The DipInfo-UniTo System for SRST 2018

Valerio Basile
Dipartimento di Informatica
Università degli Studi di Torino
Corso Svizzera 185, 10153 Torino
valeriobasile@gmail.com

Alessandro Mazzei
Dipartimento di Informatica
Università degli Studi di Torino
Corso Svizzera 185, 10153 Torino
mazzei@di.unito.it

Abstract

This paper describes the system developed by the DipInfo-UniTo team to participate to the shallow track of the Surface Realization Shared Task 2018 (Mille et al., 2018). The system employs two separate neural networks with different architectures to predict the word ordering and the morphological inflection independently from each other. The UniTo realizer is language independent, and its simple architecture allowed it to be scored in the central part of the final ranking of the shared task.

1 Introduction

Natural Language Generation from formal structures, and in particular tree-like structures, has been approached with a variety of methods in the literature. For instance, SimpleNLG (Gatt and Reiter, 2009) takes as input a tree-like representation (a sort of *quasi*-syntactic tree enriched with a series of features) and produces an English sentence. SimpleNLG has is largely used in different NLG systems and has been ported to a number of different language (Italian among them (Mazzei et al., 2016)).

In the PhD thesis of Basile (2015), the generation process starts from a recursive representation of the semantics of a discourse (a Discourse Representation Structure, from Discourse Representation Theory) and it is carried out by transforming the original DRS into a directed graph (quite similar to a tree) aligned with the surface form at the word level. While the approach of Basile (2015) is aimed towards generation from abstract representations of meaning, in practice it is applicable to similar structures encoding information at a different level of abstraction, such as the trees that form the input of the present shared task.

We draw further inspiration from the aforementioned work in dividing the generation process into the word ordering prediction and morphology inflection generation. We follow a simplified approach by considering these two subtasks as independent from each other. We implement two modules based on neural networks that work in parallel, and whose output is later combined to produce the final surface form (cf. Figure 2).

In this paper we describe the the DipInfo-UniTo realizer (hencefort UniTO realizer) participating to the shallow track of the Surface Realization Shared Task 2018 (Mille et al., 2018).

In Section 2 we describe the system implemented from scratch for the word ordering subtask, and in Section 3 we briefly describe the deep learning-based approach that we used for the morphology inflection subtask. In Section 4 we describe the experimental pipelines used for training and testing the UniTo realizer and, moreover, we report the results on the test set. Finally, Section 5 closes the paper with some considerations and points to future developments.

2 Word Ordering

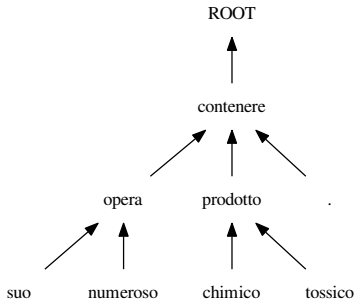
We adopted a *local ordering* approach to the task of predicting word ordering, as opposed to *global ordering*. We reformulate the problem of sentence-wise word ordering in terms of reordering its component subtrees, and subsequently recomposing the ordering of the words at the sentence level starting from the ordered subtrees.

The algorithm is composed of three steps: splitting the input unordered tree into single-level unordered subtrees (Section 2.1); predicting the local word order for each subtree (Section 2.2); recomposing the single-level ordered subtrees into a single multi-level ordered tree to obtain the global word order (Section 2.3).

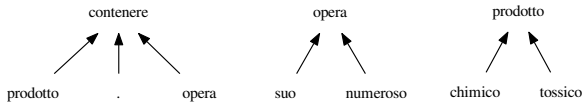
Proceedings of the First Workshop on Multilingual Surface Realisation, pages 65–71
Melbourne, Australia, July 19, 2018.

2.1 Extracting Lists of Items to Rank from the Input Trees

In the first step, we split the original unordered universal dependency multilevel tree into a number of single-level unordered trees, where each subtree is composed by a head (the root) and all its dependents (the children), in a way similar to (Bohnet et al., 2012).



(a) Tree corresponding to the sentence in the Italian training set *"Numerose sue opere contengono prodotti chimici tossici."*



(b) Three subtrees extracted from the main tree.

contenere prodotto . opera	opera suo opera	prodotto chimico tossico

(c) Three lists of items to order, corresponding to the three subtrees.

