CCG Supertagging Using Morphological and Dependency Syntax Information

Luyện Ngọc Lê[1] and Yannis Haralambous[2]

[1] IMT Atlantique & UMR CNRS 6285 Lab-STICC, CS 83818, 29238 Brest Cedex 3, France
luyen.le@imt-atlantique.fr, ORCID: 0000-0002-2043-0679

[2] IMT Atlantique & UMR CNRS 6285 Lab-STICC, CS 83818, 29238 Brest Cedex 3, France,
yannis.haralambous@imt-atlantique.fr, ORCID: 0000-0003-1443-6115

Abstract. After presenting a new CCG supertagging algorithm based on morphological and dependency syntax information, we use this algorithm to create a CCG French Tree Bank corpus (20,261 sentences) based on the FTB corpus by Abeillé *et al.* We then use this corpus, as well as the Groningen Tree Bank corpus for the English language, to train a new BiLSTM+CRF neural architecture that uses (a) morphosyntactic input features and (b) feature correlations as input features. We show experimentally that for an inflected language like French, dependency syntax information allows significant improvement of the accuracy of the CCG supertagging task, when using deep learning techniques.

Keywords: CCG Supertagging · Dependency Syntax · FTB Corpus · BiLSTM, CRF

1 Introduction

Combinatory Categorial Grammars (CCG) [36] provide a transparent interface between syntax and underlying semantic representation. They allow access to a deep semantic structure of the sentence and facilitate recovering of non-local dependencies involved in the construction such as coordination, extraction, control, and raising. CCGs have been introduced by Mark Steedman [34, 35] as a non-transformational grammatical theory relying on combinatory logic. CCGs are strongly lexicalized in the sense that words are associated with one or more syntactic types, called *lexical categories*. These can be basic (e.g., S, NP, PP) or complex, obtained by using the *functors* / and \ on basic categories (e.g., S/NP, NP\PP etc). Each lexical category has a specific meaning, for example, NP is the noun or the noun phrase, S\NP represents a verb phrase or an intransitive verb that requires a subject (NP) on its left as argument (cf. Figure 1 for an example). The process of assigning lexical categories to words is called *supertagging* because, contrarily to POS tags, CCG tags are detailed syntactic structures.

In this paper we first assign CCG labels to words using syntax dependencies and POS tags, and then we build complete CCG derivation trees for sentences in

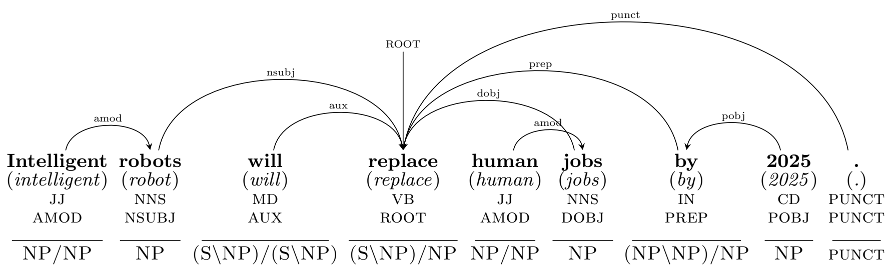
punct
ROOT
prep
nsubj
dobj
aux
amod
amod
pobj
Intelligent (intelligent) JJ AMOD NP/NP
robots (robot) NNS NSUBJ NP
will (will) MD AUX (S\NP)/(S\NP)
replace (replace) VB ROOT (S\NP)/NP
human (human) JJ AMOD NP/NP
jobs (jobs) NNS DOBJ NP
by (by) IN PREP (NP\NP)/NP
2025 (2025) CD POBJ NP
. (.) PUNCT PUNCT PUNCT

Fig. 1. A sentence with POS tags, dependencies and CCG lexical categories.

a traditional approach. Afterwards we use this information as input features to train a neural network, in order to improve the accuracy of a CCG supertagging task model.

CCG supertagging plays an important role in parsing systems, as a preliminary step to the build of complete CCG derivation trees. In general, this task can be considered as a sequence labeling problem with input sentence $s_{\text{input}} = (w_1, w_2, \ldots, w_n)$ and the CCG supertags $s_{\text{output}} = (t_1, t_2, \ldots t_n)$ as output. Input features can be words or they can be extracted from words, such as suffix, capitalization property or characters selection [25, 40, 41, 2, 23]. We will use morphosyntactic annotations such as lemma, suffix, POS tags and dependency relations [20] to build feature sets. These annotations are extremely useful in order to add additional information about word as well as long-range dependencies in the sentence (Figure 1). These novel features allow us to improve accuracy of a supertagging neural network. We also consider adding correlations between features as additional input features of the network and examine the results.

