations

A noteworthy advantage of bilinear or biaffine attention over traditional attention is that it requires less memory to compute. Traditional attention requires explicitly computing a size d hidden state for each ordered pair of words in the length n sentence; the resulting tensor $\mathbf{H}$ is $(n \times n \times d)$-dimensional. Bilinear attention, however, can be computed more memory-efficiently; because it uses matrix multiplications, it never explicitly computes a full $(n \times n \times d)$ hidden state. As a result, computing traditional attention requires $O(dn^2)$ memory, whereas computing bilinear attention requires only $O(dn + n^2)$ memory. Similarly, the memory complexity of our approach to label classification is $O(dnc + n^2c)$—with c being the number of classes to predict—whereas the complexity of the traditional attention approach is $O(dn^2 + cn)$. As long as the number of labels c is significantly smaller than the length of the longest sequence in the dataset, this bilinear classification mechanism will also be more memory-efficient than any variant that concatenates each pair of recurrent output vectors.

Since BiLSTMs use $O(dn)$ memory, this choice can noticeably affect the memory requirements of the network. While our TensorFlow (Abadi et al., 2015) implementation[2] of a deep biaffine parser with multiple hidden MLP layers can train with less than 2.5GB of GPU memory, our implementation of an otherwise identical parser that uses a traditional attention mechanism requires more than 4GB, even with only one hidden MLP layer. Since the attention mechanism has far fewer parameters than the BiLSTMs, the concatenation-based approach winds up using upwards of 33% of its consumed memory on training a part of the network that comprises less than 1% of the parameters, which is clearly not ideal.

4 Experiments and Results

4.1 Hyperparameter selection

In this section we go in depth into how the different hyperparameters affect parsing performance. Here we use the Penn Treebank train and validation splits, converted to the Stanford Dependencies using version 3.5.0 of the Stanford dependency converter, reporting unlabeled attachment score (UAS) and labeled attachment score (LAS). When not otherwise specified, our model uses: 100-dimensional word and tag embeddings with word vectors initialized to GloVe (Pennington et al., 2014) trained on Wikipedia and Gigaword and an 15% chance of dropping tag embeddings; 4-layer BiLSTMs with 300-dimensional left and right LSTMs, using the form of recurrent dropout suggested by Gal & Ghahramani (2015) with a 75% keep probability between timesteps and a 67% keep probability between layers; a 1-layer 100 dimensional MLP layer with the elu function (Clevert et al., 2015), also with a 67% keep probability; the Adam optimizer (Kingma & Ba,

[2]Using a different framework or implementation could yield different results

		MLP depth					
		0		1		2	
		UAS	LAS	UAS	LAS	UAS	LAS
BiLSTM depth	2	95.03*	93.01*	94.98*	93.14*	94.84*	92.95*
	4	95.08*	93.05*	**95.24**	**93.37**	95.19	93.24

Table 1: Effect of network depth on performance. Statistically signficant differences between the highest-scoring model are marked with an asterisk.

2014) with $\beta_1 = \beta_2 = .9$ and learning rate $2e^{-3}$, annealed continuously at a rate of .75 every 2,500 iterations; 120 epochs of training, with batches of approximately 5,000 tokens. Words that only occur once in the training set are replaced with a special `<UNK>` token. These hyperparameters were selected based on a random search followed by further refining using both grid search and trial-and-error. The performance of the different hyperparameter configurations is shown in Table 1 and Table 2.

4.1.1 DEPTH

First we examine how making the network deeper affects performance, since all other models discussed hereuse two-layer neural networks (except Cheng et al. (2016), who use one-layer networks). We test using two or four BiLSTM layers, and zero, one, or two MLP layers after the last BiLSTM. What we find is that making the network deeper improves performance to an extent—when the BiLSTM is shallow, adding an MLP doesn't significantly improve performance, presumably because the dimensionality reduction limits its representative power too much—but when the BiLSTM is deeper and can learn more abstract features, dimensionality reduction successfully helps the model avoid overfitting. Using a deeper MLP, however, actually hinders performance. The training accuracies of the two- and four-LSTM models with two-layer MLPs are comparable, suggesting that the deeper network isn't overfitting. Instead, it seems more likely that the biggest gain from the MLP layer is dimensionality reduction rather than adding significant further nonlinear abstraction, so the second layer serves only to needlessly distort the information learned by the LSTM.

It should be noted, however, that the increase in accuracy does come with a cost–while the two-layer LSTM can parse about 1000 sentences per second on an nVidia Titan X GPU machine, the four-layer one can only parse about 500 sentences per second. Including dimensionality reduction, however, speeds up parsing by about 50 sentences per second.

