State-of-the-art Italian dependency parsers based on neural and ensemble systems

Oronzo Antonelli*
Università di Bologna

Fabio Tamburini**
Università di Bologna

In this paper we present a work which aims to test the most advanced, state-of-the-art syntactic dependency parsers based on deep neural networks (DNN) on Italian. We made a large set of experiments by using two Italian treebanks containing different text types downloaded from the Universal Dependencies project and propose a new solution based on ensemble systems. We implemented the proposed ensemble solutions by testing different techniques described in literature, obtaining very good parsing results, well above the state of the art for Italian.

1. Introduction

Syntactic parsing of morphologically rich languages like Italian often poses a number of hard challenges. Various works applied different kinds of freely available parsers on Italian training them using different treebank resources and different methods to compare their results (Lavelli 2014; Alicante et al. 2015; Lavelli 2016) and gather a clear picture of the syntactic parsing task performance for the Italian language. In this direction, it seems relevant to cite the EVALITA[1] periodic campaigns for the evaluation of constituency and dependency parsers devoted to the syntactic analysis of Italian (Bosco and Mazzei 2011; Bosco et al. 2014).

Other studies regarding the syntactic parsing of English (Nivre and McDonald 2008; Surdeanu and Manning 2010) or Italian (Lavelli 2013; Mazzei 2015) tried to enhance the parsing performance by building some kind of *ensemble systems*.

By looking at the cited papers we can observe that they evaluated the state-of-the-art parsers before the "neural networks revolution" not including, with few exceptions, the last improvements proposed by new research studies.

The goal of this paper is twofold: first we would like to test the effectiveness of parsers based on the newly-proposed technologies, mainly deep neural networks, on Italian; second, we would like to propose an ensemble system able to further improve the neural parsers performance when parsing Italian texts.

This paper is structured as follows: Section 2 describes the architectures of the nine parsers we tested; Section 3 illustrates the datasets we employed for the evaluations; in Section 4 we will show the results of the single parsers evaluation while in Section 5 we will describe the different kind of ensemble systems that we propose to evaluate using the same datasets. In Section 6 we will draw some provisional conclusions.

* Dept. of Computer Science and Engineering - Mura Anteo Zamboni 7, 40126 Bologna, Italy.
E-mail: antonelli.oronzo@gmail.com
** Dept. of Classic Philology and Italian Studies - Via Zamboni 32, 40126 Bologna, Italy.
E-mail: fabio.tamburini@unibo.it
1 http://www.evalita.it

2. The Neural Dependency Parsers

We considered nine state-of-the-art parsers representing a wide range of contemporary approaches to dependency parsing whose architectures are based on neural network models.

In (Chen and Manning 2014) it is proposed, for the first time, to represent words, Part-of-speech (PoS) tags and dependency types through a dense encoding using embedding vectors. In this way it is possible to automate the learning process of the features avoiding to extract them following manually designed templates. The parser was developed using the transition-based approach and it learns a classifier based on a neural network that chooses the correct transition using the arc-standard system and a greedy deterministic parsing algorithm. Each word w_i, PoS tag t_j and dependency type l_k is represented by its respective embedding vector $\boldsymbol{e}_i^w, \boldsymbol{e}_j^t, \boldsymbol{e}_k^l$. The neural network chosen is a Multi-Layer Perceptron (MLP) with only one hidden layer. In the three layers data are represented by the features vector $[\boldsymbol{x}^w, \boldsymbol{x}^t, \boldsymbol{x}^l]$. The $\boldsymbol{x}^w$ vector is composed of 18 embedding vectors for words in a given stack or buffer position of a configuration, so $\boldsymbol{x}^w = [\boldsymbol{e}_1^w; \ldots; \boldsymbol{e}_{18}^w]$. The same is true for the vector $\boldsymbol{x}^t = [\boldsymbol{e}_1^t; \ldots; \boldsymbol{e}_{18}^t]$, which includes 18 PoS embedding, and the $\boldsymbol{x}^l = [\boldsymbol{e}_1^l; \ldots; \boldsymbol{e}_{12}^l]$, consisting of 12 dependency type embedding vectors. The hidden layer $\boldsymbol{h}$ computes a linear combination of the input layer, a cubic activation function is applied ($g(x) = x^3$) and the result is merged into a softmax layer that predicts the transition with the highest probability:

$$\begin{aligned} \boldsymbol{h} &= (\boldsymbol{W}_1^w \boldsymbol{x}^w + \boldsymbol{W}_1^t \boldsymbol{x}^t + \boldsymbol{W}_1^l \boldsymbol{x}^l + \boldsymbol{b}_1)^3 \\ \boldsymbol{p} &= \text{softmax}(\boldsymbol{W}_2 \boldsymbol{h}) \end{aligned} \tag{1}$$

The cubic activation function, in place of classical activations such as hyperbolic tangent ($\tanh$) and sigmoid function, is suitable for capturing the interactions between the three elements considered in model learning: words, PoS tags and dependency types. The training set $\mathcal{D}^{(\text{train})} = \{(c_i, t_i)\}$ consists of the configuration-transition pairs and does not use templates to transform the configuration. The parameters of the network are learned by minimizing the cross-entropy loss, using the l_2 regularization and the AdaGrad optimization algorithm with mini-batches.

In (Dyer et al. 2015) a transition-based parser based on the arc-standard system is proposed. This model try to learn the representation of the entire state of the parser, which is obtained from the representation of the buffer, the stack and the history of the actions taken by the parser. To represent the entire state it uses a technique that the authors called *LSTM stack*, based on Recurrent Neural Networks (RNN), more specifically on Long Short-Term Memories (LSTM) (Hochreiter and Schmidhuber 1997), and supports push and pop actions just like the elements of the parser configuration (stack and buffer). The idea is to represent the $S = stack(w_1, \ldots, w_n)$ through the final state of the RNN applied to the word sequence $w_1, \ldots, w_n$ contained in the stack. The model uses three *LSTM stack* structures for each element of the configuration: one for the stack S, one for the buffer B and the last for the history A. At each step t the parser uses the representations $\boldsymbol{s}_t, \boldsymbol{b}_t, \boldsymbol{a}_t$ of the elements S, B and A to determine the transition to apply. The representation of the parser state at step t, called $\boldsymbol{p}_t$, is defined as

$$\boldsymbol{p}_t = \max\{0, \boldsymbol{W}[\boldsymbol{s}_t; \boldsymbol{b}_t; \boldsymbol{a}_t] + \boldsymbol{d}\} \tag{2}$$

Each token $\boldsymbol{x}$ relative to any element of a configuration, given as input to the RNN, is represented by concatenating the embedding vector $\boldsymbol{w}$ of the word and the embedding vector $\boldsymbol{t}$ relative to its

PoS tag:

$$\boldsymbol{x} = \max\{0, \boldsymbol{V}[\boldsymbol{w}; \boldsymbol{t}] + \boldsymbol{b}\} \tag{3}$$

To represent the edges of the partial dependency tree in A it uses a vector $\boldsymbol{c}$ obtained from the composition of the embedding vectors $\boldsymbol{h}, \boldsymbol{d}, \boldsymbol{r}$, which indicate the head, the dependent and the type of dependency, respectively:

$$\boldsymbol{c} = \tanh(\boldsymbol{U}[\boldsymbol{h}; \boldsymbol{d}; \boldsymbol{r}] + \boldsymbol{e}) \tag{4}$$

Given the representation of the correct sequence of transitions $\boldsymbol{z}$ and the input phrase $\boldsymbol{s}$, the parser tries to minimize the negative conditional log-likelihood:

$$p(\boldsymbol{z}|\boldsymbol{s}) = -\sum_{t=1}^{|\boldsymbol{z}|} \log p(z_t|\boldsymbol{p}_t) \tag{5}$$

where $p(z_t|\boldsymbol{p}_t)$ is the probability of performing a specific transition z_t given the representation of the status $\boldsymbol{p}_t$. Network parameters are learned via the Stochastic Gradient Descent (SGD) optimization algorithm without mini-batches and applying a l_2 regularization factor. The adopted parsing algorithm can use a beam search.

In (Ballesteros, Dyer, and Smith 2015) a change to the representation of the equation (3) is proposed: the embedding vector $\boldsymbol{w}$ of the word is replaced by a representation based on the single characters from which the word is composed, through the application of a bidirectional LSTM. Given a word, we call $\overrightarrow{\boldsymbol{w}}$ the vector of the final state of the RNN that reads the characters from left to right and $\overleftarrow{\boldsymbol{w}}$ the final state of the RNN that reads the characters in the opposite direction. The representation of a token $\boldsymbol{x}$ in (3) is redefined as:

$$\boldsymbol{x} = \max\{0, \boldsymbol{V}[\overrightarrow{\boldsymbol{w}}; \overleftarrow{\boldsymbol{w}}; \boldsymbol{t}] + \boldsymbol{b}\} \tag{6}$$

This modification also introduces an additional transition to the arc-standard system that allows the production of dependency tree that are also non-projective.