In the past few years, Recurrent Neural Networks (RNN) [15] along with its variants such as Long-Short Term Memory (LSTM) [17, 14] and GRU [8] have been proven to be effective for many NLP tasks, and especially for sequence labeling such as POS tagging, Named Entity Recognition etc. In the CCG supertagging task, different RNN-based models have been proposed and have obtained high accuracy results. Following this trend, we base our model on the Bi-Directional LSTM (BiLSTM) architecture associated with Conditional Random Fields (CRF) as output layer. Thus, we take advantage of the ability to remember the information of previous and next words in the sentence with the BiLSTM network and increase the ability to learn from the relationship of output labels with CRF.

The main contributions of our work are:

1. the use of morphosyntactic information for the traditional supertagging task;
2. the creation of a CCG Tree Bank for French language using this method;

3. the use of a new neural network architecture based on BiLSTM and CRF for the supertagging task trained on a standard English Tree Bank and on our French Tree Bank.

As we will see, morphosyntactic information and the new neural network architecture improve the task performance significantly in the case of French language, but not significantly in the case of English language.

The remainder of the paper is organized as follows. In the next section we describe our approach of using POS tags and dependency relations in (traditional) CCG supertagging. In Section 3, we discuss the state of the art of machine learning methods for CCG supertagging. In Section 4 we present our new neural network model architecture. In Section 5, we evaluate our method on an English and a French corpus, the latter been developed by ourselves using the methods described in Section 1. Finally, we conclude and discuss open questions.

2 From Dependency Syntax to CCG Derivation Tree

In [1], Abeillé, Clément & Kinyon announce a French Tree Bank based on syntax constituents. In [7, 6] Candito, Crabbé & Denis convert this Tree Bank into syntax dependencies. We use information from these two corpora to build a CCG Tree Bank, as described in this section.

Dependency parsing consists in building a tree rooted at the head of the sentence (usually the verb), the edges of which, called *dependencies*, connect words and are labeled by syntactic functions, e.g., subject, object, oblique, determiner, attribute etc. Dependency syntax trees are obtained by parsers such as MaltParser [30], Stanford Parser [12], MST parser [28], Spacy [19], etc. These tools also provide POS tags of words.

In order to assign CCG lexical categories to words of a sentence, we start by calculating its dependency tree. Then, we process words which have unique lexical categories in the corpus: e.g., nouns have lexical category NP, adjectives have lexical category NP/NP or NP\NP depending whether they are on the left or on the right of the noun, etc. Once we have assigned these unique (or position-dependent, as in adjectives) lexical categories, we move over to verbs.

The main verb of the sentence, which is normally the root of the dependency tree, may have *argument dependencies*, labeled *suj*, *obj*, *a_obj*, *de_obj*, *p_obj*, i.e., correspondences with subject, direct and indirect object, and/or *adjunct dependencies* labeled *mod*, *ats*, etc., representing complementary information such as number, time, place, and so on. We assign lexical category S\NP to a main verb having a subject to its left, and then we add a /NP (or a \NP, depending on its position with respect to the verb) for each direct object or indirect object (in the order of words in the sentence).

Our next step is to binarize the dependency tree on the basis of information about dominant sentence structure: In French, most sentences are SVO, as in "*Mon fils (S) achète (V) un cadeau (O)*" (My son buys a gift), or SOV as in "*Il (S) le (O) donnera (V) à sa mère (indirect O)*" (He will give it to his mother).

Using this general linguistic property, we can extract and classify the components of the sentence into: subject, direct object, indirect object, verbs, complement phrases.

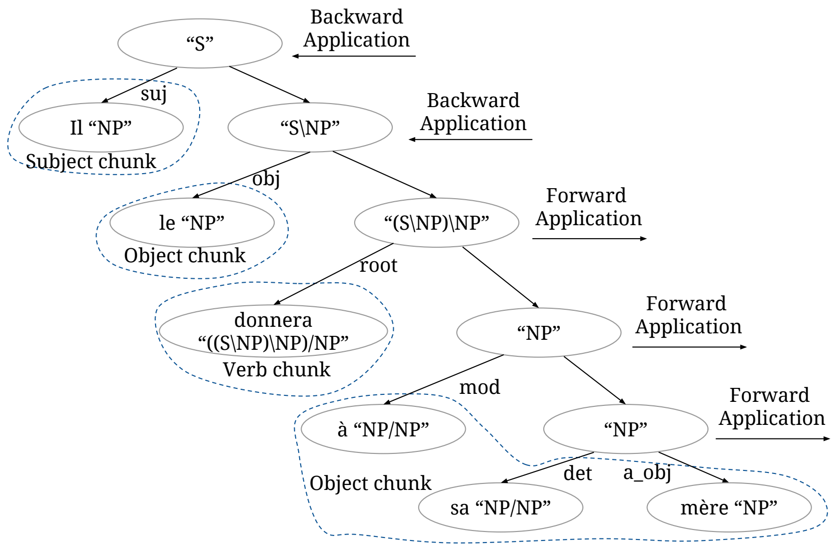

Fig. 2. CCG derivation tree of sentence "He will give it to his mother."