Figure 1: Illustration of the process of splitting the input tree into subtrees and extracting lists of items for learning to rank.

An example is shown in Figure 1: from the (unordered) tree representing the sentence "*Numerose sue opere contengono prodotti chimici tossici.*" (1a), each of its component subtrees (limited to one-level dependency) is considered separarately (1b). The head and the dependents of each subtree form a list of unordered items (1c). Crucially, we leverage the flat structure of the subtrees in order to extract structures that are suitable as input to the learning to rank algorithm in the next step of the process.

As a consequence of the design of our approach, in some cases the correct word order cannot be predicted. In particular, this is the case for non-projective tree structures, because the only realizations allowed by the formalism are those deriving from the dependency structure. For instance, the dependency tree representing the sentence *He gave a talk yesterday about generation* cannot be realized by the UniTo realizer since the tree itself is not projective. In this case, the best realization could be along the lines of *He gave yesterday a talk about generation*.

2.2 Supervised Learning to Rank

In the second step of the word ordering prediction algorithm, we predict the relative order of the head and the dependents of each subtree with a *learning to rank* approach. We employ the list-wise learning to rank algorithm *ListNet*, proposed in (Cao et al., 2007). The relatively small size of the lists of items to rank allows us to use a list-wise approach, as opposed to pairwise or point-wise approaches, while keeping the computation times manageable. Indeed, ListNet is a generalized version of the pairwise learning to rank algorithm RankNet (Burges et al., 2005).

ListNet uses a list-wise loss function based on *top one probability*, i.e., the probability of an element of being the first one in the ranking. The top one probability model approximates the *permutation probability* model that assigns a probability to each possible permutation of an ordered list. This approximation is necessary to keep the problem tractable by avoiding the exponential explosion of the number of permutations.

Formally, the top one probability of an object j is defined as

$$P_s(j) = \sum_{\pi(1)=j, \pi \in \Omega_n} P_s(\pi)$$

that is, the sum of the probabilities of all the possible permutations of n objects (denoted as Ω_n) where j is the first element. $s = (s_1, ..., s_n)$ is a given list of *scores*, i.e., the position of elements in the list. Considering two permutations of the same list y and z (for instance, the predicted order and the reference order) their distance is computed using cross entropy. The distance measure and the top one probabilities of the list elements are used in the loss function:

$$L(y, z) = -\sum_{j=1}^{n} P_y(j) log(P_z(j))$$

The list-wise loss function is plugged into a linear neural network model to provide a learning environment. ListNet takes as input a sequence

of ordered lists of feature vectors (the features are encoded as numeric vectors). The weights of the network are iteratively adjusted by computing a list-wise cost function that measure the distance between the reference ranking and the prediction of the model and passing its value to the gradient descent algorithm for optimization of the parameters.

We used an implementation of ListNet[1] that was previously applied in a surface realization task with a similar supervised setting (Basile, 2015). On top of the core ListNet algorithm, this implementation features a regularization parameter to prevent overfitting.

The choice of features for the supervised learning to rank component is a critical point of our solution. We use several word-level features encoded as one-hot vectors:

- The universal POS-tag.
- The treebank specific POS tag.
- The morphology features and the head-status of the word (head of the single-level tree vs. leaf).

Furthermore, we included word representations, differentiating between content words and function words:

- For open-class word lemmas (content words) we added to the feature vector the corresponding specific language embedding from the pre-trained multilingual model Polyglot (Al-Rfou' et al., 2013).
- Closed-class word lemmas (function words) are encoded as one-hot bags of words vectors.

An implementation of the feature encoding for the word ordering module of our architecture is available online[2].

