Multiple Tasks Integration: Tagging, Syntactic and Semantic Parsing as a Single Task

Timothée Bernard*
LLF, Université de Paris, France
timothee.bernard@ens-lyon.org

Abstract

Departing from both sequential pipelines and monotask systems, we propose Multiple Tasks Integration (MTI), a multitask paradigm orthogonal to weight sharing. The essence of MTI is to process the input iteratively but concurrently at multiple levels of analysis, where each decision is based on all of the structures that are already inferred and free from usual ordering constraints. We illustrate MTI with a system that performs part-of-speech tagging, syntactic dependency parsing and semantic dependency parsing. We observe that both the use of reinforcement learning and the release from sequential constraints are beneficial to the quality of the syntactic and semantic parses. We also observe that our model adopts an easy-first strategy that consists, on average, of predicting shorter dependencies before longer ones, but that syntax is not always tackled before semantics.

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. Progress in neural networks has, however, led to the development of state-of-the-art systems that completely bypass intermediate levels of analysis that were previously considered essential. For example, the system of Zhou and Xu (2015) perform Semantic Role Labeling (SRL; Carreras and Màrquez, 2005) without referring to any explicit (morpho)syntactic information.

A well known problem of sequential systems is *error propagation*, which happens when an incorrect prediction at some point in the process leads to more incorrect predictions at a later stage. In traditional pipelines, one of the roots of error propagation lies in the fact that they feature a unidirectional flow of annotation between their different stages.

End-to-end systems with no intermediate level of analysis, in contrast, are protected against one form of error propagation. Since such systems are not contingent on possibly faulty levels of analysis, they are free from the interference they could cause. However, the absence of any intermediate decisions — symbolic traces of the system's computation — raises questions about generalisation ability and interpretability (Lipton, 2018). What is more, by reducing the information in the prediction and training signals at their disposal, they might not leverage the full power of the neural networks they are usually based on. Indeed, in their reintroduction of syntactic dependencies to the training process, Strubell et al. (2018) managed to develop a state-of-the-art system for SRL that at the same time computes good-quality syntactic parses.

Departing from these two kinds of architectures, we propose *Multiple Tasks Integration* (MTI). The principles of MTI are (i) to let the system take actions pertaining to different levels of analysis without constraining their order, and (ii), at any given point of the process, to feed all layers of annotation as input for the next predictions. The different tasks can therefore fully interact with each other as if they were a single task. Our main contribution lies in an illustration of these principles with a system that performs part-of-speech (POS) tagging, syntactic dependency parsing and semantic dependency parsing (SDP), on English data. We have chosen these specific tasks not only for their strong interdependence but also for their generality: many other tasks in NLP (e.g. SRL, coreference resolution, relation extraction) can be reduced to labelling or bi-lexical dependencies creation problems. We show that in this specific case, letting the system order freely its actions across all three tasks leads to better performance, and that this im-

*Work done while at AIST (Japan).

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 783–794
April 19 - 23, 2021.

provement concerns the syntactic layer as well as the semantic one, although to a lesser extent.

2 Related work

Fighting error propagation Lê and Fokkens (2017) identify two kinds of error propagation. First, error propagation happens when an incorrect action makes a subsequent correct action unavailable, because the two are logically inconsistent. Second, it also happens when an action leads to an unusual state (unseen during the training process), thus disrupting the system in such a way that a subsequent correct action is not taken even though both are logically compatible.

Since its apparition for speech recognition (Lowerre, 1976), *beam search* has become ubiquitous in NLP and artificial intelligence. Exploring multiple paths of actions in parallel instead of a single one mitigates the risk of error propagation of both kinds. An orthogonal approach consists in designing systems able to attain the gold annotation via multiple paths (usually involving different orderings of subsequences of actions) and training them to infer an *easy-first strategy*. Such a strategy corresponds to performing easy actions first so as to maximise the amount of reliable information available when having to tackle harder decisions. This idea has been applied to various NLP problems (Shen et al., 2007; Goldberg and Elhadad, 2010; Stoyanov and Eisner, 2012; Xie et al., 2015), but, to the extent of our knowledge, only in single-task contexts. Another promising research direction is the one of *iterative refinement*. The idea here is to use a baseline system in order to produce an initial output and to then correct it, possibly multiple times, with a refinement system that has access to both the previous output and the input of the problem. Iterative refinement techniques have been used for tasks such as machine translation (Lee et al., 2018) and SRL (Lyu et al., 2019).