(Kiperwasser and Goldberg 2016) follows the previous idea of representing the entire state of the parser using a two-way deep LSTM with k levels instead of the LSTM stack. The parser is developed in two versions: the first embodies a transition-based approach with an arc-hybrid and a dynamic oracle, while the second relies on a graph-based approach with an arc-factored model.

Given a sentence of n words $w_1, \ldots, w_n$ with the relative PoS tags $t_1, \ldots, t_n$, each word w_i and PoS tag t_i are associated with the embedding vectors $\boldsymbol{e}_i^w$ and $\boldsymbol{e}_i^t$, used to represent the input sequence $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and calculate the context $\boldsymbol{v}_i$ as:

$$\begin{aligned} \boldsymbol{x}_i &= [\boldsymbol{e}_i^w; \boldsymbol{e}_i^t] \\ \boldsymbol{v}_i &= \text{BiLSTM}(\boldsymbol{x}_i) \end{aligned} \tag{7}$$

The $\boldsymbol{v}_i$ features are used to obtain a representation ϕ that will be used as input for an MLP classifier, with only one hidden layer, which will calculate the score:

$$\text{MLP}(\phi(x)) = \boldsymbol{W}_2[\tanh(\boldsymbol{W}_2\phi(x) + \boldsymbol{b}_1)] + \boldsymbol{b}_2 \tag{8}$$

In the case of the transition-based model the MLP network learns the score of a transition given the $\phi(c)$ representation of a configuration c, obtained by combining the embedding vectors $\boldsymbol{v}_i$ associated with the first three words at the top of the stack and the first on the buffer:

$$\phi(c) = [\boldsymbol{v}_{\sigma_3}; \boldsymbol{v}_{\sigma_2}; \boldsymbol{v}_{\sigma_1}; \boldsymbol{v}_{\beta_1}] \tag{9}$$

In the case of the graph-based model the MLP classifier learns the score of each single arc (h, d) combining the embedding vector $\boldsymbol{v}_i$ of the head h and the dependent d:

$$\phi(h, d) = [\boldsymbol{v}_h; \boldsymbol{v}_d] \tag{10}$$

In the transition-based case they used a deterministic greedy parsing algorithm, while in the graph-based case the Eisner algorithm (described in (McDonald, Crammer, and Pereira 2006)) is used. Parameters are learned by minimizing the hinge loss using the Adam optimization algorithm.

In (Andor et al. 2016) a transition-based parser is proposed with an arc-standard system based on a network with a feed-forward architecture. Transition systems are characterized by a sequence of configurations $c_1, \ldots c_j$ with their transitions $t_1, \ldots, t_j$. The idea behind the model is to assume that there is a unique relationship between the sequence of transitions $t_1, \ldots, t_{j-1}$ and the state c_j. In other words, it is assumed that a state encodes the entire history of transitions. The goal is to learn, through the feed-forward network, the function $\mathrm{s}(t_{1:j-1}, t_j)$ which calculates the score of the next valid transition t_j for c, given the sequence of previous transitions $t_{1:j-1}$ (which is assumed to encode the configuration c). Let $\mathcal{T}_n$ be the set of valid transitions of length n, the model tries to find the solution to the optimization problem

$$\arg\max_{t_{1:n} \in \mathcal{T}_n} p_G(t_{1:n}) = \arg\max_{t_{1:n} \in \mathcal{T}_n} \sum_{j=1}^{n} \mathrm{s}(t_{1:j-1}, t_j) \tag{11}$$

where p_G defines a Conditional Random Field (CRF) probability distribution for the transition sequence $t_{1:n}$:

$$p_G(t_{1:n}) = \frac{\exp \sum_{j=1}^{n} \mathrm{s}(t_{1:j-1}, t_j)}{\sum_{t'_{1:n} \in \mathcal{T}_n} \exp \sum_{j=1}^{n} \mathrm{s}(t'_{1:j-1}, t'_j)} \tag{12}$$

To approximate the $\arg\max$ function a beam search is used to make learning easier. The parameters are learned by minimizing the CRF loss and using the SGD optimization algorithm with momentum.

(Cheng et al. 2016) proposes an arc-factored graph-based parser that makes use of a bidirectional RNN with attention mechanism to analyze the sentence. The representation of each word w_i is obtained starting from the combination of the one-hot vectors $\boldsymbol{e}_i$ of the word attributes to which an LReLU activation function is applied:

$$\boldsymbol{x}_i = \mathrm{LReLU}[\boldsymbol{P}(\boldsymbol{E}^{\mathrm{pos}}\boldsymbol{e}_i^{\mathrm{pos}} + \boldsymbol{E}^{\mathrm{form}}\boldsymbol{e}_i^{\mathrm{form}} + \boldsymbol{E}^{\mathrm{lemma}}\boldsymbol{e}_i^{\mathrm{lemma}} + \ldots)] \tag{13}$$

$$\text{LReLU}(x) = \begin{cases} 0.1x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

The kind of RNN cell used for the components is the Gated Recurrent Unit (GRU) (Cho et al. 2014) with the LReLU activation function instead of the $\tanh$. The state of the RNN is given by $\boldsymbol{h}_j = \text{GRU}(\boldsymbol{h}_{j-1}, \boldsymbol{x}_j)$, applied in both directions of the sentence so as to obtain $\boldsymbol{h}_j^l$ and $\boldsymbol{h}_j^r$. In this model one-hot encoding is preferred rather than dense encoding because, in this way, it is possible to avoid establishing the dimension of the embeddings.

The parser consists of three components: memory, right-left queries, and left-right queries. Given a sentence $S = w_0 w_1 \dots w_n$ the parser constructs the $\boldsymbol{m}_0, \boldsymbol{m}_1, \dots, \boldsymbol{m}_n$ elements of the memory component by combining the states obtained from the application of the RNN on the sentence in both directions, so $\boldsymbol{m}_j = [\boldsymbol{h}_j^l, \boldsymbol{h}_j^r]$.

Each $\boldsymbol{q}_t$ element of the query components is used to query the $\boldsymbol{m}_j$ elements of the memory component and return a score $a_{t,j}$. The score indicates the weight of the attention relative to the arc composed of the word in position t (dependent) and that in position j (head) for $t = 1, \dots, n$ and $j = 0, \dots, n$, and it is calculated as:

$$a_{t,j} = \text{softmax}(\boldsymbol{V} \tanh(\boldsymbol{C}\boldsymbol{m}_j + \boldsymbol{D}\boldsymbol{q}_t)) \tag{14}$$

The authors defined the *soft* embedding of the elements of the memory component, similar to the attention mechanism, as $\tilde{\boldsymbol{m}}_t = \sum_{j=1}^{n} a_{t,j}\boldsymbol{m}_j$ and calculate the elements of the query components as $\boldsymbol{q}_t = \text{GRU}(\boldsymbol{q}_{t-1}, [\tilde{\boldsymbol{m}}_t \boldsymbol{x}_t])$. The representations of the two query components, in both directions, are passed to a MLP network with a single hidden layer that returns the probability of the head-dependent relationships as:

$$\boldsymbol{y}_t = \text{softmax}(\boldsymbol{U}[\tilde{\boldsymbol{m}}_t^l; \tilde{\boldsymbol{m}}_t^r] + \boldsymbol{W}[\boldsymbol{q}_t^l; \boldsymbol{q}_t^r]) \tag{15}$$

where $y_{t,1}, \dots, y_{t,m}$ are the probabilities of all m possible relationships. Analyzing all the head-dependent combinations one can implicitly capture graph-based information of a higher order than the first, even using an arc-factored model. The parsing algorithm used is the one proposed by (Chu and Liu 1965)/(Edmonds 1967) and the parameters of the network are learned by minimizing the cross-entropy loss using the Adam optimization algorithm.