The algorithm we propose for transforming a dependency tree into a binary tree consists is subdivided into two steps:

1. we extract chunks from the dependency tree based on syntactic information and dependency labels between words. For example, the subject chunk is obtained by finding a word that has a dependency labeled *suj*, the verb chunk corresponds to the root of the dependency structure, direct or indirect object chunks are obtained as words with links directed to the root verb and having labels *obj* or *p_obj*, etc.;
2. we build a binary tree for each chunk, and then combine the binary trees in inverse order of the dominant sentence structure. For example if SVO is the dominant structure, we start by building the binary tree of the *object* chunk, then combine it with the binary tree of the *verb* chunk, and finally we obtain the binary tree of the *subject* chunk.

In Fig. 2, the reader can see four chunk groups in the dependency tree, displayed as regions of the binarized tree.

By applying this approach to the complete set of 21,550 dependency trees of the FTB corpus, we have obtained a bank of CCG derivation trees for 94,02 % of the sentences. This new corpus is available (together with the code implementing the algorithm described above) as a resource in the frame of the verifiability, reproducibility, and working description policy of CICLING 2019 conference.

3 Machine Learning and Supertagging

One of the first applications of machine learning to CCG supertagging is the development of a statistical parser by Clark & Curran [10]. They proceed in two main steps: supertagging and combining of lexical categories. Their supertagging approach is based on the log-linear model by using the lexical category set in a local five-word context to obtain a distribution. The model's features are words POS tags included in the five-word window, plus the two previously assigned lexical categories (to the left). They applied their method on the CCG Bank corpus [18] with 92.6% of accuracy for words and (only) 36.8% of accuracy for complete sentences (that is the percentage of sentences of which *all* words are tagged correctly).

Like many others supervised methods, CCG supertagging requires a sufficiently large amount of labeled training data to achieve a good result. Mike & Mark [26] have introduced a semi-supervised approach to improve a CCG parser with unlabeled data. They have constructed a model for the prediction of CCG lexical categories, based on vector-space embeddings. Features are words and some other information (e.g., POS tagging, chunking, named-entity recognition, etc.) in the context window. Their experiments used the neural network model of Collobert [11] in association with conditional random fields (CRF) [38].

Using RNN for CCG supertaging has been proven to provide better results with a similar set of features and window size in the work of Xu [42]. However, the conventional RNN is often difficult to train and there still exist problems such as gradient vanishing and exploding, in the layers over long sequences [4, 31]. Therefore, LSTM networks—a special variant of RNN which is capable of learning long-term dependencies—were proposed to overcome these RNN limitations. In particular, Bi-directional LSTM network models have been created with the ability to store two-way information, and the majority of literature in the area [25, 40, 41, 2, 23] uses this model with different training procedures and achieves high accuracy.

The performance of BiLSTM networks models has been improved by combining them with a CRF model for the sequence labeling task [21, 36, 27]. Using a BiLSTM-CRF model similar to the one in [21], the authors of [22] have shown the efficiency of CRF by achieving a higher accuracy in CCG supertagging and multi-tagging tasks.

In most of the above works, similarly to many sequence labeling tasks, the model inputs are words and their features are extracted directly from words. However, we claim that lexical categories assignment to words can use morphological and dependency syntax to enrich the feature set. In the following section, we present a neural network model based on BiLSTM-CRF architecture with moprhosyntactic features.

4 Neural Network Model

4.1 Input Features

We will use the following input features for words in sentences:

- the *word* per se (word);
- the *word lemma* (lemma);
- the *POS tag* of the word (postag);
- the *dependency relation* (deprel) of the word with its parent in the dependency tree (and the tag "root" for the head of the dependency tree, which has no parent).

Each one of these features provides predictive information about the CCG supertag label. Therefore, our input sentence will be $s = \{x_1, x_2, ...x_n\}$ where each x_i is a vector of the features $x_i = [\mathrm{word}_i, \mathrm{lemma}_i, \mathrm{postag}_i, \mathrm{deprel}_i]$.