Other techniques are designed to help to fight error propagation of the second kind, which focus on the training of the system to make it robust to its own mistakes. One possibility consists in training the system to predict the next best action in any state it find itself in, instead of simply staying on the (errorless) gold path. Doing so requires the possibility for a next best action to be determined, which, while trivial in some cases such as POS tagging, is not in others, in particular for structured prediction. Hence the introduction of the notion of *dynamic oracle* by Goldberg and Nivre (2012) for syntactic parsing. An alternative is *reinforcement learning* (RL; Sutton and Barto, 2018), which also trains the system from its actual trajectories, but depends instead on *rewards* defined for each action in the particular state it is taken. Note that in adequate settings, RL has been shown to lead to the emergence of easy-first strategies. This is the case in semantic parsing, as shown in the work of Kurita and Søgaard (2019), which we extend here to both POS tagging and syntactic parsing following our proposed Multiple Tasks Integration principles rather than implementing a traditional MultiTask Learning scheme.

MultiTask Learning (MTL) The main idea behind MTL is that one can often increase the performance of a given neural-based system by sharing some of its weights with other systems trained to perform other tasks (Caruana, 1997). Doing so tends to strengthen the training signal of the shared parts and to drive the model to develop richer and more relevant distributional representations. Refer to the texts of Ruder (2017) and Zhang and Yang (2018) for reviews of the wide NLP literature on the subject.

While MTL deals with tasks usually done rather independently, an alternative approach — which does not require weight sharing — is explored by Peng et al. (2018a). They work with pipelines where intermediate layers are expressed as solving a constrained maximisation problem, a setting for which they develop a technique, SPIGOT, to backpropagate gradient through argmax operations (which are discrete, hence non-differentiable). This allows for a lower module to be trained to predict structures used by a higher module so as to optimise the predictions of the latter. Note that the gain in performance obtained by introducing SPIGOT in a pipeline cannot necessarily be attributed to a reduction in error propagation: a lower module might end up generating less accurate yet more helpful predictions.

Joint syntactic-semantic parsing In 2008 and 2009, the CoNLL shared tasks were focused on the joint parsing of syntactic and SRL-based semantic dependencies (Surdeanu et al., 2008; Hajič et al., 2009). Most participants developed sequential systems that parse syntax before semantics, one notable exception being the proposal of Henderson et al. (2008), further developed by Henderson et al.

(2013) and on which the work of Swayamdipta et al. (2016) is based. These joint parsers are transition-based systems that alternate between sequences of syntactic actions and semantic ones. Both types of actions share the same symbolic structure, meaning that these systems allow for a strong interaction between syntax and semantics. While they exhibit no sequentiality globally over the two subtasks, the alternation of actions still enforces a strict ordering locally: once a new token is shifted from the input queue, it is processed syntactically before being processed semantically. This contrasts with the method adopted by Constant and Nivre (2016) for joint syntactic parsing and multiword expression detection, who compute, at each time step, a distribution of probability over actions pertaining to both tasks — as in our system. However, their linear (left-to-right) processing still imposes strong constraints on possible action orders.

More recently, Strubell et al. (2018) designed a model for SRL on the basis of the transformer architecture (Vaswani et al., 2017) in which one attention head is trained to attend to the syntactic head of the corresponding tokens. At inference time, the model is able to recover good-quality syntactic parses after (argmax) discretisation of the attention vectors. In this system and in stark contrast with our approach, the interaction between the two tasks can only take place during the single pass within the transformer network, where SRL and syntactic dependencies are only present under distributional forms (what we could call the "distributional soup of information"), before they are independently discretised into symbolic structures.