(Dozat and Manning 2017; Dozat, Qi, and Manning 2017) proposed an arc-factored graph-based parser based on the one of (Kiperwasser and Goldberg 2016). The difference is that instead of using a similar MLP network they used *biaffine* layers. In the original approach the $\boldsymbol{v}_i \in \mathbb{R}^d$ state of the Equation (7) is used to calculate the $\boldsymbol{s}_i$ score of an arc by a linear transformation $\boldsymbol{s}_i = \boldsymbol{W}\boldsymbol{v}_i + \boldsymbol{b}$ where $\boldsymbol{W} \in \mathbb{R}^{n \times d}$ and $\boldsymbol{b} \in \mathbb{R}^n$. In the biaffine architecture the transformation is obtained by introducing the product of two matrices $\boldsymbol{H}\boldsymbol{W} \in \mathbb{R}^{d \times d}$ instead of the single $\boldsymbol{W}$ matrix and the product $\boldsymbol{H}\boldsymbol{b} \in \mathbb{R}^d$ instead of bias $\boldsymbol{b}$. Before using biaffine transformation, the $\boldsymbol{v}_i$ state is given as input to different MLPs with three levels in order to eliminate irrelevant information. The biaffine transformation is applied to the state generated by the MLP application

in order to predict the $\hat{y}_i^{(\text{arc})}$ head of the word i as:

$$\begin{aligned} \boldsymbol{h}_i^{(\text{arc-dep})} &= \text{MLP}^{(\text{arc-dep})}(\boldsymbol{v}_i) \\ \boldsymbol{h}_i^{(\text{arc-head})} &= \text{MLP}^{(\text{arc-head})}(\boldsymbol{v}_i) \\ \boldsymbol{s}_i^{(\text{arc})} &= \boldsymbol{H}^{(\text{arc-head})}\boldsymbol{W}^{(\text{arc})}\boldsymbol{h}_i^{(\text{arc-dep})} + \boldsymbol{H}^{(\text{arc-head})}\boldsymbol{b}^{(\text{arc})} \\ \hat{y}_i^{(\text{arc})} &= \arg\max_j s_{ij}^{(\text{arc})} \end{aligned} \tag{16}$$

After predicting the head, this method selects the type of dependency $\hat{y}_i^{(\text{rel})}$ as follows:

$$\begin{aligned} \boldsymbol{h}_i^{(\text{rel-dep})} &= \text{MLP}^{(\text{rel-dep})}(\boldsymbol{v}_i) \\ \boldsymbol{h}_i^{(\text{rel-head})} &= \text{MLP}^{(\text{rel-head})}(\boldsymbol{v}_i) \\ \boldsymbol{s}_i^{(\text{rel})} &= \boldsymbol{h}_{\hat{y}_i^{(\text{arc})}}^{\top(\text{rel-head})}\boldsymbol{U}^{(\text{rel})}\boldsymbol{h}_i^{(\text{rel-dep})} + \boldsymbol{W}^{(\text{rel})}(\boldsymbol{h}_i^{(\text{rel-dep})} \oplus \boldsymbol{h}_{\hat{y}_i^{(\text{arc})}}^{(\text{rel-head})}) + \boldsymbol{b}^{(\text{rel})} \\ \hat{y}_i^{(\text{rel})} &= \arg\max_j s_{ij}^{(\text{rel})} \end{aligned} \tag{17}$$

The chosen MST algorithm for parsing is Chu-Liu/Edmonds. The parameters are learned by training jointly the two biaffine classifiers and minimizing the sum of the cross-entropy loss.

In (Shi, Huang, and Lee 2017; Shi et al. 2017) the authors proposed a parser that combines a compact representation of the features of three different parsing paradigms: the two arc-hybrid and arc-eager systems of the transition-based approach and the arc-factored model for the graph-based approach. For each sentence the embedding vectors of each word are derived, through the use of a bidirectional LSTM network, starting from the character-level representation described in (Ballesteros, Dyer, and Smith 2015). The graph-based model learns the score of the edges following the deep biaffine architecture from (Dozat and Manning 2017) previously discussed. The two transition-based models share the same configurations and the score is calculated using a deduction system that learns the joint model, with initial axiom $[0, 1]$ and final state $[0, n + 1]$. The sequence with the highest score of the deduction system leading to the configuration $[0, n + 1]$ constitutes the predicted sequence of transitions. The parameters are learned by minimizing the hinge loss through the Adam optimization algorithm.

In (Nguyen, Dras, and Johnson 2017), a neural network model is proposed that can jointly learn both the PoS tagging and the graph-based arc-factored dependency parsing. The basic idea is that the more the PoS tags are accurate the better the performance of the parsing improves and, vice versa, the structure of the dependency tree can solve some ambiguity of PoS tags.

Given an input sentence $w_1, \ldots, w_n$ consisting of n words, the embedding of each word w_i is built by concatenating the word embedding vector $\boldsymbol{e}_{w_i}$ to the vector $\boldsymbol{e}_{w_i}^c$ obtained by embedding its representation in characters as in (Ballesteros, Dyer, and Smith 2015):

$$\boldsymbol{e}_i = [\boldsymbol{e}_{w_i}, \boldsymbol{e}_{w_i}^c]. \tag{18}$$

The i-th word w_i is represented by the vector $\boldsymbol{v}_i$ obtained as $\boldsymbol{v}_i = \text{BiLSTM}(\boldsymbol{e}_i)$. The score of the edge with head w_i and dependent w_j is calculated as:

$$\text{score}(w_i, w_j) = \text{MLP}([\boldsymbol{v}_{w_i}; \boldsymbol{v}_{w_j}]) \tag{19}$$

The loss function L_{arc} used to learn the parameters of the parsing task is the hinge loss. For the PoS tagging the tag sequence is represented by the state obtained by applying a bidirectional LSTM on the tag sequence and the loss function $L_{\text{pos}}(\hat{\boldsymbol{t}}, \boldsymbol{t})$ to be minimized is the cross-entropy, where $\hat{\boldsymbol{t}}$ is the sequence of the predicted PoS tags and $\boldsymbol{t}$ the real one. The parser uses the Eisner MST parsing algorithm and the model parameters are learned by minimizing the sum of the two loss functions L_{pos} and L_{arc} through the Adam optimization algorithm.

It is worth mentioning the Italian dependency parser available in the package *spaCy*[2] based on DNN models. Unfortunately we could not include this parser into our evaluation because the available models have been trained on different corpora and in different experimental conditions and retraining new models would have required the production of a new parser using the spaCy API.

In Table 1 we summarised the fundamental characteristics for all the nine parsers considered in this study.

Table 1
All the neural parsers considered in this study with their fundamental features as well as their abbreviations used throughout the paper. In this table “Tb/Gb" means “Transition/Graph-based", “arc-s/h/f" means “arc-standard/hybrid/factored" and “cle" indicates the Chu-Liu/Edmonds algorithm.

Parser Reference	Abbrev.	Method	Parsing
(Chen and Manning 2014)	CM14	Tb: arc-s	greedy
(Ballesteros, Dyer, and Smith 2015)	BA15	Tb: arc-s	beam search
(Kiperwasser and Goldberg 2016)	KG16:T	Tb: arc-h	greedy
(Kiperwasser and Goldberg 2016)	KG16:G	Gb: arc-f	eisner
(Andor et al. 2016)	AN16	Tb: arc-s	beam search
(Cheng et al. 2016)	CH16	Gb: arc-f	cle
(Dozat and Manning 2017)	DM17	Gb: arc-f	cle
(Shi, Huang, and Lee 2017; Shi et al. 2017)	SH17	Tb: arc-h	greedy-eager
(Nguyen, Dras, and Johnson 2017)	NG17	Gb: arc-f	eisner

3. Datasets

We set-up each parser using the data from the Italian Universal Dependencies (UD) treebanks, UD Italian 2.1 (general texts) and UD Italian PoSTWITA 2.2 (tweets). These treebanks are annotated following the Universal Dependencies v2 format (Nivre et al. 2016). Before proceeding with the experiments, all the treebanks were converted from the CoNLL-U format to the CoNLL-X format (Buchholz and Marsi 2006), using the conversion script made available in the official UD repository. This conversion was necessary because some parsers were developed previously at the definition of the CoNLL-U format and accept only the CoNLL-X format.

3.1 UD Italian 2.1

This corpus contains generic domain texts[3]. The UD Italian treebank was obtained by converting the corpus ISDT (Italian Stanford Dependency Treebank) from the Stanford Dependencies annotation scheme to the Universal Dependencies scheme, as described in (Attardi, Saletti,

2 https://spacy.io/
3 https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-2515

and Simi 2015). The ISDT was released for the first time during the dependency parsing task at EVALITA 2014 and was derived from the conversion of the MIDT corpus (Merged Italian Dependency Treebank) (Bosco, Montemagni, and Simi 2013). MIDT is the result of the merger and conversion of two previously existing dependency treebanks for the Italian language:

- The Turin University Treebank (TUT) (Bosco et al. 2000);
- The ISST-TANL treebank, initially released as ISST-CoNLL for the CoNLL 2007 shared task and developed from the Italian Syntactic-Semantic Treebank (ISST) (Montemagni et al. 2003).

The whole corpus is composed of $13,884$ unique sentences, those already contained in the ISDT treebank plus other new sentences added after the conversion in the UD format. The subdivision of the entire corpus in Training, Development and Test sets is shown in Table 2.