Before describing our model, let us briefly review, in the following section, pre-existant models with which we will compare it.

4.2 Basic Bi-Directional LSTM and CRF Models

Unidirectional LSTM Model As mentioned earlier, the shortcomings of standard RNNs in practice involve gradient vanishing and an explosion problem when dealing with long term dependencies. LSTMs are designed to cope with these gradient problems. Basically, a conventional RNN is defined as follows: the input $x = (x_1, x_2, \dots, x_T)$ feeds the network, and the network computes the hidden vector sequence $h = (h_1, h_2, \dots, h_T)$, and the output sequence, $y = (y_1, y_2, \dots, y_T)$, from $t = 1, \dots, T$ where T is the number of time steps as in the following formulas:

$$h_t = f(Ux_t + Wh_{t-1} + b_h) \tag{1}$$

$$y_t = g(Vh_t + b_y), \tag{2}$$

where U, W, V denote weight matrices that are computed in training time, b denotes bias vectors and $f(z)$, $g(z)$ are activation functions.

Based on the basic architecture of a RNN, an LSTM layer is formed from a set of memory blocks [17, 16]. Each block contains one or more recurrently connected memory cells and three gate units: input, output and forget gate. More specifically, activation computation in a memory cell at time step t is defined by the following formulas:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{3}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{4}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \tag{5}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{6}$$

$$h_t = o_t \odot \tanh(c_t), \tag{7}$$

where i_t, f_t, o_t, c_t correspond to input gate, forget gate, output gate and cell vectors, σ is the logistic sigmoid function, tanh is the hyperbolic tangent function, W terms denote weight matrices, and b terms denote bias vectors.

Bidirectional LSTM Model In order to assign a supertag to a word, we need to use the word's information and its relations to the previous and next word in the sentence. Two-way information access from past to future and vice versa gives global information in a sequence. However, the LSTM cell only retrieves information from the past using input and output of the previous LSTM cell. In other words, an LSTM cell does not receive any information from the LSTM cell *following* it. Therefore, a Bi-Directional LSTM (BiLSTM) model has been proposed in [33, 3] to overcome this problem, as follows:

$$\text{Bi-LSTM}_{\text{sequence}}(x_{1:n}) = \text{LSTM}_{\text{backward}}(x_{n:1}) \circ \text{LSTM}_{\text{forward}}(x_{1:n}). \tag{8}$$

In general architectures, one may have one forward LSTM layer and one backward LSTM layer for the complete sequence and run them in reverse time. The features of the two layers are concatenated at the level of the output layers. Thus, information from both the past and the future is transmitted to each memory LSTM cell. The hidden state is computed as follows:

$$h_t = f(W_{\overleftarrow{h}} \overleftarrow{h_t} + W_{\overrightarrow{h}} \overrightarrow{h_t}), \tag{9}$$

where $\overleftarrow{h_t}$ is backward hidden sequence, $\overrightarrow{h_t}$ is the forward hidden sequence.

CRF Model BiLSTM networks are used to build efficient predictive models of the output sequence based on the features of the input sequence. However, they can not consider the correlation between output labels and their neighborhoods. In our case, CCG supertags, by nature, *always* have correlations with the previous or next labels, for example, an output CCG supertag of a word is NP/NP (usually an article), which allows us to predict the fact that the next CCG supertag is NP.

In order to enhance the ability to predict next labels from current label in an output sequence, two approaches can be used:

1. building a tag distribution for each training step and using an heuristic search algorithm to find optimal tag sequences [39];
2. focusing on the context with sentence-level information instead of only word-level information. The leading work of this approach is the CRF model of [24].

We use the second approach in the output layer of our model. The combination of BiLSTM network and CRF network can improve the efficiency of the model by strengthening the relationship between the output labels through the CRF layer, based on the input features through the BiLSTM layer.

4.3 Our Model for Feature Set Enrichment and CCG Supertagging

In the model we propose (see Fig. 3), each input is a set of features: word, lemma, POS tag and dependency relation. These features are vectorized with a fixed size by using the embedding matrix in the embedding layer. In the next layer, the

