Multilingual Graph-Based Neural Dependency Parsing Using BiLSTMs, Structured Attention & Conditional Random Fields

Roman Pearah
CLMS, University of Washington
rpearah@uw.edu

Abstract

In this paper, I describe a neural network architecture for multilingual syntactic dependency parsing in response to the CoNLL 2017 Shared Task and drawing from the work of Kiperwasser and Goldberg (2016), Dozat and Manning (2016), and Ma and Hovy (2017). The architecture breaks the parsing problem into an encoder for transforming annotated text into a feature representation using bidirectional LSTMs, a biaffine structured attention mechanism for scoring potential labeled arcs, and a graph-based CRF for determining loss. I evaluate the model, along shared task criteria, on 3 different diverse treebanks.

1 Introduction

Dependency parsing is an important component of a variety of NLP systems, including machine translation systems, word disambiguators, and low-resource language processors. As such, a parser is often not an end in itself and usually precedes the final stages of these larger systems. Therefore, it is important to design and build them to be very accurate so as not to propagate errors to downstream components.

In designing towards this goal, parsers fall into two board categories: *transition-based* and *graph-based*. The former works by parameterizing a model over the transitions of an abstract state machine that can be used to derive dependency graphs, choosing the best parsing action for the current parsing state. The latter works by parameterizing a model over the entire space of valid graphs, choosing the best graph as a complete structure. The primary advantage of graph-based parsers is their ability to infer from global information and their adaptability to non-projective parse trees.

Neural dependency parsers, taking advantage of recent advancements in neural networks and deep learning architectures, have become the state-of-the-art (Andor et al., 2016; Dozat and Manning, 2016), combining the tasks of automatic feature selection, long-distance contextualization, and structured learning into models that can be trained end-to-end as differentiable computations.

In this paper, I describe a graph-based neural dependency parser combining features from many of the best models to date (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2016; Ma and Hovy, 2017), including distributed word representations, bidirectional long short-term memories (BiLSTMs), structured attention, and conditional random fields (CRFs).

To give context to the motivations for these choices and others, I will describe how such a system is evaluated (Section 4), how parameters can be trained (Section 5.1), and a network architecture appropriate to the various demands of the task (Sections 5.2-5.6). I will also discuss the CoNLL 2017 shared task that will serve as the source of data and the crucible for the model's performance (Sections 2-3).

2 CoNLL 2017 Shared Task

On December 11, 2016, SIGNLL (ACL's Special Interest Group on Natural Language Learning) announced a shared task for its 2017 Conference on Computational Natural Language Learning (CoNLL). The focus of the task is "learning syntactic dependency parsers that can work in a real-world setting, starting from raw text, and that can work over many typologically different languages, even surprise languages for which there is little or no training data, by exploiting a common

syntactic annotation standard."[1]

Beginning with raw text, participating systems must "find labeled syntactic dependencies between words, i.e. a syntactic *head* for each word, and a *label* classifying the type of the dependency relation." This is a well-defined task and comes with an official collection of data that all systems must use for development and training. As such, the parser described in this paper was developed as if it were a participating system for this task and using the task's data.

While all participating systems are technically expected to take raw text as input, they have the option to use data preprocessed by UDPipe[2]. In light of that, this paper focuses only on the dependency analysis phase and the model presented takes as input the UDPipe annotated data. Other components or phases that might be part of an end-to-end system are outside the scope of this work.

Furthermore, the shared task does not dictate how systems (and by extension, their dependency parsers) should approach the multilingual aspect of the problem. Systems are free to train one or more models for each language, a universal model, or anything in between. They are also free to decide which models will run for any given input.

I have chosen to take the model-per-language approach here, leaving experimentation with universal parsing for future work. I will also not cover approaches for dealing with surprise languages using low-information modeling techniques but will let the results stand as a pointer to the overall generalizability of the architecture.

3 Data

The CoNLL 2017 shared task participants are provided training data taken from the Universal Dependencies release 2.0[3], which is made up of 64 treebanks in 45 languages, each divided into training and development sets. At the time of writing, the test phase had completed and the test data was made available.

Data is given in both raw text and CoNLL-U format[4], preprocessed by UDPipe. Additionally, word2vec (Mikolov et al., 2013) embeddings in 100 dimensions are available[5] for all languages. Combined, the data contains 11,814,230 tokens, 12,102,983 words, and 630,518 sentences from a variety of domains.

No additional preprocessing or debiasing will be done on the provided data before feeding it into the model, with the exception of lowercasing forms (optionally).

