

# Towards Multiple Tasks Integration with Reinforcement Learning

Timothée Bernard

National Institute of Advanced Industrial Science and Technology (AIST)

timothee.bernard@ens-lyon.org

## 1 Introduction

Historically, Natural Language Processing (NLP) systems have generally been built as sequential pipelines, where each module adds another layer of annotation, in order of (supposed) increasing complexity. For example, the system of [4], winner of the CoNLL-2005 Shared Task on Semantic Role Labeling (SRL) [2], starts with a syntactic analysis, followed by a *pruning* stage that collects argument candidates, an *argument identification* stage, an *argument classification* one and an inference stage that takes the final labelling decisions based on the classification probabilities output by the previous stage as well as a set of linguistic constraints. With the progress of neural networks, however, we have seen the apparition of state-of-the-art systems that completely bypass intermediate levels of analysis that were previously considered essential. For example, the systems of [16] and [6] performs respectively SRL and coreference resolution, both without referring to any explicit (morpho)syntactic information.

A well known problem of sequential systems is *error propagation*. Error propagation happens when an incorrect action makes a subsequent correct action unavailable, because the two are logically inconsistent, or when the action leads to a state that is unusual (unseen during the training process) with the effect of disrupting the system so that a subsequent correct action fails to be predicted even though it is logically consistent [7]. Additionally, traditional pipelines are conceptually limited by the fact that the flow of annotation between their different stages is *unidirectional*. For example, a system could perform Word Sense Disambiguation (WSD) before semantic parsing and in that case WSD classification can be used during semantic parsing, but not semantic parsing decisions for WSD. Alternatively, the system can preform semantic parsing first, and in that case it is the other way around. However, it is not clear that one ordering is the “right” one, optimal for all inputs. Intuitively, one might prefer do what is easy first, considering both tasks, in order to maximise the information available when taking the hardest

decisions. This kind of *easy-first strategy* has indeed been applied to various NLP problems, but only in single-task contexts [10, 3, 11, 15].

End-to-end systems with no intermediate level of analysis cannot, by definition, suffer from the first type of error propagation. However, by reducing the information in the training signal at their disposal, they might not leverage the full power of the neural networks they are usually based on. Indeed, by reintroducing syntactic dependencies in the training process, [12] managed to develop a state-of-the-art system for SRL that at the same time computes a good quality syntactic parse. The solution they implement, however, is very specific to the combination of tasks they consider. Furthermore, the interaction between the two types of information can only happen during the unique pass inside a neural network, where SRL and syntactic dependencies are only present under distributional forms, before they are independently discretised into symbolic structures.

In this work — currently under progress —, we propose to build a system that performs part-of-speech (POS) tagging, syntactic dependency parsing and semantic dependency parsing (SDP) in a non-sequential way. At each time step, the system can take actions related to any of these three annotation layers, with no order constraint. Each of these layers of annotation are fed as input for the next time steps, so they can fully interact with each other. Our model is inspired by the semantic dependency parser of [5], which was shown to develop an easy-first strategy (short semantic dependencies before longer ones) thanks to a Reinforcement Learning (RL) procedure. The most obvious difference between their work and ours is that we *fully integrate* different tasks into a unique system. This naturally leads to differences in architecture. The RL algorithm that we use is also slightly different from theirs.

## 2 Overview of the model

We use the data from task 18 of SemEval 2015 (broad-coverage semantic dependency parsing) [8].

In this work, we use only the English portion of the training data (which originates in the WSJ corpus) annotated with the DM semantic formalism. In this formalism, the semantic structure is a directed acyclic graph (each token may have zero, one or more parents) with labelled edges and some tokens can in addition be labelled as *top predicates*. We use the gold syntactic dependencies provided but ignore the annotated lemma information.

As [5]’s model, ours analyses an input sentence in multiple (time) steps. At each step  $t$ , the lower part of the network (Section 4), produces a vector representation for each token. This representation encodes, among other things, the current state of the syntactic and semantic structures. Then, the higher part of the network (Section 5) computes in parallel for each token  $k$  a distribution of probability  $\pi_{k,t}$  of actions. The set of possible actions includes exactly: one action per POS (TAG- $l$ ), one action per pair  $(i, l)$  where  $i$  is a token of the sentence and  $l$  a syntactic dependency label (SYN- $i-l$ ), idem with semantic dependency labels (SEM- $i-l$ ), a TOP\_PRED action and a NOTHING action.<sup>1</sup> Then, for each token  $k$  an action  $a_{k,t}$  is performed, sampled from the corresponding distribution  $\pi_{k,t}$ . The episode stops when all tokens simultaneously select the NOTHING action, or when a step limit is reached.