2.3 From Local Order to Global Order

We reconstruct the global (i.e. sentence-level) order from the local order of the one-level trees under the hypothesis of projectivity. If the local reordering of the one-level tree T_1^h with root h and children $c_1...c_M$ produces an order of nodes $n_1 n_2 ... n_{M+1}$, the hypothesis of projectivity implies that in the global word order the position of all the children of the node n_j will be after the position of the node n_{j-1} and before the position of the node n_{j+1}. So, the node global order (O) of a k-level tree T_k^h rooted by the node h and with children $c_1...c_M$ can be rewritten formally in terms of the local order as:

$$O(T_k^h) = \begin{cases} h & \text{if } k=0 \\ O_{ln}(h, c_1, ..., c_M) & \text{if } k=1 \\ O_{ln}(h, O(T_{k-1}^{c_1}), ..., O(T_{k-1}^{c_M})) & \text{if } k>1 \end{cases}$$

where $O_{ln}(h, c_1, ..., c_M)$ is the permutation learned by the ListNet algorithm from the training set and parametrized over the feature set $F(h, c_1, ..., c_M)$ (cf. Section 2.2), that is

$$O_{ln}(h, c_1, ..., c_M) \stackrel{def}{=} P_{ListNet}^{F(h,c_1,...,c_M)}(h, c_1, ..., c_M)$$

3 Morphology Inflection

For the task of morphological inflection prediction, we implemented a module to work in parallel with the word order module described previously. This component of the system considers the morphology inflection as an alignment problem between characters that can be modeled with the sequence to sequence paradigm.

We used a deep neural network architecture based on a hard attention mechanism. The model has been recently introduced by Aharoni and Goldberg (2017) and showed state-of-the-art performance on several morphological inflection benchmarks. The model consists of a neural network in an encoder-decoder setting. However, at each step of the training, the model can either write a symbol to the output sequence, or move the attention pointer to the next state of the sequence. This mechanism is meant to model the natural monotonic alignment between the input and output sequences, while allowing the freedom to condition the output on the entire input sequence.

We trained the system[3] on the SRST training data set with no particular parameter tuning, that is, adopting an "off-the-shelf" approach. Moreover, we used a straight approach by using all the morphological features provided by the original UD treebank annotation and the dependency relation binding the word to its head. So, in the training pipeline (Figure 2), we

[1] https://github.com/valeriobasile/listnet

[2] https://github.com/alexmazzei/ud2ln

[3] An implementation of the model by Aharoni and Goldberg (2017) is freely available as https://github.com/roeeaharoni/morphological-reinflection

transform the training files into a set of structures $((lemma, features), form)$ in order to learn the neural inflectional model associating a $(lemma, features)$ to the corresponding $form$. The neural inflectional model is exploited in the test pipeline in order to compute the $form$ corresponding to a specific $(lemma, features)$ in the test file.

4 Experiments

Since our approach does not rely on language specific procedures or hand-made rules, we have initially planned to train the UniTo realizer for all the ten languages proposed by the SRST organizers. However, because of time constraints, we decided to focus on four specific languages first: English, Spanish, French and Italian (EN-ES-FR-IT). In particular, for English, French and Italian the learning time for word ordering and morphology inflection was around 36 and 24 hours respectively[4]. In contrast, for Spanish language, which has a considerable larger learning file, the learning time was approximatively doubled.

4.1 Pipelines

We designed two processing pipelines for the training phase and testing phase as depicted in Figure 2. We applied separately four times both the pipelines for the four tested languages EN-ES-FR-IT.

In the training pipeline, we created two distinct files starting from the UD treebank training files. The first file contains morphological information (that is $((lemma, features), form)$, cf. Section 3) and it is used to create the morphological inflection model by using the deep learning architecture described in Section 3. The second file contains the vector representation of the tree features (embeddings or function words, morphological features, etc., cf. Section 2.2) and it is used to create the word order model by using the linear neural network architecture described in Section 2.

In the testing pipeline, we created two distinct files starting from the test files provided from the organizers. Both files are created with the same procedures of the training pipelines. The first file was used to test the morphological neural model and to create a mapping from the pair lemma-features to the inflected form. The second file was used to test the word order neural model by providing the local word orders for the subtrees and the word order at the sentence level (cf. Section 2.3). In a subsequent step, the information from the morhological map and from the word ordered trees are merged into one single complete and CONLL compliant tree structure. Finally, the trees are detokenized (see 4.3) in order to produce the sentences that are submitted as the final output of the system.

[4]The experiments were run on two distinct multi-core PCs with GNU/Linux operating systems and GPU computing capabilities

4.2 Datasets

The rules of the shallow track for the SRST 2018 allowed to use any resource to train the surface realizers. However, in order to investigate about the syntactic information contained in the Universal Dependency format and its appropriateness for NLG tasks, we decided to use mostly information derived from the project Universal Dependency (Nivre et al., 2016). Indeed, the only exception regards the encoding of the open classes words in terms of language specific pre-compiled embeddings for the word order model (Al-Rfou' et al., 2013) (cf. Section 2.2)).