4 Evaluation and Usage

4.1 Quality Metric

Dependency parsers are usually evaluated and compared using either an *unlabeled* attachment score (UAS), a *labeled* attachment score (LAS), or both, since they provide a meaningful and easy-to-digest 0-100 (100 being best) accuracy metric for directly comparing parsers on a given task. Because the CoNLL 2017 shared task sets the LAS as the sole quality metric, that is the primary metric I will use.

The LAS is defined as the percentage of tokens (including punctuation) with both the correct head and the correct dependency relation label (as opposed to the UAS, which only measures the percentage of correct heads, regardless of the label).

Because a participating system might be doing its own tokenization, the LAS score for the purposes of the CoNLL task is:

> modified to take word segmentation mismatches into account. A dependency is therefore scored as correct only if both nodes of the relation match existing gold-standard nodes. Precision P is the number of correct relations divided by the number of system-produced nodes; recall R is the number of correct relations divided by the number of gold-standard nodes. We then define LAS as the F1 score:

$$F_1 = \frac{2PR}{P+R}$$

However, because I used the UDPipe data, which sets the gold standard for word segmentation, precision and recall will be identical and the metric reduces to the standard LAS.

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

[2] https://ufal.mff.cuni.cz/udpipe

[3] https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1983

[4] http://universaldependencies.org/docs/format.html

[5] https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1989

4.2 Additional Constraints

In addition to the evaluation metric, it is also be important to constrain and report on the overall size of the model and speed of convergence so that it can be judged for practicality in production environments. Throughout the description of the network architecture, these additional constraints will manifest as choices to reduce the number of hyperparameters and parameters where possible and to apply best practices to control the behavior of the optimization procedure.

5 Network Architecture

5.1 Loss Function

Looking ahead, the output of the model (described below) is a set of scores for all labeled arc tuples (h, m, l). If these scores are interpreted as *energy outputs*, then according to Do and Artieres (2010), the model can be viewed as a log-linear conditional random field (CRF). Taking the form from Koo et al. (2007):

$$P(\boldsymbol{y}|\boldsymbol{x};\Theta) = \frac{exp\Big(\sum\limits_{(x_h,x_m,l)\in\boldsymbol{y}} \phi(x_h, x_m, l;\Theta)\Big)}{Z(\boldsymbol{x};\Theta)} \tag{1}$$

where Θ is the set of parameters in the model, $\phi(x_h, x_m, l;\Theta)$ is the energy for a labeled arc, and $Z(\boldsymbol{x};\Theta)$ is the normalizing potential function.

This straightforwardly suggests a **negative log likelihood** (NLL) loss function, where

$$\begin{aligned} -logP(\boldsymbol{y}|\boldsymbol{x};\Theta) = & -\sum_{(x_h,x_m,l)\in\boldsymbol{y}} \phi(x_h, x_m, l;\Theta) \\ & - logZ(\boldsymbol{x};\Theta) \end{aligned} \tag{2}$$

Therefore, calculating the loss depends on knowing the partition function $Z(\boldsymbol{x};\Theta)$. Unfortunately, this partition function implies summing over all single-root, non-projective trees for a sentence.

Koo et al. (2007) offer a solution to this problem by making use of Kirchhoffs Matrix-Tree Theorem to compute the partition function from an adjacency matrix of potentials in $O(n^3+|L|n^2)$ time for a sentence of length n and L as the set of labels. Ma and Hovy (2017) make use of the the log-linear technique for their neural dependency parser as a novel extension of Dozat and Manning (2016)'s architecture.

Because NLL encourages a close fit to the target, here defined by a set of labeled arcs, it should be a good proxy for the LAS evalutation metric. A max-margin based loss function would possibly be superior, but solutions to such a function, in light of the challenges above, are non-differentiable. Using NLL allows the updating of parameters through backpropagation.

5.2 Token Embeddings

The proposed parser works with a distributed representation of each token in a sentence, embedding its word form, lemma, and part-of-speech (POS) tag. It uses the additive token embedding approach suggested by Botha and Blunsom (2014):

$$\begin{aligned} \boldsymbol{x}_i = & E^{form}\boldsymbol{e}_i^{form} + E^{pos}\boldsymbol{e}_i^{pos} \\ & + E^{lemma}\boldsymbol{e}_i^{lemma} + \dots \end{aligned} \tag{3}$$

where $\boldsymbol{e}_i^{feat} \in \{0,1\}^{|V^{feat}|}$ is a one-hot vector for a feature of the ith token $\in V^{feat}$, $E^{feat} \in \mathbb{R}^{e\times|V^{feat}|}$ is a learned parameter, and $\boldsymbol{x}_i \in \mathbb{R}^d$ is the final continuous embedding. This method is preferred to one that concatenates different feature embeddings because it reduces the number of hyperparameters and learned parameters in the model, allows for easily dropping features that may not be available, and reduces the memory overhead.