Semantic Dependency Parsing (SDP) The SDP task, introduced for the SemEval workshop (Oepen et al., 2014, 2015), represents semantic analysis as a directed acyclic graph (DAG) the nodes of which are tokens of the sentence. This means that each token may have zero or more parents. In addition, zero or more tokens in each sentence may be annotated as *top predicates*.

A wide range of techniques has been applied in the literature in order to tackle SDP. Peng et al. (2017, 2018b,a) propose graph-based constrained maximisation systems based on the AD^3 decoding algorithm (Martins et al., 2011). Dozat and Manning (2018) adapt a graph-based syntactic parser to DAG parsing while Wang et al. (2018) adapt a transition-based one. Zhang et al. (2019) present a very general sequence to graph transduction system. Wang et al. (2019) obtain excellent results by scoring not only the potential dependencies but also some of the potential pairs of dependencies. A second-order parser is thus defined, which relies on a (differentiable) mean field variational inference layer (or, alternatively, a loopy belief propagation one). The model closest to ours is the one of Kurita and Søgaard (2019), who see SDP as an iterated head-selection problem. They take inspiration from Zhang et al. (2017)'s system, which builds a syntactic tree by selecting a head for each token and correcting the corresponding dependencies with well-formedness heuristics. In this work, we see both syntactic and semantic parsing head-selection problems, that we tackle jointly with POS tagging.

3 Model

3.1 Overview

We define six types of *actions*. Relative to a token i, TAG-t corresponds to tagging i with POS tag t, SYN-j-l to creating a syntactic dependency labelled l from token j to i, ROOT to setting i as the (syntactic) root, SEM-j-l to creating a semantic dependency labelled l from token j to i, TOP_PRED to setting i as a (semantic) top predicate and, finally, HALT to doing nothing.

The inference process is summarised in Algorithm 1. At each time step s, the system first encodes the current state of the analysis into a sequence of one vector per token. These encodings contain information from the three different layers of annotation being built. Then, for each token i independently, its *policy* — a distribution of probability over all possible actions (TAG-t, SYN-j-l, etc.) —, $\pi_{i,s}$, is computed. From each policy $\pi_{i,s}$, an action $a_{i,s}$ is then selected and performed, thus enriching the annotation structure.[1] This means that, at any given step, one action per token is applied, as opposed to a single action for (typical) transition systems. At that point, either the next step starts, or, in case all tokens selected the HALT action, the analysis (or *episode*) stops.[2]

The system is designed to perform one action per token at each time step for computational reasons. Indeed, doing so allows to drastically reduce the number of steps required to analyse a sentence.

[1] During training, we sample from the distribution. At inference time, we take the argmax.

[2] If s reaches a limit $s_{\max}$, HALT is imposed to all tokens, leading to the end of the episode. $s_{\max}$ is set to 1.5 times the minimum number of steps required to correctly analyse any training instance (namely, 12).

Algorithm 1: Inference algorithm

input: A sentence $\mathbf{x}$.
Initialise empty POS tag, syntactic and semantic annotation structures:
$(s_{\text{tag}}, s_{\text{syn}}, s_{\text{sem}})$;
$s \longleftarrow 0$;
$continue \longleftarrow True$;
while $continue = True$ **do**
  $\mathbf{l} \longleftarrow encode(\mathbf{x}, s_{\text{tag}}, s_{\text{syn}}, s_{\text{sem}})$;
  for $i \in shuffle([\![1, |\mathbf{x}|]\!])$ **do**
    if $s < s_{max}$ **then**
      $\pi_i \longleftarrow policy(l_i)$;
      $a_i \longleftarrow select(\pi_i)$;
    else
      $a_i \longleftarrow$ HALT;
    apply a_i on $(s_{\text{tag}}, s_{\text{syn}}, s_{\text{sem}})$;
  if $\forall i$, $a_i =$ HALT **then**
    $continue \longleftarrow False$;
  $s \longleftarrow s + 1$;
return $(s_{tag}, s_{syn}, s_{sem})$

Note that this does not entail that analyses are three steps long, as (i) tokens can have more than a single semantic head and (ii) any token can decide to wait with a HALT action.