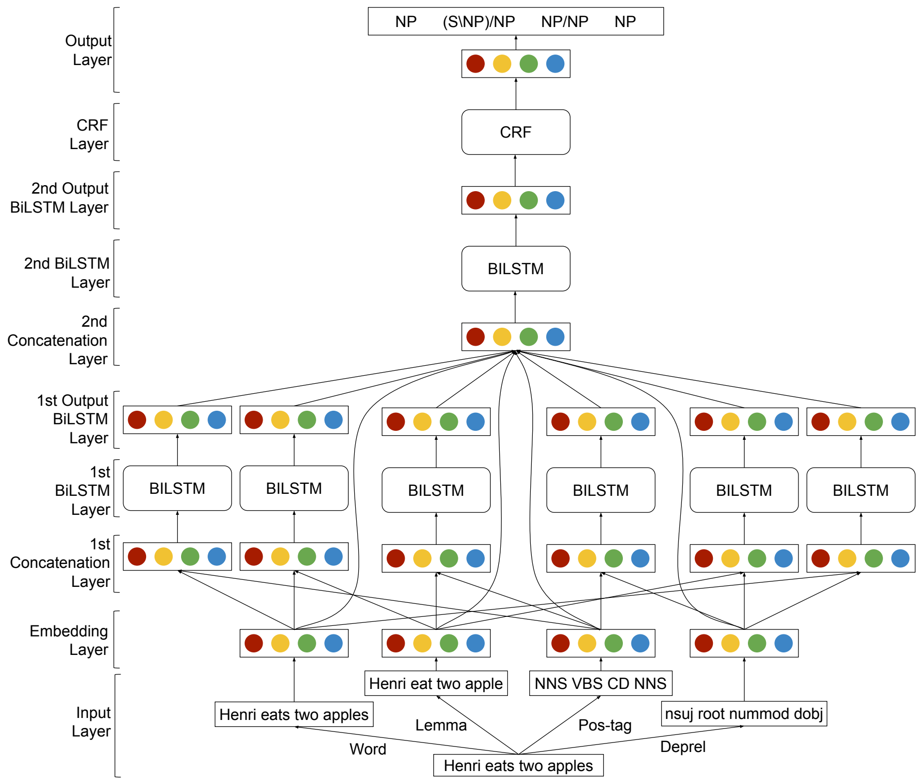

Fig. 3. Architecture of the BiLSTM network with a CRF layer on the outputs.

correlation between pairs of features is calculated by combining them. Then, we use a BiLSTM network to memorize and learn the relations with other words in the context of the sentence for these pairs of features. After that, all features are concatenated to become the input of the second BiLSTM network layer. Finally, CCG supertag labels are obtained in the output of the CRF network layer, the input of which is the output of the 2nd output BiLSTM layer.

5 Evaluation

5.1 Dataset and Preprocessing

We use the two different corpora to experiment our model, one in English and one in French. The first corpus is the Groningen Meaning Bank (GMB) corpus [5] which has been built for deeper semantic analysis on a discourse scope. The second one is our CCG Corpus for French which we extracted from the French Tree Bank (FTP) corpus [1] by using the dependency analysis of the sentence (see Section 2).

In order to obtain a standard dataset for training process, we extract all sentences with annotations for each word such as lemma, POS tag, dependency relation and CCG label, for each corpus. As the GMB corpus does not contain dependency relations, we have used the Stanford Parser [12] to add it a posteriori.

We compare the structures of the two corpora in Table 1. In particular, there is a difference in the distribution of sentences according to their length (see also Fig. 4). In the GMB corpus, the distribution of the number of short sentences and long sentences is relatively similar. This is quite different in the FTB corpus where there are more short sentences, and where long sentences spread over a wider range. This difference of the distribution rate in the datasets can affect the training process outcomes of the two corpora.

Statistic	#Sentences	#Words	#Word tokens	#Lemma tokens	#POS labels	#Deprels	#CCG labels
GMB	23,451	1,037,739	32,073	26,987	43	56	636
FTB	18,724	570,054	28,748	18,762	29	27	73

Table 1: Statistics on the two corpora.

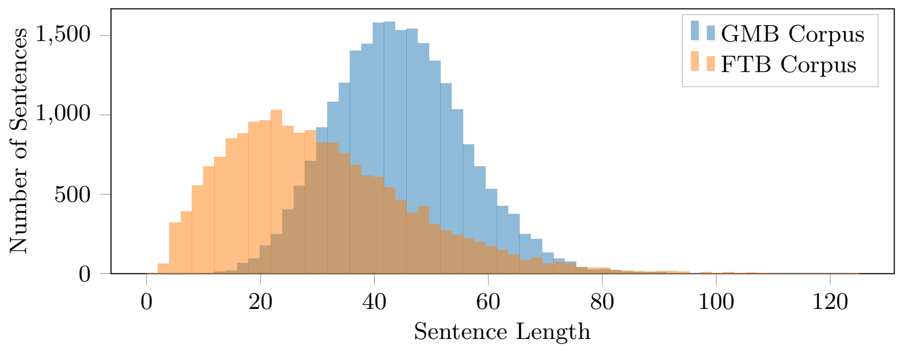

Fig. 4. Histogram of sentence length in the corpora.