4.1.2 ATTENTION MECHANISM

Next, we compare three attention-based scoring mechanisms—since our model uses a very different hyperparameter configuration from Kiperwasser & Goldberg's, we implemented the concatenation-based attention mechanism in addition to our deep biaffine one. However, the biaffine layer of our parser has $O(d^2)$ parameters and the last layer of the concatenation-based one has only $O(d)$. In order to ensure that the extra parameters in our model don't influence the outcome, we also train a special case of our model with a $O(d)$-parameter bilinear layer where $U^{(arc)}$ and each slice $U^{(rel)}_{:,i,:}$ is diagonal; for efficiency we also abstain from including the linear bias terms. What we find is that both the full biaffine parser and the smaller diagonal bilinear parser significantly outperform the concatenation-based one.

4.1.3 RECURRENT CELL

We also tested to see how the choice of LSTM or GRU affects performance. However, we found that GRUs were unable to train with recurrent dropout, with the loss exploding after a few iterations of training even under a lower learning rate. Greff et al. (2015) suggest modifying the formulation of LSTMs to make them more GRU-like by using a coupled input-forget gate (CifLSTM), but retaining the output gate of the vanilla LSTM. We thus modified the formulation slightly to remove one of the tanh nonlinearities—which is not needed when using an update gate $\mathbf{z}_t$ (18 - 19)—and trained

Attention	UAS	LAS	Recurrent cell	UAS	LAS
Biaffine	**95.24**	**93.37**	LSTM	**95.24**	**93.37**
Diag Bilinear	95.14	93.24	Cif-LSTM	95.20	93.31
Concat	94.96*	93.03*	GRU	-	-
Adam	UAS	LAS	Tag dropout	UAS	LAS
$\beta_2 = .9$	**95.24**	**93.37**	85% keep	**95.24**	**93.37**
$\beta_2 = .999$	94.93*	93.05*	100% keep	94.99*	93.08*
GD	89.31*	86.59*	0% keep	94.90*	92.97*

Table 2: Effect of select hyperparameters on performance. Statistically signficant differences from the highest-scoring model are marked with an asterisk.

parsers using them.

$$\mathbf{c}_t = \mathbf{i}_t \odot \tanh(\mathbf{a}_t) + (1 - \mathbf{f}_t) \odot \mathbf{c}_{t-1} \quad \text{Vanilla LSTM cell} \quad (18)$$

$$\mathbf{c}_t = \mathbf{z}_t \odot \mathbf{a}_t + (1 - \mathbf{z}_t) \odot \mathbf{c}_{t-1} \quad \text{Cif-LSTM cell} \quad (19)$$

Critically, CifLSTM cells were able to train with dropout in spite of sharing one simplification to LSTMs that GRUs have; the reason why GRUs failed to train almost certainly has to do with scaling up the hidden state at training time. When using the dropped hidden state to compute the activations for the gates and next hidden state, the previous hidden state needs to be scaled up by the inverse of the keep probability in order to ensure that the expected activations at training time are the same as the actual activations at test time (Srivastava et al., 2014). However, because the GRU cell always reveals its hidden state and uses the update gate to retain the value of the hidden state across steps, scaling up the hidden state at every step in the sequence increases the magnitude of the the activations exponentially. Because of LSTMs' distinction between the cell state and the hidden state, they don't suffer from this problem to nearly the same extent.

One recently proposed alternative to dropout, zoneout (Krueger et al., 2016), would address this issue with using dropout in GRUs–however, we leave experimenting with this for future work.

4.1.4 Optimization algorithm

We choose to optimize with Adam (Kingma & Ba, 2014), which keeps a moving average of the L_2 norm of the gradient for each parameter throughout training and divides the gradient for each parameter by this moving average, ensuring that the magnitude of the gradients will on average be close to one. However, we find that the value for β_2 recommended by Kingma & Ba—which controls the decay rate for this moving average—is too high. When this value is very large, the magnitude of the current update is heavily influenced by the larger magnitude of gradients very far in the past, with the effect that the optimizer can't adapt quickly to recent changes in the model. Thus we find that setting β_2 to $.9$ instead of $.999$ makes a large, positive impact on final performance.

4.1.5 Tag dropout

All models under consideration here—including our own—use POS tags as input. While we find training on POS tags to be very helpful for final performance, we want to ensure that our model doesn't overfit to specific sequences of POS tags and we want to ensure that it remains robust to upstream tagging errors. So to keep our model from depending too heavily on POS tags, we randomly drop them 15% of the time, finding this to noticeably improve performance.