We do not ensure that the syntactic (resp. semantics) structure computed is a tree (resp. DAG); we leave the implementation of relevant heuristics as a subject of future work. We do impose several constraints, however: (i) a token can be annotated with only one POS tag, (ii) there can be at most one root, (iii) there can be no incoming syntactic dependency on a root, (iv) there can be at most one syntactic (resp. semantic) dependency from token j to i. Our strategy is to always perform the selected action, overwriting possible incompatible previous annotations.[3]

While POS tags, syntactic and semantic dependencies are used during training (see Section 4), only the raw tokens are required at inference time.

[3]This is also ensured within the actions selected for the different tokens of the sentence during a given time step, which, strictly speaking, we do not perform "in parallel", but in random order, as can be seen in Algorithm 1. As a result, if ROOT is selected for two tokens, one of them is set first as the root (deleting any incoming syntactic dependencies) — information that is immediately overwritten by setting the other one as the root.

3.2 Training

We typically train our model with a supervised pre-training phase followed by a reinforcement learning (RL) phase. During the pre-training phase, the log-likelihood of the model on *goldish* sequences of actions is maximised: for each token of a sentence, we generate a sequence of actions of minimum length leading to the gold annotation, randomly permute it (each time with a different permutation) and pad it with HALT so as to match the length of the longest sequence (over the sentence), before a final HALT is added. For example, for a token tagged with the POS tag of id 7, syntactically dependent (label of id 12) on the token 3, and that is neither a top predicate nor semantically dependent on any token, one of the two goldish sequences is

[SYN-3-12, TAG-7, HALT, HALT, HALT]

(if all tokens in the sentence can be fully annotated with a maximum of four actions). Using such sequences intuitively teaches the system to predict all annotations as quickly as possible, with no ordering preference. When the performance of the model on the SDP task saturates (early stopping on the development set), we switch to the RL phase, during which we expect the model to infer a good strategy as to how to order the different actions.

In relation to the reinforcement process, each action $a_{i,s}$ is associated with a reward $r(a_{i,s}, i, s)$. The role of RL is to train the model to maximise the expected sum of rewards $J = \mathbb{E}(R)$, where $R = \sum_s R_s$ and $R_s = \sum_i r(a_{i,s}, i, s)$. We compute the rewards in the following way. Let $\#\text{pos}$ be the number of POS annotations (which is also the number of tokens) in the training set, while N is the number of sentences. We then define $r_{\text{pos}} = \frac{N}{\#\text{pos}}$. The creation of a correct POS annotation and the suppression of an incorrect one (which can happen by overwriting) both correspond to a reward of r_{pos}, while the creation of an incorrect POS annotation and the suppression of a correct one correspond to a reward of $-r_{\text{pos}}$. Syntactic and semantic dependencies are treated similarly, with syntactic root (resp. semantic top predicate) annotations counted as a virtual syntactic (resp. semantic) dependencies. As a consequence, the construction of the full gold structure corresponds on average to a reward of 3 per sentence, equally balanced across the three layers. The reward associated to a given action is then computed as the sum of the reward of its effects, minus a small constant negative penalty

in the case of non HALT actions (set at a tenth of the average reward per token in the training set) aimed at discouraging the model from loitering.

The RL algorithm we use to optimise our model is a modification of REINFORCE (Williams, 1992). We adapt it to take into account the fact that multiple actions (one per token) are performed at each time step. To do so, we simply distribute future rewards to each token's discounted reward equally. More formally, given the *discount factor* γ, the discounted reward for token i at time step s is defined as

$$G_{i,s} = r(a_{i,s}, i, s) + \sum_{s' \geq 1} \gamma^{s'} \frac{R_{s+s'}}{n},$$

where n is the length of the sentence.[4] The direction of the parameters update for a given episode is then the one obtained from REINFORCE summed over all tokens:

$$\nabla_\theta L_p = \sum_{s \geq 0, i} (G_{i,s} - b_{i,s}) \nabla_\theta \log \pi_{i,s}(a_{i,s})$$

where $b_{i,s}$ is the *baseline term*.[5] We use a *state value* baseline, which is trained by minimising its squared error with the observed return:

$$\nabla_\theta L_b = \sum_{s \geq 0, i} 2 \left(\nabla_\theta b_{i,s} - G_{i,s}\right).$$

The policy and baseline parameters updates are weighted with coefficients 0.67 and 0.33 respectively. Note that we update the baseline term also during the pre-training phase, using the rewards obtained following the goldish sequences. Finally, we additionally maximise the entropy of each policy with a coefficient 0.002.