The task organizers provided ten training and ten development files derived from the version 2.1 of the UD dataset for the ten languages included in the shallow track. Indeed, they provided a modified versions of the original treebanks in which the information about the inflected word form was removed and, the original word order was replaced with a random order. Additionally, the organizers provided ten text files containing the sentences of the treebank in their original form.

However, we noted that the training files provided by the organizers had an unresolvable ambiguity in the case of a sentence containing the same lemma multiple times. As a consequence, we decided to use the original versions 2.1 of the treebank files since they contain both the gold word order and the inflected forms of the word. During the conversion of the dependency trees into a vector form (see Section 2), we ignored the information about word ordering and inflected forms.

For English, Spanish and French, we used the training files developed in the English, Spanish-AnCora, and French main UD treebanks respectively. In contrast, for Italian we built a new training file by merging together the training file of the Italian main UD treebank with the training files of the UD Italian treebanks Italian-PUD, Italian-

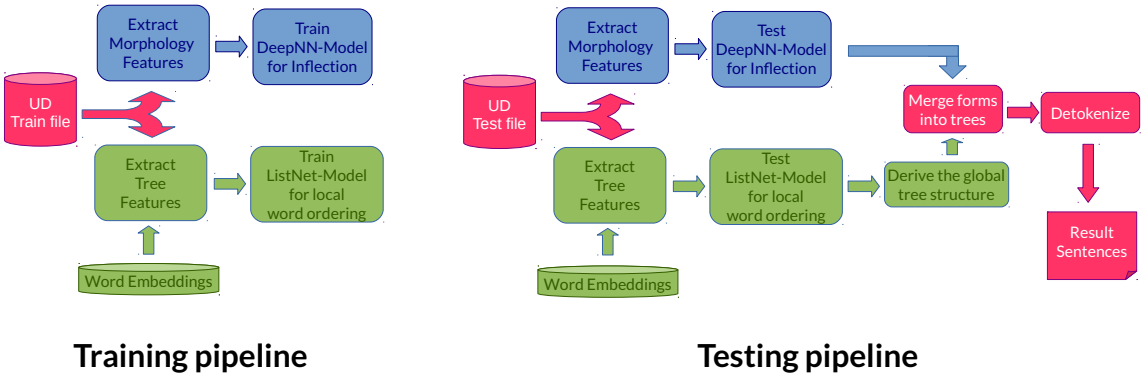


Figure 2: The training and testing pipelines.

ParTUT and Italian-PoSTWITA.

4.3 Detokenization

In order to produce the final result of the realization one needs to transform the UD tree produced by the UniTo realizer into a single string containing the sentence. Since the final goal of the task was to reproduce an output sentence close to the original sentence used by the treebanks developers, we needed to post-process the tree with additional two phases, that are *contraction* and *space removal*.

Contraction In this phase the sentence was modified in order to produce the contracted form for some specific multi-word constructions. In particular, for Spanish, French and Italian, there are two linguistic phenomena to account for, that are *articulated preposition* and *clitics*.

For instance, Italian provides a morphological mechanism to contract prepositions and articles into articulated prepositions. Indeed, there are 7 Italian simple prepositions (*di* (of), *a* (to), *da* (from), *in* (in), *con* (with), *su* (on)) which contract with the article. For instance, *la casa della zia* (the house of-the aunt) = la + casa + della (di [preposition] + la [definite article feminine singular]) + zia. In a similar way, clitics are pronouns which in Italian in particular cases can be included in the verb form, like in *Dammi la mela* (Give-me the apple) = Dammi (dai [verb] + me [pronoun]) + la + mela.

Since they are special case of multiwords, both articulated prepositions and clitics have a special annotation status into UD treebanks. Indeed, there is a line containing the multiword indexed with integer ranges, like *della* 3-4, and additional lines with single tokens annotation, like *di* 3 and *la* 4. We exploit this annotation by automatically extracting from the EN-ES-FR-IT UD treebanks all the regular expressions that are necessary to recompose the multiwords from the tokens (e.g. the PERL regular expression $s/\ di\ la\ /\ della\ /gi$). By using the UD treebanks training files of EN-ES-FR-IT we found 0[5], 923, 9, and 920 regular expressions respectively.