Table 2
UD Italian 2.1 corpus splitting.

	Sentences	Words
Training set	12,838	270,703
Development set	564	11,908
Test set	482	10,417

3.2 UD Italian PoSTWITA 2.2

The second corpus we used contains social media texts[4]. The corpus, described in (Sanguinetti et al. 2018), consists of texts taken from the Twitter Italian platform. UD Italian PoSTWITA was created from a dataset used for the part-of-speech social media tagging in EVALITA 2016. The subdivision of the corpus is shown in Table 3.

Table 3
UD Italian PoSTWITA 2.2 corpus splitting.

	Sentences	Words
Training set	5,368	99,441
Development set	671	12,335
Test set	674	12,668

4. Neural Parsers Evaluation

For all parsers, we used the default settings for training, following the recommendation of the developers. Table 4 summarise the set up for each parser listing the values of each relevant hyperparameter.

4 https://github.com/UniversalDependencies/UD_Italian-PoSTWITA

Table 4
Values of the parsers hyperparameters: η is the learning rate; γ is the momentum factor; $\epsilon, \beta_1, \beta_2$ are the parameters for the Adam optimiser; b is the size of the mini-batch; p is the probability of dropout; h is the width of the MLP hidden levels; k is the number of RNN levels; l the size of the RNN; λ is the weight of the regularization term.

Parser	Hyperparameters
CM14	$\eta = 0.01,\ \epsilon = 10^{-6},\ b = 10.000,\ p = 0.5,\ h = 200,$ iter $= 20.000,\ \lambda = 10^{-8}$
BA15	$\eta = 0.1,\ k = 2,\ l = 100,$ iter $= 5.500,\ \lambda = 10^{-6}$
KG16	$\eta = 0.1,\ \epsilon = 10^{-8},\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ k = 2,$ $l = 125, h = 100,$ epochs $= 30,\ p = 0.5$
AN16	$\eta = 0.02,\ \gamma = 0.9,\ h_1 = h_2 = 512,\ b = 8,$ beam $= 16,$ epochs $= 10$
CH16	$\eta = 0.0004,\ h = 368,\ b = 1$
DM17	$\eta = 0.002,\ \beta_1 = \beta_2 = 0.9,\ \epsilon = 10^{-12},\ k = 3,$ $l = 300,\ h = 100, b = 5.000,$ iter $= 25.000,\ p = 0.33$
SH17	$\eta = 0.001,\ \beta_1 = 0.9,\ \beta_2 = 0.9999,\ \epsilon = 10^{-8},\ k = 2,$ $l = 256, h = 128,\ b = 50,$ epochs $= 20$
NG17	$\eta = 0.1,\ \epsilon = 10^{-8},\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ k = 2,\ l = 128,$ $h = 100,$ epochs $= 30$

The experiments were organized in three different setups:

- **setup0**: we trained and tested the parsing models on the generic Italian treebank (UD Italian 2.1 dataset). This setup will be useful to compare the performance of the parsers trained on generic texts with those obtained starting from texts coming from social media;
- **setup1**: we trained and tested the parsing models on a treebank in Italian language consisting solely of social media domain texts (UD Italian PoSTWITA 2.2);
- **setup2**: we trained the parsing models by joining the train and validation subsets of the the previous setups together and keeping the test set for setup1.

Evaluation results were obtained using the *Evaluation tool* software DEPENDABLE[5], described in (Choi, Tetreault, and Stent 2015), based on the standard evaluation script `eval.pl` of the CoNLL-X Shared Task.

After the influential paper from (Reimers and Gurevych 2017) it is clear to the community that reporting a single score for each DNN training session could be heavily affected by the system initialisation point and we should instead report the mean and standard deviation of various runs with the same setting in order to get a more accurate picture of the real systems performance and make more reliable comparisons between them.

For each parser we evaluated five different instances; counts include punctuation and report the mean and standard deviation of the values obtained in the different runs for each setup. On all models, the statistical significance level was calculated using DEPENDABLE and applying the McNemar statistical test.

5 Available online: `https://github.com/emorynlp/dependable`

Table 5 shows the parsers' performance on the test set for the three setups described above executing the training/validation/test cycle for 5 times.

EVALITA 2014 results reported the best score that establishes the state of the art for the dependency parsing in Italian: UAS 93.55% and LAS 88.76%[6]. The corpus used for setting up the systems participating to the task was the ISDT (Italian Stanford Dependency Treebank), the same corpus from which UD Italian was created. The UAS value for the best parser of experiments on UD Italian 2.1 (setup0) slightly exceeds the state of the art established in EVALITA 2014, while the LAS is 91.84%, 3.08% higher that the best parser in EVALITA 2014. However, it should be noted that despite being built on the same corpus, the UD Italian does not coincide exactly with the ISDT, so the comparison is not completely fair since the two treebanks have differences in their construction as an annotation scheme and sentences added and corrected in the UD Italian that do not appear in ISDT. In fact, in (Attardi, Saletti, and Simi 2015) it is shown how the change from ISDT to UD Italian 1.2 generates better performance on experiments conducted with statistical parsers. The results in the Attardi's paper are the only evaluations published in the literature on the UD Italian treebank, in version 1.2. Among the experiments conducted with statistical parsers the best result on UD Italian 1.2 is the Mate parser (Bohnet 2010; Bohnet and Kuhn 2012) with UAS evaluation 92.47% and LAS 90.22%. Even in this case we cannot make a complete comparison with the results obtained using UD Italian 2.1[7]. Notwithstanding that UD Italian 1.2 and UD Italian 2.1 are similar, they do not completely match; however, the evaluations obtained from the parser from (Dozat and Manning 2017) trained on UD Italian 2.1 are superior to those of Mate on UD Italian 1.2 in both UAS and LAS.

The results on setup1 are much lower when compared with those of setup0, but this is reasonable considering that the UD PoSTWITA treebank 2.2 is much smaller than the UD Italian 2.1 and that the syntactic analysis of tweets is linguistically more difficult than standard Italian. On the other hand, if we consider the models learned from the union of the two treebanks, we can see that there is still room for improvement with an increase of $\sim$1.7% in UAS and $\sim$2% in LAS, compared to the models learned using only the UD Italian PoSTWITA 2.2. From this we can deduce that to have good performance within a domain, specifically for social media texts, it is essential to use as much data as possible, including domain data, to train the parsing models. To further support this hypothesis, it can be noted that joining the two treebanks, UD Italian 2.1 and UD Italian PoSTWITA 2.2, we were able to get a nice performance improvement.

In any setup the DM17 parser exhibits the best performance, notably very high for general Italian. As we can expect, the performance on setup1 were much lower than that for setup0 due to the intrinsic difficulties of parsing tweets and to the scarcity of annotated tweets for training. Joining the two datasets in the setup2 allowed to get a relevant gain in parsing tweets even if we added out-of-domain data. For these reasons, for all the following experiments, we abandoned the setup1 because it seemed more relevant to use the joined data (setup2) and compare them to setup0.

5. An Ensemble of Neural Parsers

The DEPENDABLE tool in (Choi, Tetreault, and Stent 2015) is able to compute ensemble upper bound performance assuming that, given the parsers outputs, the best tree can be identified by an oracle "MACRO" (MA), or that the best arc can be identified by another oracle "MICRO" (mi). Table 6 shows that, by applying these oracles, we have plenty of space to improve the

6 http://www.evalita.it/2014/tasks/dep_par4IE

7 For changes between versions you can see the Changelog of the UD Italian corpus: https://github.com/UniversalDependencies/UD_Italian-ISDT

Table 5
Mean/standard deviation of UAS/LAS for each parser and for the three different setups by repeating the experiments 5 times. All the results are statistically significant ($p < 0.05$) and the best values are showed in boldface.