Optimisation is done using Adam (Kingma and Ba, 2015). The search for learning rates has been done manually, optimising the semantic F1 on the development set using the DM formalism.[6] We first found that a learning rate of 5.10^{-4} during pre-training gave satisfying results (the other optimiser parameters are left as set by default in TensorFlow, i.e. $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$), then that using a learning rate of 5.10^{-5} during the subsequent RL phase was a reasonable choice.

[4] Kurita and Søgaard (2019) do not discuss the definition of the discounted rewards and only use the immediate local reward (corresponding to a discounting factor $\gamma = 0$). We use here the fairly standard value $\gamma = 0.99$.

[5] 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 (Sutton and Barto, 2018, §13).

[6] See Section 4 about the different formalisms.

3.3 Token representations

We first define a *base encoding* for each token, composed of a 100-dimensional pre-trained GloVe word embedding (Pennington et al., 2014) concatenated to a POS tag embedding[7], a sum of prefix embeddings, a sum of suffix embeddings, a few other *tagging* features and two binary features indicating whether the token is currently predicted as the root or a semantic top predicate. (More detail is given in appendix A.) These base encodings are first sent through a dense ReLU layer before serving as input to the *token representation module*, which is composed of two layers of the architecture depicted in Figure 1. These layers contain a *syntax encoder* and a *semantics encoder*, the role of which are to linearise the partial syntactic and semantic structures respectively. These graph encoders are inspired by the work of Kurita and Søgaard (2019); the main difference lies in the use here of summation instead of a more powerful but also more expensive recurrent neural network.

Given a labelled graph — defining a set of parents $\mathit{parents}(i)$ and of children $\mathit{children}(i)$ for each token i along with $l(j,i)$, the label of the arc from j to i when it exists — and noting u_i for the input corresponding to token i and v_l for the embedding of label l, we define the *downward encoding* of i as $w_i^d = \sum_{j \in \mathit{children}(i)} \mathit{Dense}([u_j, v_{l(i,j)}])$ (where *Dense* represents a dense ReLU layer). Similarly, the *upward encoding* of i is defined as $w_i^u = \sum_{j \in \mathit{parents}(i)} \mathit{Dense}([u_j, v_{l(j,i)}])$ (where *Dense* represents another dense ReLU layer). Finally, the *bothward encoding* of i is defined as the concatenation of the two previous vectors: $w_i = [w_i^d, w_i^u]$. For the output of the syntax encoder, we use the bothward encodings of the syntactic graph while for the output of the semantic encoder we use only the upward encodings of the semantic graph.

The output of the second layer of the token representation module is sent (i) to a multilayer perceptron (MLP) that computes the state value $b_{i,t}$ (the baseline term) and (ii) to different sub-networks that compute the logits of the actions (i.e. the values from which the probabilities are obtained by applying softmax), which are described in the next section.[8]

[7] The corresponding vocabulary includes a "not tagged yet" embedding.

[8] All BiLSTMs that we use are actually 2-layer BiLSTMs, and all MLPs have two hidden layers.