We start with a pre-training phase, during which the log-likelihood of the model on *goldish* sequences is minimised: for each token, we first build a sequence of (non-NOTHING) actions leading to the gold annotation, shuffle it, add a final NOTHING actions and pad all sequences with NOTHING actions so that they have the same length for all tokens. When the performance on SDP saturates, we start the RL phase described in the next section, during which the models try to find optimal sequences of actions (whose ordering we do not know).

### 3 Training

During the RL phase, we associate with each action  $a$  (for a given token  $k$  at a given time step  $t$ ) a scalar *reward*,  $r(a, k, t)$ , described at the end of this section. We train the model to maximise the expected sum of rewards,  $J = \mathbb{E}(R)$ , where  $E = \sum_t R_t$  and  $R_t = \sum_k r(a_{k,t}, k, t)$ .

To do so, we adapt here the *REINFORCE* algorithm [14] to our parallel processing. This simply consists in treating each token independently, except for the fact that we distribute future re-

<sup>1</sup>For the sake of simplicity, in the following section, 3, we will talk about top predicates as if they were the dependants of special virtual semantic dependencies.

wards to each token’s discounted reward equally.<sup>2</sup> More formally, given  $\gamma$  the *discount factor*, we define the discounted reward for token  $k$  at time  $t$  as  $G_{k,t} = r(a_{k,t}, k, t) + \sum_{i \geq 1} \gamma^i \frac{R_{t+i}}{n}$ , where  $n$  is the length of the sentence. The direction of the parameters update for a given episode is then:  $\sum_{t \geq 0, k} (G_{k,t} - b_{k,t}) \nabla_{\theta} \log \pi_{k,t}(a_{k,t})$  where  $b_{k,t}$  is the *baseline term*. The baseline term is not strictly necessary; its goal is to reduce the variance of the estimate of the gradient in order to speed the learning process [13, §13]. We use a *state value* baseline, which is trained by minimising its squared error with the observed return,  $L = \sum_{t \geq 0, k} (b(S_{k,t}) - G_{k,t})^2$ . The policy and baseline parameters updates are weighted with coefficient 0.67 and 0.33 respectively.

We now turn to the definition of the rewards. In this work, we give equal importance to all layers of annotation (i.e., POS tagging, syntactic parsing and semantic parsing). Let  $\#\text{pos}$ ,  $\#\text{synt}$  and  $\#\text{sdp}$  be, respectively, the number of tokens tagged with a POS (this is in fact the number of tokens, as they are all tagged), the number of syntactic dependencies and the number of semantic dependencies. Also, let  $N$  be the number of sentences in the training corpus. We then define  $r_{\text{pos}} = \frac{N}{\#\text{pos}}$ ,  $r_{\text{synt}} = \frac{N}{\#\text{synt}}$  and  $r_{\text{sdp}} = \frac{N}{\#\text{sdp}}$ . The reward of a given action is computed as the sum of the reward of its effects. Creating a correct (resp. incorrect) POS annotation corresponds to a  $r_{\text{pos}}$  (resp.  $-r_{\text{pos}}$ ) reward, and similarly with  $r_{\text{synt}}$  for syntactic dependencies and  $r_{\text{sem}}$  for semantic ones. In addition, as, in the current system, a new action overwrites any previous incompatible ones, when an annotation is removed, a reward equal to the opposite of its creation is added. For example, if at step  $t$  the action for token  $k$  is to add an incorrect tag, if at that point token  $k$  was correctly tagged, the corresponding reward is  $-2r_{\text{pos}}$  because an incorrect tag is added and a correct one is removed. Correctly annotating the entire training corpus then leads to a total reward of  $\#\text{pos} * r_{\text{pos}} + \#\text{synt} * r_{\text{synt}} + \#\text{sem} * r_{\text{sem}}$ , i.e.,  $3N$ . In other words, a reward of 3 on average per sentence.

### 4 Token representation

We first define a *base encoding* for each token, composed of 100-dimensional pre-trained GloVe word embeddings [9] concatenated to a set of *tagging* features including prefix and suffix embeddings. These base encodings serve as input to the *token representation module*, the architecture of which is depicted

<sup>2</sup>[5] do not discuss the definition of the discounted rewards because they only use the immediate local reward (discounting factor  $\gamma = 0$ ).

in Figure 1. The base encodings are fed independently to a BiLSTM, a syntax and a semantics encoder; the three outputs are concatenated before being fed to a final BiLSTM. The syntax and the semantics encoders are both *graph encoders* defined in the following way. Given a graph — defining a set of parents  $\text{parent}(i)$  for each token  $i$  along with  $l(j, i)$ , the label of the arc from  $j$  to  $i$  for  $j \in \text{parent}(i)$  — and writing  $u_i$  for the base encoding of token  $i$  and  $v_l$  for the embedding of label  $l$ , the graph encoding of the sentence is  $(w_i)_i = \text{BiLSTM}((w_i')_i)$  where  $w_i' = \sum_{j \in \text{parent}(i)} \text{Dense}([u_j, v_{l(j,i)}])$ .