The vocabulary for all features is increased by the addition of an `UNKNOWN` term, a `ROOT` term (and in the case of minibatching, a `PAD` term). Optionally, the parameter values for E^{form} can be initialized with pre-trained *word2vec* embeddings $E^{w2v} \in \mathbb{R}^{100\times|V^{form}|}$ provided to participants in the shared task. To accommodate these datasets, the hyperparameter corresponding to the first dimension of all E^{feat} parameters is fixed at $e = 100$.

Including the root token embedding $\boldsymbol{x}_0$, a fully embedded sentence of length n has the representation

$$S = \langle \boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n \rangle \tag{4}$$

5.3 BiLSTM

Following Kiperwasser and Goldberg (2016), Dozat and Manning (2016), and Ma and Hovy (2017), the model will take the embedded sentence representation as the input to a stack[6] of bidirectional long short-term memories (BiLSTMs) with h hidden states to produce a recurrent representation $\boldsymbol{r}_i \in \mathbb{R}^{2h}$ of each token. LSTMs are designed

[6] Dozat and Manning (2016) found that multiple layers outperform one in the context of dependency parsing.

to be effective at modeling the long-distance context that is essential to tasks like this one without the vanishing/exploding gradient problems faced by vanilla RNNs. A *bidirectional* variation of LSTM additionally benefits the model by taking into account both past and future context. This intuitively corresponds to the nature of arcs in the dependency parse of a sentence, where the head might be found on either side of a term.

In order to reduce dimensionality and avoid overfitting, the output of the topmost BiLSTM layer will be passed through an MLP hidden layer:

$$\boldsymbol{h}_i = MLP(\boldsymbol{r}_i, \Theta, \varphi) = \varphi(W\boldsymbol{r}_i + \boldsymbol{b}) \quad (5)$$

where $\Theta = \{W \in \mathbb{R}^{d \times 2h}, \boldsymbol{b} \in \mathbb{R}^d\}$ are learned parameters and φ is the non-linear activation function.

As an enhancement to previous work, the model described here has the capability of applying two independent MLP layers for head and modified terms respectively.

5.4 Structured Attention Mechanism

The BiLSTM-encoded representations for tokens is used to train a scoring mechanism using a *bi*affine transformation. This involves training a set of parameters $\Theta = \{\boldsymbol{U} \in \mathbb{R}^{d \times d}, \boldsymbol{w}_{h,l} \in \mathbb{R}^d, \boldsymbol{w}_{m,l} \in \mathbb{R}^d, b_l\}$ for all $l \in L$, where L is the set of potential arc labels. The biaffine scoring function is taken from (Ma and Hovy, 2017) as:

$$\begin{aligned}\phi(x_h, x_m, l; \Theta) = \boldsymbol{h}_h^T U_l \boldsymbol{h}_m \\ + \boldsymbol{w}_{h,l}^T \boldsymbol{h}_h \\ + \boldsymbol{w}_{m,l}^T \boldsymbol{h}_m + b_l\end{aligned} \quad (6)$$

where $\boldsymbol{h}_h$ and $\boldsymbol{h}_m$ are the representations of a potential head and modifier respectively.

As a matter of implementation detail, the parameters are represented by tensors $\boldsymbol{\Theta} = \{\mathbf{U} \in \mathbb{R}^{d \times d \times |L|}, \mathbf{W}_h \in \mathbb{R}^{d \times |L|}, \mathbf{W}_m \in \mathbb{R}^{d \times |L|}, \mathbf{b}_l \in \mathbb{R}^{1 \times |L|}\}$.

The can be understood as a type of structured attention mechanism (see Luong et al. (2015)), where one "queries" the memory of target (head) representations with source (modifier) representations according to an attention distribution. It can also be viewed as a variable-class classifier where the number of classes (i.e. the tokens in the sentence) is not known *a priori*. In the form presented in (4), this has the conceptual advantage of allowing for the interpretation of the first three summed components as $P(h|m)$, prior $P(h)$, and prior $P(m)$ respectively.

At this point, the model will have produced the scores necessary to drive the CRF and calculate loss as described previously, all using an end-to-end differentiable computational graph that can be optimized through backpropagation.

5.5 Dropouts

To discourage overfitting and improve convergence time, dropouts are applied to the token embedding, BiLSTM, and MLP layers.