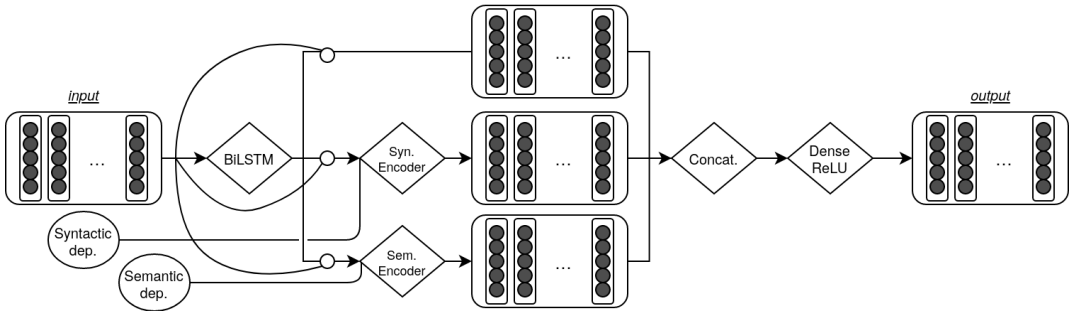

Figure 1: One layer of the token representation module. Each sequence is composed of one vector per token. Each of the circles directly linked to the output of the BiLSTM is a gate parametrised by a scalar α and represents the function $(v_1, v_2) \mapsto \sigma(\alpha)v_1 + (1 - \sigma(\alpha))v_2$. See the text for the definition of the syntax and semantics encoder.

3.4 Action logits

The logit of each action is computed by one of three sub-networks. The first one is an MLP that returns the logits for TAG-t actions, ROOT, TOP_PRED and HALT. The second returns the logits for SYN-j-l actions: the logit, for token i, to select token j as governor, for all possible dependency labels is given by $\text{MLP}([v_i, v_j, v'_{i,j}]) \in \mathbb{R}^{|L|}$, where $|L|$ is the number of dependency labels and $v'_{i,j} \in \mathbb{B}^{|L|+1}$ is a one-hot vector indicating whether j is currently governor of i and if so, what is the label of the corresponding dependency. The last sub-network returns the logits for SEM-j-l actions in exactly the same way. The policy is then obtained by applying the softmax function to the concatenation of the output of these different sub-networks.

4 Experiments

To test whether MTI is a viable paradigm and determine the impact of RL, in this section we test the model described above along with three variants. These four models can be seen as combinations of two binary traits. The first trait pertains to the ordering of the actions: *sequential* models simulate sequential pipelines, while for *free* models, as above, no particular constraint is imposed. A sequential model can only select TAG-t actions during the first step of an episode, only SYN-j-l and ROOT during the second and only SEM-j-l, TOP_PRED and HALT afterwards. This is ensured by using as policy for each token the vector obtained by normalising a masked version of its usual policy.[9] The second trait pertains to the learning process: *supervised* models are only trained in the pre-training phase while *RL* models, as above, are further trained using reinforcement learning.

4.1 Data

We use data from the SemEval 2015 Task 18 (Oepen et al., 2015), which provides POS tags, lemmas, syntactic parses and semantic parses for English, Chinese and Czech texts. Concerning English, the same data is annotated with three distinct formalisms: DELPH-IN MRS-Derived Bi-Lexical Dependencies (DM), Enju Predicate–Argument Structures (PAS) and Prague Semantic Dependencies (PSD). They all represent a semantic analysis as a bi-lexical dependency DAG. We use here the standard split of the English data. The train set is composed of 33,964 sentences from sections 0–19 of the WSJ corpus, the development set of 1,692 from section 20 and the (in-domain) test set of 1,410 from section 21. During training, for simplicity, we always use the gold (morpho)syntax annotation provided under the form of Stanford Basic dependencies derived from the PTB, even when working with the DM formalism (which comes with different annotations).[10]

4.2 Semantics

Table 1 shows the average F1 for labelled semantic dependencies over the three formalisms in each setting. Three randomly initialised runs are used for each formalism, with each performance computed using the script provided by the shared task. Everywhere in this text, F1 are *micro*-F1 (i.e. computed over the whole dataset instead of at the sentence level). We first observe that RL models perform better than their supervised-only counterparts, a

[9] The goldish sequences used during pre-training are modified accordingly. The ordering of the selection of the possibly multiple semantic parents of any given token is still random.

[10] All data and evaluation scripts can be found on the shared task's website: http://alt.qcri.org/semeval2015/task18/.

(sem.)	sequential	free	Δ
supervised	84.9 (86.4)	85.7 (87.2)	0.8 (0.8)
RL	86.5 (87.9)	87.3 (88.5)	0.8 (0.6)
Δ	1.6 (1.5)	1.6 (1.3)	

Table 1: Average of the F1 over the three semantic formalism on the in-domain test set. Scores on the development set are given in brackets.