setup0				
	Valid. Ita		**Test Ita**	
	UAS	LAS	UAS	LAS
CM14	88.20/0.18	85.46/0.14	89.33/0.17	86.85/0.22
BA15	91.15/0.11	88.55/0.23	91.57/0.38	89.15/0.33
KG16:T	91.17/0.29	88.42/0.24	91.21/0.33	88.72/0.24
KG16:G	91.85/0.27	89.23/0.31	92.04/0.18	89.65/0.10
AN16	85.52/0.34	77.67/0.30	87.70/0.31	79.48/0.24
CH16	92.42/0.00	89.60/0.00	92.82/0.00	90.26/0.00
DM17	**93.37**/0.27	**91.37**/0.24	**93.72**/0.14	**91.84**/0.18
SH17	89.67/0.24	85.05/0.24	89.89/0.29	84.55/0.30
NG17	90.37/0.12	87.19/0.21	90.67/0.15	87.58/0.11
setup1				
	Valid. PoSTW		**Test PoSTW**	
	UAS	LAS	UAS	LAS
CM14	81.03/0.17	75.24/0.30	81.50/0.28	76.07/0.17
BA15	83.44/0.20	77.70/0.25	84.06/0.38	78.64/0.44
KG16:T	77.38/0.14	68.81/0.25	77.41/0.43	69.13/0.43
KG16:G	78.81/0.23	70.14/0.33	78.78/0.44	70.52/0.51
AN16	77.74/0.25	66.63/0.16	77.78/0.33	67.21/0.30
CH16	84.78/0.00	78.51/0.00	86.12/0.00	79.89/0.00
DM17	**85.01**/0.16	**78.80**/0.09	**86.26**/0.16	**80.40**/0.19
SH17	80.52/0.18	73.71/0.14	81.11/0.29	74.53/0.26
NG17	82.02/0.11	75.20/0.24	82.74/0.39	76.22/0.41
setup2				
	Valid. Ita+PoSTW		**Test PoSTW**	
	UAS	LAS	UAS	LAS
CM14	85.52/0.13	81.51/0.05	82.62/0.24	77.45/0.23
BA15	87.85/0.13	83.80/0.12	85.15/0.29	80.12/0.27
KG16:T	83.89/0.23	77.77/0.26	80.47/0.36	72.92/0.46
KG16:G	84.70/0.14	78.41/0.14	81.41/0.37	73.49/0.19
AN16	82.95/0.33	73.46/0.37	79.81/0.27	69.19/0.19
CH16	89.16/0.00	84.56/0.00	86.85/0.00	80.93/0.00
DM17	**89.72**/0.10	**85.85**/0.13	**87.22**/0.24	**81.65**/0.21
SH17	85.85/0.36	80.00/0.39	83.12/0.50	76.38/0.38
NG17	86.81/0.04	82.13/0.09	84.09/0.07	78.02/0.11

performance by building some kind of ensemble system able to cleverly choose the correct information from the different parsers outputs and combine them improving the final solution. This observation motivates our proposal.

Table 6
Results obtained by building an ensemble system based on the oracles mi e MA computed by the DEPENDABLE tool considering all parsers outputs.

	Validation		**Test**	
	UAS	LAS	UAS	LAS
setup0				
mi	98.30%	97.82%	98.08%	97.72%
MA	96.62%	95.10%	96.31%	94.82%
setup2				
mi	97.08%	96.02%	96.32%	94.73%
MA	94.62%	91.29%	93.27%	88.50%

To combine the parser outputs we tested three ensemble schemas proposed in literature: voting, reparsing and distilling. The next three subsections discuss these techniques applied to our problem and present the obtained results.

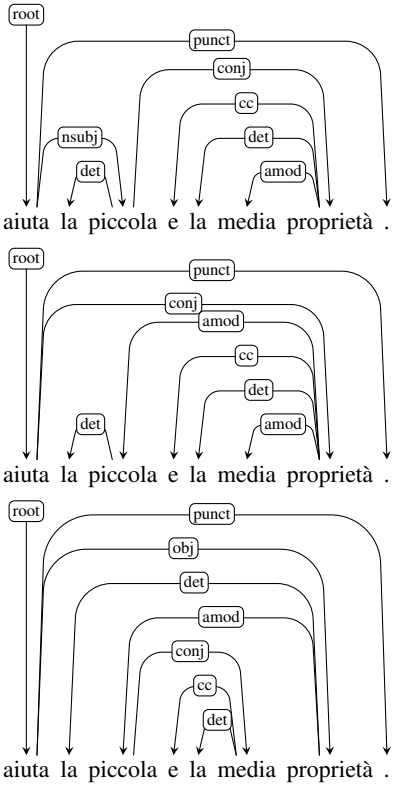
5.1 Voting

The voting technique was proposed for the first time in (Zeman and Žabokrtský 2005). Given the sentence $S = w_1 w_2 \dots w_n$, each of m basic parsers contributes to the defined ensemble by assigning one score to each candidate dependency relation (w_i, r, w_j), with $0 \leq i, j \leq n$ and $i \neq j$, where w_i is the dependent, w_j is the head and r is the type of the dependency relation. Dependency relations are contained among those available in the m trees predicted by parsers. In the example of Figure 1 we consider three dependency trees (shown in the upper left) produced by three different parsers. Starting from these, each individual distinct arc becomes an arc classified in the list of candidate arcs. All the candidate arcs, therefore, appear in at least one of the starting dependency trees. The $\mathbf{s_c}$ score is the number of votes of each candidate arc, calculated as the number of times that it is part of an individual dependency tree. After passing the list of candidates with their scores, for each word contained in the sentence the arc that has the maximum score is accepted and, in the event of a tie, the arc from the first parser is chosen. In the example of Figure 1 there is a tie for the word *proprietà* whose three candidate arcs all have $\mathbf{s_c} = 1$. This voting strategy is known as *majority*.

Since there is no restriction on the structure of the tree, the tree generated by the majority strategy has no guarantee to be well formed. As you can see in Figure 1, starting from three well-formed dependency trees a new dependency tree has been composed that is not well formed. To avoid this problem it is possible to use the strategy of *switching* which consists in checking if the final tree obtained through the majority strategy is well formed and, if not, replace it entirely with the dependency tree produced by the first parser. In order to build a feasible ensemble applying these techniques, we must consider at least the outputs of three different parsers.

As a preliminary analysis we try to capture and measure the diversity, or equivalently the similarity, of the available parsers. The basic idea is to understand how many times the parsers agree in predicting a certain relationship, this measure their agreement.

Consider two dependency trees $T = (V, A)$ and $\tilde{T} = (V, \tilde{A})$ for the same sentence $S = w_0 w_1 w_2 \dots w_n$ where the set of nodes V (the words in the sentence) are common to both trees and the set of edges A and $\tilde{A}$ can be different. It is possible to define the *agreement* between two



candidates			$\mathbf{s_c}$
aiuta	$\xleftarrow{\text{root}}$	ROOT	**3**
la	$\xleftarrow{\text{det}}$	piccola	**2**
la	$\xleftarrow{\text{det}}$	proprietà	1
piccola	$\xleftarrow{\text{nsubj}}$	aiuta	1
piccola	$\xleftarrow{\text{amod}}$	proprietà	**2**
e	$\xleftarrow{\text{cc}}$	proprietà	**2**
e	$\xleftarrow{\text{cc}}$	media	1
la	$\xleftarrow{\text{det}}$	proprietà	**2**
la	$\xleftarrow{\text{det}}$	media	1
media	$\xleftarrow{\text{amod}}$	proprietà	**2**
media	$\xleftarrow{\text{conj}}$	piccola	1
proprietà	$\xleftarrow{\text{conj}}$	piccola	**1**
proprietà	$\xleftarrow{\text{conj}}$	aiuta	1
proprietà	$\xleftarrow{\text{obj}}$	aiuta	1
.	$\xleftarrow{\text{punct}}$	aiuta	**3**

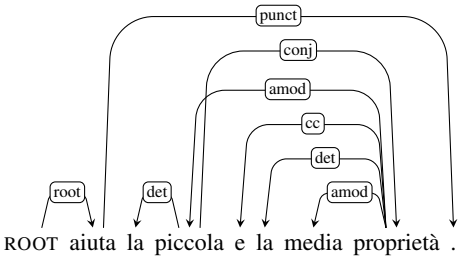


Figure 1
From the dependency trees obtained from three basic parsers (left) we establish the candidate arcs (right) from which we can generate the final dependency (below). The final tree is not well formed because it is not completely connected and presents the cycle *piccola* $\xrightarrow{\text{conj}}$ *proprietà* $\xrightarrow{\text{amod}}$ *piccola*.

dependency trees as:

$$\text{agreement}(T, \tilde{T}) = \frac{1}{|A|} \sum_{\substack{(w,r,h)\in A \\ (w,\tilde{r},\tilde{h})\in \tilde{A}}} \begin{cases} 1 & \text{if } h = \tilde{h} \text{ and } r = \tilde{r} \\ 0 & \text{otherwise} \end{cases} \tag{20}$$

where (w, r, h) and $(w, \tilde{r}, \tilde{h})$ are the dependency relations in the T and $\tilde{T}$ tree for each word $w \in \{w_1, \dots w_n\}$. From the definition it follows that the agreement is a symmetrical measure,

so we have that agreement$(T, \tilde{T})$ = agreement$(\tilde{T}, T)$. In other words the agreement calculates the percentage of the dependency relationships between two trees that have been labeled in the same way. By extending the definition of agreements on a set of trees computing the average on the whole treebank, we can calculate the agreement between the parsers using their predictions. In Table 7 the agreement for each pair of parsers using the models learned in setup0 is reported, while in Table 8 those in setup2.