5.6 Maximum Spanning Tree Decoding

Because of the equivalence between finding the highest weighted dependency parse tree and finding the maximum spanning tree (MST) of a directed graph (Mcdonald et al., 2005), the predicted parsing tree can be decoded at test time using a non-projective MST algorithm on the scores resulting from the model. In particular, the Chu-Liu-Edmonds algorithm is known to solve this problem in $\mathcal{O}(n^2)$ time. This process is not necessary during training.

6 Training Configuration

6.1 Pre-training

The only pre-trained input to the model is the the aforementioned word2vec embeddings for the token forms.

6.2 Optimization

Parameters are optimized using an Adam optimizer with an annealed learning rate across epochs. The annealing is done after each epoch by setting the optimizer's learning rate to:

$$LR_t = \alpha^{\frac{t}{T}} LR_0 \quad (7)$$

where α is the decay rate, $t \in T$ is the epoch number, and LR_0 is the initial learning rate. Gradient clipping is also used before each backward pass to avoid exploding gradients (Pascanu et al., 2012).

6.3 Batch Sizes

Mini-batching reduces the variance of gradient updates and the number of complex tensor operations that must be performed for each epoch, but they also introduce a complexity overhead to the implementation (e.g. padding for different sentence lengths within the batch). Kiperwasser and Goldberg (2016) use single instances while Dozat

Table 1: Hyperparameter Space

Layer	Parameter	Prior
Embedding	Dimensions	100
	Dropout	$U(0\%, 50\%)$
BiLSTM	Depth	$U(1, 3)$
	Hidden Size	$logU(128, 768)$
	Dropout	$U(0\%, 50\%)$
MLP	Depth	1
	Size	$logU(128, 512)$
	Separate	true or false
	Dropout	$U(0\%, 50\%)$
Attention	Nonlinear	true or false
Optimizer	Type	Adam
	Batch Size	$U(1, 10)$
	Initial LR	$U(0.001, 0.002)$
	LR Decay rate	$U(0.25, 1.0)$
	β_1, β_2	0.9, $logU(0.9, 0.99)$
	Clip	$logN(5.0, 5.0)$

Table 2: Final Hyperparameters

Layer	Parameter	Prior
Embedding	Dimensions	100
	Dropout	5%
BiLSTM	Depth	2
	Hidden Size	128
	Dropout	5%
MLP	Depth	1
	Size	256
	Separate	false
	Dropout	5%
Attention	Nonlinear	false
Optimizer	Type	Adam
	Batch Size	10
	Initial LR	0.0016
	LR Decay rate	0.45
	β_1, β_2	0.9, 0.92
	Clip	44

and Manning (2016) and Ma and Hovy (2017) use mini-batching, all with good results. The batch size, therefore, is a hyperparameter for this model.

6.4 Hyperparameters

Table 1 shows the main hyperparameters of the model and the prior distributions for experimental values. U is a uniform distribution, $logU$ is log uniform, and $logN$ is log normal.

Rather than doing manual hyperparameter optimization, the Hyperband method (Li et al., 2016) was used with 27 maximum iterations. For this process, I used (for mostly arbitrary but practical reasons) the English training and development datasets. In future work, I would like to use separate Hyperband runs for each treebank and run each optimization with a higher maximum iteration (e.g. the recommended 81). Alternatively, the language itself could be sampled during optimization. Table 2 shows the best parameters from this method (measured by loss) and these were used to generate results for all treebanks.

7 Implementation

The model was implemented in Python using PyTorch[7]. At the time of writing, PyTorch is one of a relatively new breed of "dynamic" neural network libraries that are easier to debug and reason about, as well as facilitating tasks that would be difficult or impossible to implement in "static" or "compiled" systems (e.g. Theano, Tensorflow, or, by extension, higher-level libraries such as Keras). A significant amount of the development time involved the creation of the custom CRF loss function. The source code for the final implementation, including instructions for recreating the results found below, is available on GitHub[8].

7.1 Innovations Experimentation

The model described here draws heavily upon the work of Kiperwasser and Goldberg (2016), Dozat and Manning (2016), and Ma and Hovy (2017). It does, however, combine attractive features from each into a hybrid model. First, it makes use of additive embedding, which was not used in any of these previous efforts, reducing memory requirements throughout the downstream model. Next, the model uniquely incorporates the token lemma for additional information. Furthermore, additional architectural choices concerning the separation of the encoding MLPs and a non-linearity applied to the attention weights were considered in the hyperparameter optimization. Lastly, this optimization process utilized a state-of-the-art exploration method to verify or condemn assumptions made in these previous works.