The representation of each token  $k$ ,  $v_k$ , is sent (i) to a multilayer perceptron (MLP) that computes the state value  $b_{k,t}$  (the baseline term) and (ii) to different sub-networks that compute the log-probabilities of the actions (i.e., the probabilities before applying softmax), and are described in the next section.<sup>3</sup>

## 5 Action log-probabilities

The log-probability (or *score*) of each action is computed by one of three sub-networks. The first one is a simple MLP that returns the scores for TAG actions, TOP\_PRED and NOTHING. The second returns the scores for SYN actions: the score, for token  $i$ , to select token  $j$  as governor, for all possible dependency labels is given by  $\text{MLP}([v_i, v_j, \delta]) \in \mathbb{R}^{|L|}$ , where  $\delta$  (1 or 0) indicates whether  $j$  is currently the governor of  $i$  and  $|L|$  is the number of dependency labels. The last sub-network returns the scores for SEM actions in exactly the same way. The policy for a given token is then obtained by applying the softmax function to the vector of all corresponding scores.

## 6 Preliminary results and discussion

In this section, we start by presenting some preliminary results, before sketching a road-map for the continuation of the project. This project is currently under active work; all performance measures presented here have been computed on the *development set* and with our own algorithm (instead of the standard external tools). These numbers must then be considered with caution, as punctuation marks, for example, may not be handled (or set aside) correctly when computing syntactic or tagging performances.

We think the current results are promising. With a reasonable choice of hyperparameters, our model reliably trains to get above 91% labelled F1 for SDP.

<sup>3</sup>All BiLSTMs that we use are actually 2-layer BiLSTMs, and all MLPs have two hidden layers.

The current state-of-the-art, [5], gets 91.2% labelled F1 when not using lemma information (as we do, but on the test set). Concerning syntax, our model reaches 92% labelled recall.<sup>4</sup> To put this number in perspective, [12], who produce syntactic dependency trees as a by-product of SRL, report 91.87% LAS (on the test set) using GloVe embeddings (as we do). As for POS tagging, our model assigns a tag to more than 99.9% of the tokens, for a recall above 97%. To put this number in perspective, [12] (still with GloVe embeddings and on the test set), report an accuracy of 96.92% (the current state-of-the-art of the Penn Treebank is 97.96% [1]).

Like [5], we observe that our model naturally infers a kind of easy-first strategy: it tends to create shorter dependencies before longer ones. Figure 2 illustrates this for both syntax and semantics

The next obvious step for this project is to finish finding optimal hyperparameters and properly evaluate the trained models. Then, we would be interested in studying what happens when the rewards associated with tagging and syntactic parsing are lowered. We expect that doing so will encourage precision over recall, and possibly let the system generate syntactic structure that diverge from the gold annotation. In addition, we would like to study how the model trains on partially annotated data, i.e., when some training instances are annotated for syntax only while others for semantics only.

## References

- [1] Bernd Bohnet, Ryan McDonald, Gonalo Simões, Daniel Andor, Emily Pitler, and Joshua Maynez. Morphosyntactic Tagging with a Meta-BiLSTM Model over Context Sensitive Token Encodings. In *Proc. of ACL (long papers)*, pages 2642–2652, 2018.
- [2] Xavier Carreras and Lluís Màrquez. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *Proc. of CoNLL*, pages 152–164, 2005.
- [3] Yoav Goldberg and Michael Elhadad. An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *NAACL-HTL*, pages 742–750, 2010.
- [4] Peter Koomen, Vasin Punyakanok, Dan Roth, and Wentau Yih. Generalized Inference with Multiple Semantic Role Labeling Systems. In *Proc. of CoNLL*, pages 181–184, 2005.
- [5] Shuhei Kurita and Anders Søgaard. Multi-Task Semantic Dependency Parsing with Policy Gradient for Learning Easy-First Strategies. In *Proc. of ACL*, pages 2420–2430, 2019.
- [6] Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. End-to-end Neural Coreference Resolution. In *Proc. of EMNLP*, pages 188–197, 2017.