5.2 Training procedure

We implement our neural network by using the Keras deep learning library [9]. The datasets are divided into three sets: training set, validation set and test set with the proportion

$$(0.8 * (\text{training\_set}) + 0.2 * (\text{validation\_set})) * 0.8 + 0.2 * (\text{test\_set}).$$

Validation sets are used to measure performance at each epoch. Final evaluation on the test set is based on the best accuracy results in the validation sets.

Pre-trained Word Embeddings In order to work with numeric data in the neural network, we use pre-trained word embeddings to transform words or lemmas of the corpora into numeric vectors. More specifically, we use Glove [32] (a 200-dimensional embedding trained on 6 billion words collected from Wikipedia) for the GMP corpus, and Word2vec [29] (the French version by Fauconnier [13], also with 200 dimensions, and trained on 1.6 billion words collected from the web) for the FTB corpus. For out-of-vocabulary words, we assign embeddings by random samples. Based on the distribution by length of sentences (Fig. 4), we assign a fixed length of 120 words to all sentences, so that the input dimension is $[120, 200]$ for each sentence input. Finally, the other features are transformed to numeric vectors by using a one-hot encoding matrix with size depending on their number in the dictionary.

Parameters and Hyperparameters We fix the number of training examples (batch size) as 32 for each forward or backward propagation. Each training process runs 20 times to evaluate and compare outcomes (epoch). In addition, we have experimented with the number of different hidden states, such as 64, 128, 256, 512 to find a configuration that is optimally consistent with the model. We decided to carry on the experiment with 128 hidden states because this choice optimally balances accuracy and performance.

Optimization algorithm Choosing an optimizer is a crucial part of the model building process. Our model uses the Root Mean Square Prop (RMSprop) optimizer [37] which proceeds by keeping an exponentially weighted average of the squares from past gradients. To increase convergence, the learning rate is divided by this average:

$$v_{dw} := \beta v_{dw} + (1-\beta) \cdot dw^2 \tag{10}$$

$$v_{db} := \beta v_{db} + (1-\beta) \cdot db^2 \tag{11}$$

$$W := W - \alpha \tfrac{dw}{\sqrt{v_{dw}}+\epsilon} \tag{12}$$

$$b := b - \alpha \tfrac{db}{\sqrt{v_{dw}}+\epsilon}, \tag{13}$$

where v_{dw} v_{db} are the exponentially weighted averages from past squares of gradients, dw^2 and db^2 are cost gradient related to the current layer weight, W and b denote weight and bias, α is the learning rate from 0.9 to 0.0001 ($\alpha = 0.01$ is the default setting), β is an hyperparameter to be tuned and ϵ is very small to avoid dividing by zero.

5.3 Experimental Results

In order to evaluate the proposed input features and the model, we conduct two sorts of comparison:

1. we compare the outcomes of different feature sets such as [word], [word, suffix], [word, suffix, cap(italization)], [word, lemma, suffix, cap], [word, lemma,

postag, suffix, cap], [word, lemma, postag, deprel, suffix, cap], [lemma, postag, deprel], [lemma, postag], [lemma];
2. we compare the outcomes of different neural network architectures such as BiLSTM [25], standard BiLSTM CRF [21], Double-BiLSTM CRF [22], and ours.

Evaluation on our test set is shown on Table 5.3 for the French FTB corpus and on Table 3 for the English GMB corpus.

According to our architecture and since we use correlations of features as additional features, our model requires at least two features in the input data. Therefore, we can not produce results on input data with a single feature like [word] or [lemma]. Nevertheless we compare the outcome of other models on our input features, including single input features.

Feature set	BiLSTM	BiLSTM CRF	Double BiLSTM CRF	Our model
word	*78.60*	*78.76*	*77.14*	-
word, suffix	*78.97*	*78.80*	*76.58*	78.90
word, suffix, cap	*78.56*	*78.97*	***75.96***	84.43
word, lemma, suffix, cap	79.16	79.67	78.78	78.49
word, lemma, postag, suffix, cap	81.28	81.84	81.24	81.50
word, lemma, postag, deprel, suffix, cap	83.23	83.95	83.56	84.06
word, lemma, postag, deprel	83.43	83.98	83.70	**85.05**
lemma,postag,deprel	83.00	83.05	83.15	82.40
lemma,postag	80.20	80.37	81.40	80.05
lemma	77.61	77.83	76.66	-

Table 2: 1-best tagging accuracy comparison results on the test set in the French FTB Corpus.