Table 7
Agreement calculated on the development set starting from the predictions of the models learned in the setup0 (UD Italian 2.1).

	BA15	KG16:T	KG16:G	AN16	CH16	DM17	SH17	NG17
CM14	87.59%	87.88%	87.81%	81.19%	88.11%	88.30%	80.67%	85.74%
BA15		88.97%	90.26%	82.11%	89.85%	90.59%	82.85%	87.64%
KG16:T			91.25%	82.27%	90.25%	91.02%	82.61%	88.01%
KG16:G				82.85%	90.89%	91.92%	83.30%	88.81%
AN16					82.48%	83.56%	78.83%	84.32%
CH16						92.38%	83.98%	88.66%
DM17							85.38%	90.27%
SH17								83.70%

Table 8
Agreement calculated on the development set starting from the predictions of the models learned in the setup2 (UD Italian 2.1+PoSTWITA 2.2).

	BA15	KG16:T	KG16:G	AN16	CH16	DM17	SH17	NG17
CM14	83.79%	77.13%	76.95%	77.74%	83.73%	83.60%	74.33%	80.56%
BA15		78.34%	78.10%	78.65%	84.93%	85.58%	76.18%	82.98%
KG16:T			80.15%	77.46%	79.02%	79.85%	72.95%	80.22%
KG16:G				77.54%	79.02%	80.68%	72.90%	80.79%
AN16					78.76%	79.50%	72.34%	80.16%
CH16						87.19%	76.70%	83.66%
DM17							77.45%	84.60%
SH17								76.31%

The agreement depends largely on the performance of a parser because it is related to the LAS metric, which can be defined as agreement(T, G), where G is the gold standard dependency tree.

Even if the majority strategy could generate ill-formed parses, it will be considered to provide a comparison with the switching strategy. In some cases even having a ill-formed tree, but with a good accuracy, could be useful in some processes where a manual correction of the dependency tree is provided, as happens, for example, in the semi-automatic annotation of a treebank. For this reason we developed some experiments considering both the majority and the switching strategy.

For the experiments the following voter configurations were analyzed:

- (DM17 + CH16 + BA15) considers the three best parsers, which are (Dozat and Manning 2017), (Cheng et al. 2016) and (Ballesteros, Dyer, and Smith 2015);

- (AN16 + CM14 + SH17) considers the worst three parsers, which are (Andor et al. 2016), (Chen and Manning 2014) and (Shi, Huang, and Lee 2017);
- (DM17 + CM14 + SH17) considers the best parser, (Dozat and Manning 2017), combining it with those that have minor agreements, (Chen and Manning 2014) and (Shi, Huang, and Lee 2017);
- (AN17 + ALL) considers all parsers with (Andor et al. 2016) as the first parser;
- (DM17 + ALL) considers all parsers with (Dozat and Manning 2017) as the first parser.

Table 9 shows the performance of the ensembles built on the best results on validation set obtained in the 5 training/test cycles considering both setup0 and setup2. Table 11 reports the number of ill-formed trees for the majority strategy, while Table 10 reports the number of cases when the ensemble combination output differs from the baseline, including both labeled (L) and unlabeled (U) outputs. For the best results (DM17+ALL) the difference on setup0 and setup2 is about 4%.

The results of the voting approach reported in Table 9 shows that the majority strategy is slightly better than the switching strategy, although it must be taken into account that there might be ill-formed dependency trees for the former. The percentage of ill-formed trees on validation/test set vary from a minimum of 2% to a maximum of 8%. For this reasons the majority strategy should be used when it is followed by a manual correction phase. The switching strategy performs well if the first parser of voters is one of the best parsers, in fact the combinations AN16+ALL and AN16+CM14+SH17 have worst performance than the counterparts using the best parser (DM17) as the first voter. Overall, the highest performance is achieved using all parsers together with DM17 as the first voter. For setup0 the increases were +0.19% in UAS e +0.38% in LAS, while in setup2 are +0.92% in UAS e +2.47% in LAS with respect to the best single parser (again DM17).

5.2 Reparsing

A different approach for building ensemble systems, proposed in (Sagae and Lavie 2006), is *reparsing*. In this approach we try to overcome the limitation of the majority strategy, related to the possibility of generating trees that are not well formed, exploiting the same methodologies used in some parsing algorithms. This technique builds a directed graph with all the distinct candidate arcs provided by the single parsers, where the edge weights are obtained from a specific voting algorithm. Finally, classical parsing techniques are applied to the graph using an MST algorithm that generates a well-formed dependency tree.

We will consider two types of MST algorithms for the experiments: the first is the Chu-Liu/Edmonds algorithm, which searches the entire space of non-projective dependency trees, so the generated tree can also be non-projective. For this type of approach we can also use different techniques to assign the vote, which will influence the weight attributed to the arc in the graph. We will use the three different voting weighting methods proposed in (Hall et al. 2007):

- *w2*: equally weighted;
- *w3*: weighted according to the total labeled accuracy on the validation set;
- *w4*: weighted according to labeled accuracy per coarse grained PoS tag on the validation set.

Table 9
Results of ensembles using switching and majority approaches on the best models in setup0 and setup2. The baseline is defined by the best results of DM17.

setup0				
	Validation		**Test**	
Voters/Strategy	UAS	LAS	UAS	LAS
DM17+CH16+BA15/maj.	94.20%	92.27%	93.77%	92.13%
DM17+CH16+BA15/swi.	94.11%	92.16%	93.79%	92.14%
AN16+CM14+SH17/maj.	90.43%	87.96%	91.03%	88.47%
AN16+CM14+SH17/swi.	89.44%	86.77%	90.17%	87.43%
DM17+CM14+SH17/maj.	93.84%	92.03%	93.82%	92.27%
DM17+CM14+SH17/swi.	93.76%	91.94%	93.82%	92.25%
AN16+ALL/maj.	94.37%	92.65%	93.83%	92.27%
AN16+ALL/swi.	93.99%	92.15%	93.43%	91.73%
DM17+ALL/maj.	**94.42**%	**92.67**%	**93.94**%	**92.41**%
DM17+ALL/swi.	94.38%	92.60%	93.91%	92.37%
DM17 (baseline)	93.74%	91.66%	93.75%	92.03%
setup2				
	Validation		**Test**	
Voters/Strategy	UAS	LAS	UAS	LAS
DM17+CH16+BA15/maj.	90.57%	87.16%	88.21%	83.64%
DM17+CH16+BA15/swi.	90.51%	87.10%	88.13%	83.51%
AN16+CM14+SH17/maj.	86.90%	83.60%	84.09%	79.78%
AN16+CM14+SH17/swi.	86.01%	82.50%	82.58%	77.94%
DM17+CM14+SH17/maj.	90.35%	87.21%	88.07%	83.64%
DM17+CM14+SH17/swi.	90.27%	87.11%	87.99%	83.52%
AN16+ALL/maj.	90.30%	87.26%	88.36%	84.13%
AN16+ALL/swi.	89.70%	86.45%	87.46%	83.06%
DM17+ALL/maj.	90.64%	87.60%	**88.51**%	**84.42**%
DM17+ALL/swi.	**90.65**%	**87.62**%	88.50%	84.20%
DM17 (baseline)	89.82%	85.96%	87.59%	81.95%

As noted by (McDonald et al. 2005), searching the entire space of non-projective dependency trees can sometimes be counterproductive. Although some languages allow non-projective relationships, they are still mostly projective. So, looking through all the non-projective trees, we run the risk of finding dependency trees that are not desirable even if they are well formed. For example, the training set of the UD Italian corpus contains 564 non-projective dependency tree on the entire corpus of 12838 dependency tree ($\sim$4.4%). For this reason we consider a second MST algorithm that searches only the projective dependency tree, the Eisner algorithm.

For the reparsing experiments we will consider only the best three parsers (DM17 + CH16 + BA15) and ALL parsers (the order in this case is not important).

Table 12 shows the performance of the ensembles built on the best results on validation set obtained in the 5 training/test cycles considering both setup0 and setup2, while Table 13 reports the number of cases when the ensemble combination output differs from the baseline, including both labeled (L) and unlabeled (U) outputs.