Space Removal Each language has additional specific rules for the treatment of space between words and punctuations. In order to treat this specific cases we used the detokenizer script provided in the moses project[6]: the detokenizer provides specific rules for English, French and Italian[7].

4.4 Results

In Table 1 we report the quantitative evaluation provided by shared task organizers of the surface realizer. With respect to the other teams, our results score in the middle-lower part of the final ranking: 6th out of 8 according to the BLEU and NE DIST score, and 5th out of 8 according to NIST.

[5]English language does not have neither articulated preposition and clitics.

[6]https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/detokenizer.perl

[7]As approximation, we used Italian configuration for Spanish too.

	EN	ES	FR	IT	Av.
BLUE	23.20	26.90	23.12	24.61	9.78
NE DIST	51.87	24.53	18.04	36.11	13.06
NIST	8.86	9.58	7.72	8.25	3.44

Table 1: The performance in terms of BLUE, DIST and NIST scores of the UniTo Realizer. The average is computed by considering the mean over the ten languages proposed for the shallow track.

The BLUE scores obtained suggest that the UniTo realizer have the same performances for all four languages. In contrast, the NE DIST results shows a better performance on the English language with respect to the other languages. Since BLEU and NIST give stronger weight to word order and lexical choice respectively (Zhang et al., 2004), these results suggest that our word order and morphology inflection modules equally contribute to the result. The difference in the NE DIST performance across languages has been observed in the other participants' results, and it could be due to the different morphological profile of the English with respect to the romance languages (ES-FR-IT).

5 Conclusion and Future Work

In this paper, we described the main features of the UniTo realizer, the system adopted by the DipInfo-UniTo team to participate to the shallow track of the Surface Realization Shared Task 2018. We described the two main components of the realizer: a linear neural network used to solve the word ordering subtask, and a deep neural network used to solve the morphological inflection subtask.

A number of possible improvements could be applied to the architecture. For instance, the morphological inflection could consider features deriving from sequences of words, i.e., having the word ordering module to inform the morphology module, or the other way around. Moreover, additional experiments are necessary in order to obtain the best tuning of the hyperparameters involved in the training phase.

References

Roee Aharoni and Yoav Goldberg. 2017. Morphological inflection generation with hard monotonic attention. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017*, pages 2004–2015.

Rami Al-Rfou', Bryan Perozzi, and Steven Skiena. 2013. Polyglot: Distributed word representations for multilingual nlp. In *CoNLL*, pages 183–192. ACL.

Valerio Basile. 2015. *From Logic to Language : Natural Language Generation from Logical Forms.* Ph.D. thesis, University of Groningen, Netherlands.

Bernd Bohnet, Anders Björkelund, Jonas Kuhn, Wolfgang Seeker, and Sina Zarrieß. 2012. Generating non-projective word order in statistical linearization. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 928–939. Association for Computational Linguistics.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, New York, NY, USA. ACM.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 129–136, New York, NY, USA. ACM.

Albert Gatt and Ehud Reiter. 2009. Simplenlg: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation*, ENLG '09, pages 90–93, Stroudsburg, PA, USA. Association for Computational Linguistics.

Alessandro Mazzei, Cristina Battaglino, and Cristina Bosco. 2016. Simplenlg-it: adapting simplenlg to italian. In *INLG 2016 - Proceedings of the Ninth International Natural Language Generation Conference, September 5-8, 2016, Edinburgh, UK*, pages 184–192.

Simon Mille, Anja Belz, Bernd Bohnet, Yvette Graham, Emily Pitler, and Leo Wanner. 2018. The First Multilingual Surface Realisation Shared Task (SR'18): Overview and Evaluation Results. In *Proceedings of the 1st Workshop on Multilingual Surface Realisation (MSR), 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1–10, Melbourne, Australia.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. 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 23-28, 2016.*

Ying Zhang, Stephan Vogel, and Alex Waibel. 2004. Interpreting bleu/nist scores: How much improvement do we need to have a better system. In *In Proceedings of Proceedings of Language Resources and Evaluation (LREC-2004*, pages 2051–2054.