Let us first start with the French corpus. In Table we have displayed methods from the literature in italics: word, suffix and cap(italization) as input features, BiLSTM, BiLSTM+CRF and Double BiLSTM+CRF as architectures. As the reader can see, by applying pre-existing methods we obtain a maximum accuracy of 75.96%. By using our input features with pre-existing architectures we obtain a maximum accuracy of 83.98%. By using our architecture with input features used by others we get an accuracy of 84.43%. Both of these results are significantly better than those in previous works. Finally, by combining our input features with our model we manage to gain another 1% and achieve a topmost accuracy of 85.05%.

It is interesting to notice that, even though the lemma feature carries less information than the word feature (as expected), the combination of lemma and POS tag features provides better results than the word feature, and that these results are systematically increased by 2% when dependency relations are added as well.

Feature set	BiLSTM	BiLSTM CRF	Double BiLSTM CRF	Our Model
word	*92.83*	*92.49*	*91.16*	-
word, suffix	*93.08*	*92.93*	*91.57*	92.92
word, suffix, cap	***93.30***	*93.20*	*91.48*	**94.31**
word, lemma, suffix, cap	93.33	93.26	91.78	93.38
word, lemma, postag, suffix, cap	93.29	93.02	93.25	92.44
word, lemma, postag, deprel, suffix, cap	93.45	93.18	93.15	92.46
word, lemma, postag, deprel	93.35	93.18	93.24	92.90
lemma, postag, deprel	93.25	93.13	92.98	93.26
lemma, postag	93.21	93.12	92.98	92.95
lemma	90.56	90.13	89.86	-

Table 3: 1-best tagging accuracy comparison results on the test set in English GMB Corpus.

The accuracy results for the English GMB corpus are displayed on Table 3. Here differences are less significant, and the results all lie in the 92–94% range, with a single exception: the case of the lemma feature, where we lose about 2% of accuracy. Nevertheless, when we add the POS tag feature to the lemma feature, we get a slightly better result than the word feature (the difference is about 1%). The best result is an accuracy of 94.31%, obtained by our model, but not with our morphosyntactic features but rather with the legacy word, suffix and cap(italization) features.

A general conclusion could be that morphosyntactic information brings a real advantage for neuronal supertagging of French (a language the verbs of which are highly inflected). It would be interesting to test the model with even more inflected languages such as German, Russian or Greek.

6 Conclusion

We have presented a new CCG supertagging task based on morphological and dependency syntax information, which has allowed us to create a CCG version of the French Tree Bank corpus FTB. We used this corpus to train a new BiLSTM+CRF neural architecture that uses new, morphosyntactic, input features as well as feature correlations as separate input features. We have experimentally shown that, at least for an inflected language as French, dependency syntax information is useful for improving the accuracy of the CCG supertagging task when using deep learning techniques.

Both the CCG French Tree Bank corpus we have developed as the code we used for the traditional and for the deep learning supertaggers are available in the frame of the verifiability, reproducibility, and working description policy of the CICLING 2019 conference.

References

1. Abeillé, A., Clément, L., Toussenel, F.: Building a treebank for French. In: Treebanks: Building and Using Parsed Corpora, pp. 165–187. Kluwer (2003)
2. Ambati, B.R., Deoskar, T., Steedman, M.: Shift-reduce CCG parsing using neural network models. In: Proceedings of NAACL 2016. pp. 447–453 (2016)
3. Baldi, P., Brunak, S., Frasconi, P., Pollastri, G., Soda, G.: Bidirectional dynamics for protein secondary structure prediction. In: Sequence Learning, LNCS, vol. 1828, pp. 80–104. Springer (2000)
4. Bengio, Y., Frasconi, P., Simard, P.: The problem of learning long-term dependencies in recurrent networks. In: IEEE International Conference on Neural Networks 1993. pp. 1183–1188. IEEE (1993)
5. Bos, J., Basile, V., Evang, K., Venhuizen, N., Bjerva, J.: The Groningen Meaning Bank. In: Ide, N., Pustejovsky, J. (eds.) Handbook of Linguistic Annotation, vol. 2, pp. 463–496. Springer (2017)
6. Candito, M., Crabbé, B., Denis, P.: Statistical French dependency parsing: treebank conversion and first results. In: Proceedings of LREC 2010. pp. 1840–1847 (2010)
7. Candito, M., Crabbé, B., Denis, P., Guérin, F.: Analyse syntaxique du français: des constituants aux dépendances. In: Proceedings of TALN 2009 (2009)
8. Cho, K., Van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches, arXiv:1409.1259
9. Chollet, F.: Deep Learning with Python. Manning Publications (2018)
10. Clark, S., Curran, J.R.: Wide-coverage efficient statistical parsing with CCG and log-linear models. Computational Linguistics **33**(4), 493–552 (2007)
11. Collobert, R.: Deep learning for efficient discriminative parsing. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. pp. 224–232 (2011)
12. De Marneffe, M.C., MacCartney, B., Manning, C.D., et al.: Generating typed dependency parses from phrase structure parses. In: Proceedings of LREC 2006. vol. 6, pp. 449–454 (2006)
13. Fauconnier, J.P.: French word embeddings (2015), http://fauconnier.github.io
14. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with LSTM. In: Proceedings of ICANN 99. IET (1999)
15. Goller, C., Kuchler, A.: Learning task-dependent distributed representations by backpropagation through structure. Neural Networks **1**, 347–352 (1996)
16. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks **18**(5-6), 602–610 (2005)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
18. Hockenmaier, J., Steedman, M.: CCGbank: a corpus of CCG derivations and dependency structures extracted from the Penn Treebank. Computational Linguistics **33**(3), 355–396 (2007)
19. Honnibal, M., Johnson, M.: An improved non-monotonic transition system for dependency parsing. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 1373–1378 (2015)
20. Honnibal, M., Kummerfeld, J.K., Curran, J.R.: Morphological analysis can improve a CCG parser for english. In: Proceedings of Coling 2010. pp. 445–453 (2010)