4.2 Model comparison

4.2.1 Datasets

In this work we show test results for the proposed model on three datasets, coming from two sources: the English Penn Treebank, automatically converted from constituency trees into Stanford Dependencies using both version 3.3.0 and version 3.5.0 of the Stanford Dependency converter (PTB-SD 3.3.0 and PTB-SD 3.5.0); and the Chinese Penn Treebank version 5.1 (CTB 5.1), automatically converted from constituency trees into the CoNLL 2007 dependency format with Penn2Malt. PTB-SD

	English PTB-SD 3.3.0		Chinese PTB 5.1	
Model	UAS	LAS	UAS	LAS
Dyer et al. (2015)	93.1	90.9	87.1	85.5
Ballesteros et al. (2016)	93.56	91.42	87.65	**86.21**
Kiperwasser & Goldberg (2016)	93.9	91.9	87.6	86.1
Cheng et al. (2016)	94.10	91.49	88.1	85.7
Weiss et al. (2015)	94.26	92.41	-	-
Andor et al. (2016)	94.61	92.79	-	-
Deep Biaffine	**95.44**	**93.76**	**90.07**	85.98
Deep Biaffine, SD 3.5.0	**95.66**	**94.03**		

Table 3: Results on the standard English PTB and Chinese PTB parsing datasets

3.3.0 and CTB 5.1 are datasets standardly used in other dependency parsing work over the past four years, and PTB-SD 3.5.0 is an updated version of PTB-SD 3.3.0. As is standard, we omit puncuation from evaluation and use predicted POS tags for the English PTB dataset—generated from the Stanford POS tagger (Toutanova et al., 2003)—and gold POS tags for the Chinese PTB dataset. Note that in the previous section we reported validation scores on PTB-SD 3.5.0, but in this section we report our test scores on PTB-SD 3.3.0 and compare those to those of other approaches in the literature, in order to keep the hyperparameter choices fairly independent of the test set. Word embeddings for Chinese were generated using Word2Vec (Mikolov et al., 2013) on Chinese Wikipedia.

4.2.2 RESULTS

Here, we compare our model to a number of others in the literature: the models of Dyer et al. (2015) and Ballesteros et al. (2016), which use LSTMs to generate features for a transition-based parser; the models of Weiss et al. (2015) and Andor et al. (2016), which augment the CM parser with a beam search and a globally normalized CRF objective function; and Kiperwasser & Goldberg (2016) and Cheng et al. (2016), which like this work uses BiLSTMs to generate feature embeddings used for an attention-based parser.

What we see is that, with the exception of LAS on CTB 5.1, our implementation achieves state-of-the-art results by a fairly substantial margin. It's also worth pointing out that our parser also performs better on PTB-SD 3.5.0 and 3.3.0. While it's possible that this is in part because we tuned on 3.5.0, we think it probably represents improvements in the dependency representation and converter quality between version 3.3.0 and 3.5.0.

On the CTB labeled attachment score, our model underperforms state-of-the-art in spite of getting state-of-the-art unlabeled attachment results. We have two possible explanations for this: the first is that our pretrained embeddings were inferior to those used by the other researchers, and don't capture label information as well; the second, which we feel is more likely, is that the POS tag dropout prevented the model from heavily relying on tags to make label predictions. Since ours and all other models considered train and evaluate on gold tags, this regularization likely hurts the model's ability to learn the high correlation between gold tags and labels. Since in practice models won't have access to gold tags, we don't see this as a point of major concern, but future research will need to substantiate this hypothesis.

5 CONCLUSION

In this paper we proposed using a modified version of bilinear attention in a neural network dependency parser, and showed that our approach outperformed state-of-the-art by a fairly large margin. We also discussed in detail some of the hyperparameter choices that we found to make a critical difference in end performance: we find that deeper networks of four LSTM layers outperform shallower networks of two LSTM layers when using our deep biaffine attention mechanism; we find GRU cells to have significant difficulty training with dropout, but LSTMs (vanilla or with a coupled input-forget gate) have no trouble; we argue that the default settings for the Adam optimizer should be tweaked; and we demonstrated tag dropout to be effective. Future work will explore the perfor-

mance of this parser on a wider variety of languages and, especially for morphology-rich languages, augment it with a smarter way of handling out of vocabulary tokens.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Association for Computational Linguistics*, 2016. URL `https://arxiv.org/abs/1603.06042`.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*, 2014.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A Smith. Training with exploration improves a greedy stack-LSTM parser. *Proceedings of the conference on empirical methods in natural language processing*, 2016.

Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the conference on empirical methods in natural language processing*, pp. 740–750, 2014.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. Bi-directional attention with agreement for dependency parsing. *arXiv preprint arXiv:1608.02076*, 2016.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representations*, 2015.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. Transition-based dependency parsing with stack long short-term memory. *Proceedings of the conference on empirical methods in natural language processing*, 2015.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *International Conference on Machine Learning*, 2015.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *Empirical Methods in Natural Language Processing*, 2015.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *International Conference on Learning Representations*, 2013.

Will Monroe and Yushi Wang. Dependency parsing features for semantic parsing. 2014.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12, 2014.

Richard Socher, Andrej Karpathy, Quoc V Le, Christopher D Manning, and Andrew Y Ng. Grounded compositional semantics for finding and describing images with sentences. *Transactions of the Association for Computational Linguistics*, 2:207–218, 2014.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pp. 173–180. Association for Computational Linguistics, 2003.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pp. 2773–2781, 2015.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. *Annual Meeting of the Association for Computational Linguistics*, 2015.