Table 10
Numbers of cases when there is a different output between the ensemble systems, using switching and majority, and the baseline DM17.

setup0				
	Validation /11.908		**Test** /10.417	
Voters/Strategy	U	L	U	L
DM17+CH16+BA15/maj.	208	61	188	46
DM17+CH16+BA15/swi.	192	52	175	39
AN16+CM14+SH17/maj.	1.006	424	783	336
AN16+CM14+SH17/swi.	1.130	489	870	371
DM17+CM14+SH17/maj.	170	37	139	15
DM17+CM14+SH17/swi.	157	33	129	13
AN16+ALL/maj.	382	126	328	105
AN16+ALL/swi.	460	164	386	133
DM17+ALL/maj.	356	117	282	81
DM17+ALL/swi.	312	97	255	72
setup2				
	Validation /24.243		**Test** /12.668	
Voters/Strategy	U	L	U	L
DM17+CH16+BA15/maj.	597	219	470	213
DM17+CH16+BA15/swi.	521	185	394	172
AN16+CM14+SH17/maj.	2.757	1.329	1.805	941
AN16+CM14+SH17/swi.	2.976	1.429	1.986	1.033
DM17+CM14+SH17/maj.	490	140	337	93
DM17+CM14+SH17/swi.	453	121	300	73
AN16+ALL/maj.	1.377	624	897	440
AN16+ALL/swi.	1.610	741	1.063	534
DM17+ALL/maj.	1.156	502	784	378
DM17+ALL/swi.	920	374	614	280

Table 11
Number of ill-formed trees obtained by using the majority strategy for both setups.

	setup0		**setup2**	
Voters	**Valid.** /564	**Test** /482	**Valid.** /1235	**Test** /674
DM17+CH16+BA15	9	7	31	31
AN16+CM14+SH17	45	25	88	77
DM17+CM14+SH17	6	6	19	23
AN16+ALL	18	17	73	63
DM17+ALL	17	11	75	57

The results of the reparsing approach reported in Table 12 shows that the Chu-Liu/Edmonds algorithm is slightly better than the Eisner algorithm. In this case, the choice of the strategy that will be used depends on our decision of requesting the presence or the absence of non-projectivity. The percentage of non-projective dependency trees on valid./test set for Chu-Liu/Edmonds vary from a minimum of 7% to a maximum of 12% compared with the average for the Italian corpora of 4%. Overall, the highest performance are achieved using Chu-Liu/Edmonds algorithm. For setup0 the increases are +0.25% in UAS and +0.45% in LAS, while in setup2 are +0.77% in UAS and +2.30% in LAS with respect to the best single parser (DM17).

Table 12
Results of the proposed ensembles built by using reparsing approaches on the best models in setup0 and setup2. The baseline is again defined by the best results of DM17.

setup0				
	Validation		**Test**	
Voters/Strategy	UAS	LAS	UAS	LAS
DM17+CH16+BA15/cle-w2	93.82%	91.85%	93.54%	91.83%
DM17+CH16+BA15/cle-w3	93.89%	91.82%	93.78%	92.06%
DM17+CH16+BA15/cle-w4	94.20%	92.28%	93.72%	92.04%
DM17+CH16+BA15/eisner	94.05%	92.05%	93.46%	91.78%
ALL/cle-w2	**94.31**%	92.53%	93.85%	92.23%
ALL/cle-w3	94.16%	92.41%	**94.00**%	**92.48**%
ALL/cle-w4	94.29%	**92.58**%	93.95%	92.38%
ALL/eisner	**94.31**%	92.53%	93.95%	92.35%
DM17 (baseline)	93.74%	91.66%	93.75%	92.03%
setup2				
	Validation		**Test**	
Voters/Strategy	UAS	LAS	UAS	LAS
DM17+CH16+BA15/cle-w2	90.33%	86.95%	87.69%	83.31%
DM17+CH16+BA15/cle-w3	89.82%	85.96%	87.59%	81.95%
DM17+CH16+BA15/cle-w4	90.41%	86.99%	87.94%	83.32%
DM17+CH16+BA15/eisner	90.50%	87.05%	88.04%	83.51%
ALL/cle-w2	**90.52**%	**87.53**%	**88.36**%	**84.25**%
ALL/cle-w3	89.90%	86.75%	87.79%	83.54%
ALL/cle-w4	90.42%	87.46%	88.19%	84.11%
ALL/eisner	90.45%	87.41%	88.31%	84.08%
DM17 (baseline)	89.82%	85.96%	87.59%	81.95%

5.3 Distilling

Recently, in (Kuncoro et al. 2016), an approach has been proposed for the construction of a voting-based ensemble, later defined as *distilling*, which allows the training of a single model starting from different parsers independently trained. The technique consists of learning a single parsing model from a cost matrix based on the votes of the m parser, using a particular cost function that considers the uncertainty of the prediction. The basic idea is that the disagreement among parsers can be a signal that the relation in question is ambiguous. Training a distilling model takes a long time, but it has the advantage of creating a final model that does not need to query the individual parsers from which it was built. In the ensembles considered so far the

Table 13
Numbers of cases when there is a different output between the ensemble systems, using reparsing approaches, and the baseline DM17.

setup0				
	Validation /11.908		**Test** /10.417	
Voters/Strategy	U	L	U	L
DM17+CH16+BA15/cle-w2	360	129	307	90
DM17+CH16+BA15/cle-w3	96	0	89	1
DM17+CH16+BA15/cle-w4	267	76	247	52
DM17+CH16+BA15/eisner	375	130	327	103
ALL/cle-w2	400	131	333	103
ALL/cle-w3	351	108	299	79
ALL/cle-w4	383	126	307	87
ALL/eisner	411	133	333	106
setup2				
	Validation /24.243		**Test** /12.668	
Voters/Strategy	U	L	U	L
DM17+CH16+BA15/cle-w2	1.056	496	800	424
DM17+CH16+BA15/cle-w3	0	0	0	0
DM17+CH16+BA15/cle-w4	603	264	491	236
DM17+CH16+BA15/eisner	1.047	443	789	376
ALL/cle-w2	1.347	599	882	417
ALL/cle-w3	1.261	537	804	363
ALL/cle-w4	1.274	576	822	389
ALL/eisner	1.367	607	916	436

composition time of the final dependency tree is a function of the sum of the parsing times of the m parser used to build the ensemble. In the case of distilling, the model is created only once starting from the m parser that will no longer be used.

The results of the distilling strategy using all the available parsers are reported in Table 14: unlike the previous ensemble proposals this technique exhibits worse outcomes which score below the baseline (DM17).

Table 14
Results of distilling approach on the best models in setup0 and setup2. In brackets are reported the differences between the distilled models, built by considering all parsers, and the best results of DM17, as baseline.

Setup	UAS	LAS
setup0	92.50% (–1.25%)	89.93% (–2.10%)
setup2	86.73% (–0.86%)	81.39% (–0.56%)

6. Discussion and Conclusions

For a clear comparison between the different ensemble techniques tested in our experiments, the best results have been summarised in Table 15.

Table 15
The table summarises the best results obtained for the different ensemble strategies on the test set both for setup0 and setup2. Improvements with respect to the baseline (DM17) are shown in brackets. The largest improvements are marked in bold for both setups.

setup0		
strategy	UAS	LAS
DM17+ALL/*majority*	93.94% (+0.19%)	92.41% (+0.38%)
DM17+ALL/*switching*	93.91% (+0.16%)	92.37% (+0.34%)
ALL/*cle-w3*	94.00% (**+0.25%**)	92.48% (**+0.45%**)
ALL/*eisner*	93.95% (+0.20%)	92.35% (+0.32%)
ALL/*distilling*	92.50% (–1.25%)	89.93% (–2.10%)
setup2		
strategy	UAS	LAS
DM17+ALL/*majority*	88.51% (**+0.92%**)	84.42% (**+2.47%**)
DM17+ALL/*switching*	88.50% (+0.91%)	84.20% (+2.25%)
ALL/*cle-w2*	88.36% (+0.77%)	84.25% (+2.30%)
ALL/*eisner*	88.31% (+0.72%)	84.08% (+2.13%)
ALL/*distilling*	86.73% (–0.86%)	81.39% (–0.56%)

The distilling approach shows lower results than the baseline and, in general, lower than the other methods. The best results are obtained through the majority approach which brings with it the problem of not ensuring that the tree generated is well formed. The difference between the two setups is that in setup0 the improvements do not exceed 0.5%, while in setup2 they obtained a 2.5% of gain in LAS. As for reparsing methods, both in setup0 and in setup2 the Chu-Liu/Edmonds algorithm turns out to be slightly better than Eisner's procedure. All the techniques return more or less the same improvements, with oscillations of a few decimal points.

Therefore, the choice of the strategy is due, in part, to the properties we desire in the final tree. If we are not interested in the correctness of the tree structure, because it will be manually corrected, we can safely use the majority technique. If instead we want a correct tree and we have a parser exhibiting good performance, we could use the switching technique selecting it as the first parser. If the language can contain many non-projective structures, we might want to direct our choice on reparsing with the Chu-Liu/Edmonds algorithm or, if we prefer to get projective trees, on the Eisner algorithm. As far as distilling is concerned, it has not proved to be a good method to use for models obtained from different parsers but, as the author suggests, it could be useful when we consider several models obtained from different instances of the same parser.