21. Huang, Z., Xu, W., Yu, K.: Bidirectional LSTM-CRF models for sequence tagging, arXiv:1508.01991
22. Kadari, R., Zhang, Y., Zhang, W., Liu, T.: CCG supertagging via Bidirectional LSTM-CRF neural architecture. Neurocomputing **283**, 31–37 (2018)
23. Kadari, R., Zhang, Y., Zhang, W., Liu, T.: CCG supertagging with bidirectional long short-term memory networks. Natural Language Engineering **24**(1), 77–90 (2018)
24. Lafferty, J., McCallum, A., Pereira, F.C.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proceedings of ICML 2001. pp. 282–289 (2001)
25. Lewis, M., Lee, K., Zettlemoyer, L.: LSTM CCG parsing. In: Proceedings of NAACL 2016. pp. 221–231 (2016)
26. Lewis, M., Steedman, M.: Improved CCG parsing with semi-supervised supertagging. Transactions of the ACL **2**, 327–338 (2014)
27. Ma, X., Hovy, E.: End-to-End Sequence Labeling via Bi-directional LSTM-CNNS-CRF, arXiv:1603.01354
28. McDonald, R., Pereira, F., Ribarov, K., Hajič, J.: Non-projective dependency parsing using spanning tree algorithms. In: Proceedings of EMNLP 2005. pp. 523–530 (2005)
29. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of NIPS'13. pp. 3111–3119 (2013)
30. Nivre, J., Hall, J., Nilsson, J.: Maltparser: A data-driven parser-generator for dependency parsing. In: Proceedings of LREC 2006. pp. 2216–2219 (2006)
31. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training recurrent neural networks. In: International Conference on Machine Learning. pp. 1310–1318 (2013)
32. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of EMNLP 2014. pp. 1532–1543 (2014)
33. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing **45**(11), 2673–2681 (1997)
34. Steedman, M.: Surface structure and interpretation. MIT press (1996)
35. Steedman, M.: The syntactic process. MIT press (2000)
36. Steedman, M., Baldridge, J.: Combinatory categorial grammar. In: Non-Transformational Syntax: Formal and explicit models of grammar. pp. 181–224. Wiley-Blackwell (2011)
37. Tieleman, T., Hinton, G.: Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning **4**(2), 26–31 (2012)
38. Turian, J., Ratinov, L., Bengio, Y.: Word representations: a simple and general method for semi-supervised learning. In: Proceedings of the 48th annual meeting of the association for computational linguistics. pp. 384–394 (2010)
39. Vaswani, A., Bisk, Y., Sagae, K., Musa, R.: Supertagging with LSTMs. In: Proceedings of NAACL 2016. pp. 232–237 (2016)
40. Wu, H., Zhang, J., Zong, C.: A dynamic window neural network for CCG supertagging. In: Proceedings of AAAI-17. pp. 3337–3343 (2017)
41. Xu, W.: LSTM shift-reduce CCG parsing. In: Proceedings of EMNLP 2016. pp. 1754–1764 (2016)
42. Xu, W., Auli, M., Clark, S.: CCG supertagging with a recurrent neural network. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics. vol. 2, pp. 250–255 (2015)