For example, these works all assumed the need for a fairly high dropout rate, but I discovered through the search process that a very conservative

[7] http://pytorch.org

[8] https://github.com/neverfox/ling575-sp17-project

5% performed best. Also, the MLP was justified as part of the architecture to reduce dimensionality and throw out irrelevant information from the BiLSTM encoding. The hyperparameter search concluded that *not* reducing dimensionality was a better option, cf. both Ma and Hovy (2017) and Dozat and Manning (2016). Additionally, the best parameters did not involve large hidden layers in the BiLSTM or more than two such layers, cf. Dozat and Manning (2016). Therefore, it appears initially as if the same intuition for not "over-encoding" can be better achieved by simply not exploding the feature space to begin with, which additionally benefits runtime and memory needs.

7.2 Baseline

There are two baseline systems for the CoNLL 2017 Shared Task: UDPipe (Straka et al., 2016) and SyntaxNet (Andor et al., 2016). Their results are included below, along with the results of the top participating system in the shared task for a given language (referred to as "Top PS"), for comparison to this paper's model. All of the participating results appear on the shared task website[9]. Note that this model was *not* a participating system, due to the timing of the task.

8 Results

Because of time limitations, not all 65 treebanks were trained on and tested at the time of writing. Instead, three languages were chosen: English, Chinese, and Vietnamese. THe first two are common dependency parsing targets and the last was the language that the best participating system performed the worst on among among those hat have development data available. It is also one of the worst performing languages for both baseline systems.

They were each trained for up to 20 epochs, with an early stop after 5 consecutive failures to improve on validation LAS. These stopping points turned out to be epoch 12, 12, and 13, respectively. The results appear in Tables 3-5.

Future work would, in addition to training on the remaining treebanks, include deeper hyperparameter exploration and benchmarks around run time, memory usage, and learning rates.

Nevertheless, this model successfully and significantly outperformed, in terms of LAS, both baselines and the top participating system in each language. Even though the structure was optimized on English, it performed well on Chinese and Vietnamese without modification. This is an encouraging result both in terms of the power and generalizability of this architecture. Hopefully, some of the models from which this architecture draws will soon publish results on the same data so that further comparisons can be made.

[9] http://universaldependencies.org/conll17/results-las.html

Table 3: Final Results - English Treebank

System	LAS
This system	**86.37**
UDPipe	85.82
SyntaxNet	84.20
Top PS	82.23

Table 4: Final Results - Chinese Treebank

System	LAS
This system	**83.51**
UDPipe	79.37
SyntaxNet	71.96
Top PS	68.56

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. *CoRR* abs/1603.06042. http://arxiv.org/abs/1603.06042.

Jan A. Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. *CoRR* abs/1405.4273. http://arxiv.org/abs/1405.4273.

Trinh-Minh-Tri Do and Thierry Artieres. 2010. Neural conditional random fields. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. JMLR: WCP, volume 9.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR* abs/1611.01734. http://arxiv.org/abs/1611.01734.

Table 5: Final Results - Vietnamese Treebank

System	LAS
This system	**69.09**
UDPipe	66.22
SyntaxNet	55.61
Top PS	47.51

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *CoRR* abs/1603.04351. http://arxiv.org/abs/1603.04351.

Terry Koo, Amir Globerson, Xavier Carreras, and Michael Collins. 2007. Structured prediction models via the matrix-tree theorem. In *In EMNLP-CoNLL*.

Lisha Li, Kevin G. Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2016. Efficient hyperparameter optimization and infinitely many armed bandits. *CoRR* abs/1603.06560. http://arxiv.org/abs/1603.06560.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *CoRR* abs/1508.04025. http://arxiv.org/abs/1508.04025.

Xuezhe Ma and Eduard H. Hovy. 2017. Neural probabilistic model for non-projective MST parsing. *CoRR* abs/1701.00874. http://arxiv.org/abs/1701.00874.

Ryan Mcdonald, Fernando Pereira, Kiril Ribarov, and Jan Haji. 2005. Non-projective dependency parsing using spanning tree algorithms. In *In Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. pages 523–530.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. *CoRR* abs/1310.4546. http://arxiv.org/abs/1310.4546.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2012. Understanding the exploding gradient problem. *CoRR* abs/1211.5063. http://arxiv.org/abs/1211.5063.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. European Language Resources Association (ELRA), Paris, France. http://www.lrec-conf.org/proceedings/lrec2016/pdf/873$_{P}aper.pdf$.