We have studied the performance of some neural dependency parsers on generic and social media domain. Using the predictions of each single parser we combined the best outcomes to improve the performance in various ways. The ensemble models give an improvement of $\sim 1\%$ in UAS and $\sim 2.5\%$ in LAS in the mixed setup (2).

The improvement of LAS is, in most cases, at least twice the value of UAS. This could mean that ensemble models catch with better precision the type of dependency relations rather than head-dependent relations, since the candidates relations are taken from existing dependency

trees and not generated from scratch. This consideration needs, however, further studies in order to be verified.

All the proposed ensemble strategies, except for distilling, perform more or less in the same way, therefore the choice of the strategy to use is due, in part, to the properties that we want to obtain on the combined dependency tree.

Our proposal was inspired by the work of (Mazzei 2015). Unlike from his study, we use a larger set of state-of-the-art parsers, all based on neural networks, in order to gain more diversity among the models used in the ensembles; furthermore we have experimented the distilling strategy and the Eisner reparsing algorithm and we built ensembles on larger datasets using both generic and social media texts.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

References

Alicante, Anita, Cristina Bosco, Anna Corazza, and Alberto Lavelli. 2015. Evaluating Italian Parsing Across Syntactic Formalisms and Annotation Schemes. In Roberto Basili, Cristina Bosco, Rodolfo Delmonte, Alessandro Moschitti, and Maria Simi, editors, *Harmonization and Development of Resources and Tools for Italian Natural Language Processing within the PARLI Project*. Springer International Publishing, Cham, pages 135–159.

Andor, Daniel, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally Normalized Transition-Based Neural Networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany, August. ACL.

Attardi, Giuseppe, Simone Saletti, and Maria Simi. 2015. Evolution of Italian Treebank and Dependency Parsing towards Universal Dependencies. In *Proceedings of the Second Italian Conference on Computational Linguistics CLiC-it 2015: 3-4 December 2015, 2015*. Torino: Accademia University Press.

Ballesteros, Miguel, Chris Dyer, and Noah A. Smith. 2015. Improved Transition-based Parsing by Modeling Characters instead of Words with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 349–359, Lisbon, Portugal, September. ACL.

Bohnet, Bernd. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*, pages 89–97, Beijing, China, August 23-27.

Bohnet, Bernd and Jonas Kuhn. 2012. The Best of BothWorlds – A Graph-based Completion Model for Transition-based Parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 77–87, April 23-27, Avignon, France.

Bosco, Cristina, Felice Dell'Orletta, Simonetta Montemagni, Manuela Sanguinetti, and Maria Simi. 2014. The EVALITA 2014 Dependency Parsing task. In *Proceedings of the Fourth International Workshop EVALITA 2014*, pages 1–8, Pisa, Italy, December.

Bosco, Cristina, Vincenzo Lombardo, Daniela Vassallo, and Leonardo Lesmo. 2000. Building a Treebank for Italian: a Data-driven Annotation Schema. In *Proceedings of the Second International Conference on Language Resources and Evaluation, LREC 2000*, Athens, Greece, 31 May - June 2.

Bosco, Cristina and Alessandro Mazzei. 2011. The EVALITA 2011 Parsing Task. In *Working Notes of EVALITA 2011*. CELCT, Povo, Trento.

Bosco, Cristina, Simonetta Montemagni, and Maria Simi. 2013. Converting Italian Treebanks: Towards an Italian Stanford Dependency Treebank. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 61–69, Sofia, Bulgaria, August. ACL.

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, June. ACL.

Chen, Danqi and Christopher Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language*

Processing (EMNLP), pages 740–750, Doha, Qatar, October. ACL.
Cheng, Hao, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional Attention with Agreement for Dependency Parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2204–2214, Austin, Texas, November. ACL.
Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October, 25–29. Association for Computational Linguistics.
Choi, Jinho D., Joel Tetreault, and Amanda Stent. 2015. It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 387–396, Beijing, China, July. ACL.
Chu, Y.J. and T.H. Liu. 1965. On the Shortest Arborescence of a Directed Graph. *Science Sinica*, 14:1396–1400.
Dozat, Timothy and Christopher D. Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *Proceedings of the 2017 International Conference on Learning Representations*, Toulon, France, April 24-26.
Dozat, Timothy, Peng Qi, and Christopher D. Manning. 2017. Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada, August. ACL.
Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China, July. ACL.
Edmonds, Jack. 1967. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.
Hall, Johan, Jens Nilsson, Joakim Nivre, Gülsen Eryigit, Beáta Megyesi, Mattias Nilsson, and Markus Saers. 2007. Single Malt or Blended? A Study in Multilingual Parser Optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 933–939, Prague, Czech Republic, June. ACL.
Hochreiter, Sepp and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.
Kiperwasser, Eliyahu and Yoav Goldberg. 2016. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.
Kuncoro, Adhiguna, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an Ensemble of Greedy Dependency Parsers into One MST Parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753, Austin, Texas, November. ACL.
Lavelli, Alberto. 2013. An Ensemble Model for the EVALITA 2011 Dependency Parsing Task. In Bernardo Magnini, Francesco Cutugno, Mauro Falcone, and Emanuele Pianta, editors, *Evaluation of Natural Language and Speech Tools for Italian*, pages 30–36, Berlin, Heidelberg. Springer Berlin Heidelberg.
Lavelli, Alberto. 2014. Comparing State-of-the-art Dependency Parsers for the EVALITA 2014 Dependency Parsing Task. In *Proceedings of the Fourth International Workshop EVALITA 2014*, pages 15–20, Pisa, Italy, December.
Lavelli, Alberto. 2016. Comparing State-of-the-art Dependency Parsers on the Italian Stanford Dependency Treebank. In *Proceedings of the Third Italian Conference on Computational Linguistics (CLiC-it 2016)*, pages 173–178, Napoli, Italy, December.
Mazzei, Alessandro. 2015. Simple Voting Algorithms for Italian Parsing. In Roberto Basili, Cristina Bosco, Rodolfo Delmonte, Alessandro Moschitti, and Maria Simi, editors, *Harmonization and Development of Resources and Tools for Italian Natural Language Processing within the PARLI Project*. Springer International Publishing, Cham, pages 161–171.
McDonald, Ryan, Koby Crammer, and Fernando Pereira. 2006. Spanning Tree Methods for Discriminative Training of Dependency Parsers. Technical Report MS-CIS-06-11, University of Pennsylvania Department of Computer and Information Science, January.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada, October. ACL.

Montemagni, Simonetta, Francesco Barsotti, Marco Battista, Nicoletta Calzolari, Ornella Corazzari, Alessandro Lenci, Antonio Zampolli, Francesca Fanciulli, Maria Massetani, Remo Raffaelli, Roberto Basili, Maria Teresa Pazienza, Dario Saracino, Fabio Zanzotto, Nadia Mana, Fabio Pianesi, and Rodolfo Delmonte. 2003. Building the Italian Syntactic-Semantic Treebank. In Anne Abeillé, editor, *Treebanks: Building and Using Parsed Corpora*. Springer Netherlands, Dordrecht, pages 189–210.

Nguyen, Dat Quoc, Mark Dras, and Mark Johnson. 2017. A Novel Neural Network Model for Joint POS Tagging and Graph-based Dependency Parsing. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 134–142, Vancouver, Canada, August. ACL.

Nivre, Joakim, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Portorož, Slovenia, May.

Nivre, Joakim and Ryan McDonald. 2008. Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-HLT 2008*, pages 950–958, Columbus, Ohio, June. Association for Computational Linguistics.

Reimers, Nils and Iryna Gurevych. 2017. Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348, Copenhagen, Denmark, September. ACL.

Sagae, Kenji and Alon Lavie. 2006. Parser Combination by Reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York, New York, June. ACL.

Sanguinetti, Manuela, Cristina Bosco, Alberto Lavelli, Alessandro Mazzei, Oronzo Antonelli, and Fabio Tamburini. 2018. PoSTWITA-UD: an Italian Twitter Treebank in Universal Dependencies. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May, 7-12. European Language Resources Association (ELRA).

Shi, Tianze, Liang Huang, and Lillian Lee. 2017. Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23, Copenhagen, Denmark, September. ACL.

Shi, Tianze, Felix G. Wu, Xilun Chen, and Yao Cheng. 2017. Combining Global Models for Parsing Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 31–39, Vancouver, Canada, August. ACL.

Surdeanu, Mihai and Christopher D. Manning. 2010. Ensemble Models for Dependency Parsing: Cheap and Good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652, Los Angeles, California, June. ACL.

Zeman, Daniel and Zdeněk Žabokrtský. 2005. Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 171–178, Vancouver, British Columbia, October. ACL.