<sup>4</sup>As we currently do not force the system to compute complete syntactic trees, precision, recall and F1 could vary; in practice, the model assigns a syntactic head to more than 99.5% of tokens and the different measures are almost equal.

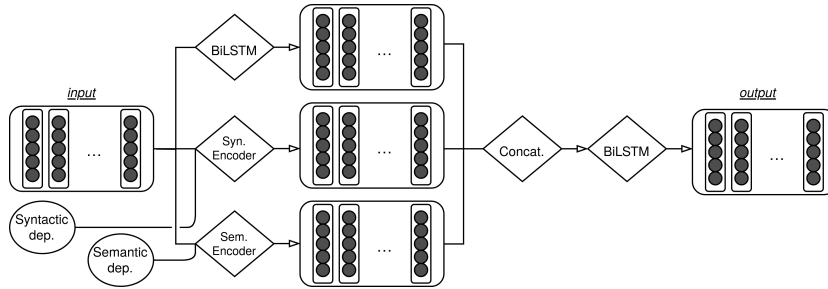


Figure 1: Architecture of the token representation module. The input is the sequence of *base encodings*. Each sequence is composed of one vector per token.

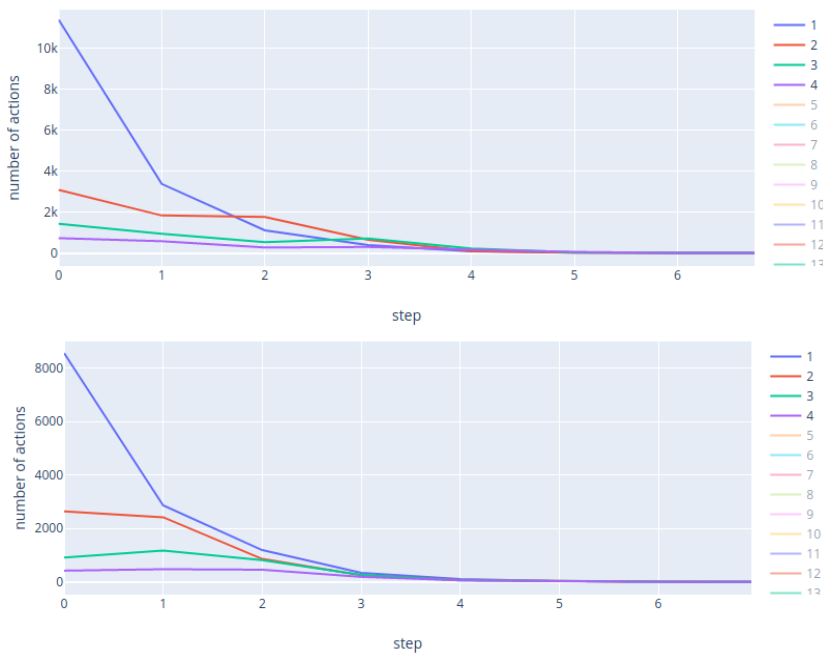


Figure 2: Number of syntactic (top)/semantic (bottom) dependencies created at each step (on the whole development set) for dependencies of length one to four.

- [7] Minh Lê and Antske Fokkens. Tackling Error Propagation through Reinforcement Learning: A Case of Greedy Dependency Parsing. In *Proc. of EACL (long papers)*, pages 677–687, 2017.
- [8] Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. SemEval 2015 Task 18: Broad-Coverage Semantic Dependency Parsing. In *Proc. of SemEval*, pages 915–926, 2015.
- [9] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *EMNLP*, pages 1532–1543, 2014.
- [10] Libin Shen, Giorgio Satta, and Aravind Joshi. Guided Learning for Bidirectional Sequence Classification. In *Proc. of ACL*, pages 760–767, 2007.
- [11] Veselin Stoyanov and Jason Eisner. Easy-first Coreference Resolution. In *Proc. of COLING*, pages 2519–2534, 2012.
- [12] Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. Linguistically-Informed Self-Attention for Semantic Role Labeling. In *Proc. of EMNLP*, pages 5027–5038, 2018.
- [13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning series. 2nd edition, 2018.
- [14] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.
- [15] Jun Xie, Chao Ma, Janardhan Rao Doppa, Prashanth Mannem, Xiaoli Fern, Thomas G. Dietterich, and Prasad Tadepalli. Learning Greedy Policies for the Easy-First Framework. In *AAAI*, 2015.
- [16] Jie Zhou and Wei Xu. End-to-end learning of semantic role labeling using recurrent neural networks. In *Proc. of ACL (long papers)*, pages 1127–1137, 2015.