(synt.)	sequential	free	Δ
supervised	91.0 (91.1)	91.1 (91.4)	0.2 (0.3)
RL	91.8 (91.9)	92.3 (92.2)	0.5 (0.4)
Δ	0.8 (0.8)	1.2 (0.8)	

Table 2: Average syntactic F1 on the in-domain test set. Scores on the development set are given in brackets.

(tag.)	sequential	free	Δ
supervised	96.9 (97.1)	96.9 (97.1)	0.0 (0.0)
RL	97.1 (97.2)	97.2 (97.2)	0.0 (0.0)
Δ	0.2 (0.1)	0.2 (0.1)	

Table 3: Average tagging F1 on the in-domain test set. Scores on the development set are given in brackets.

result consistent with the findings of Kurita and Søgaard (2019). This indicates that RL allows the model to infer a good ordering strategy (limited to the ordering of the semantic actions in the sequential regime) and/or trains it to be more robust to its own mistakes (an effect similar to that of dynamic oracles). Second, we observe that free models perform better than their sequential counterparts. This indicates, as we had hypothesised, that a strict ordering of actions following a traditional hierarchy of levels of analysis is not optimal and that MTI is a potentially powerful form of MTL (note that the sequential models here implement a relatively traditional form of MTL). Finally, we see that these two effects are additive.

4.3 Syntax and POS tagging

Let us turn to the performance of our model on the syntactic parsing and POS tagging tasks in the four settings. Table 2 shows the F1 for labelled syntactic dependencies[11], averaged over all nine corresponding runs. Following Chen and Manning (2014)'s recommendation, punctuation is excluded from the evaluation. What we observe is that while the use of RL and (more particularly) non-sequentiality have a small impact on syntactic parsing, it is still noticeable. All scores on the test set are statistically different according to Pitman's permutation test, with a p-value under 10^{-3}.[12] The combination of RL with free-ordering appears a particularly good combination.

As shown in Table 3, the use of RL also has a statistically significant impact on POS tagging, although of very low magnitude. Free-ordering, however, has no perceptible effect (the corresponding deltas are in fact positive but strictly smaller than 0.05 and the distributions they compare are not distinguishable).

Note that while we do not use the same split of the WSJ Corpus as them, a comparison with the 91.87 LAS on syntax and 96.92 accuracy on POS tagging obtained by the model of Strubell et al. (2018) — which is also trained to perform these tasks jointly with a form of semantic parsing — indicates that our free+RL model, with 92.3 F1 on syntax and 97.2 on POS tagging, performs at least reasonably well.

4.4 Inferred strategies

In order to better understand why our MTI paradigm leads to better performance not only on the syntactic task but also on the semantic one, we now perform a brief study of the ordering strategy inferred by our free+RL model. We focus here on the SYN-j-l and SEM-j-l actions and look at the average time step at which they are created as a function of their length.[13] Figure 2 shows that for the PAS formalism, on average, the system creates (i) syntactic dependencies before semantic ones and (ii) short dependencies before long ones.[14] This tends to indicate that the model follows an easy-first strategy and that, as often assumed, it is preferable

[11] As we do not constrain the model to produce complete syntactic trees, the natural measure here is indeed F1. In practice, however, the *parsing rate* (the ratio of tokens that either have an incoming dependency or are predicted to be the root) is always higher that 99%, meaning that recall and precision are almost equal and that F1 is very similar to a labelled attachment score (LAS). A similar remark applies to POS tagging.

[12] We use the permutation test to check whether the distributions of the scores in each setting (nine runs) are distinct from each other. See the work of Dror et al. (2018) for an NLP-oriented discussion of statistical significance.

[13] Sometimes, though rarely, does the model select the same action multiple times (for the same token) over a single episode. We filter out useless actions, that is to say, the ones that do not modify the structure when they are applied.

[14] Each figure mentioned in this paragraph corresponds to a randomly picked run. The behaviour observed is very consistent across the different runs for a given semantic formalism.

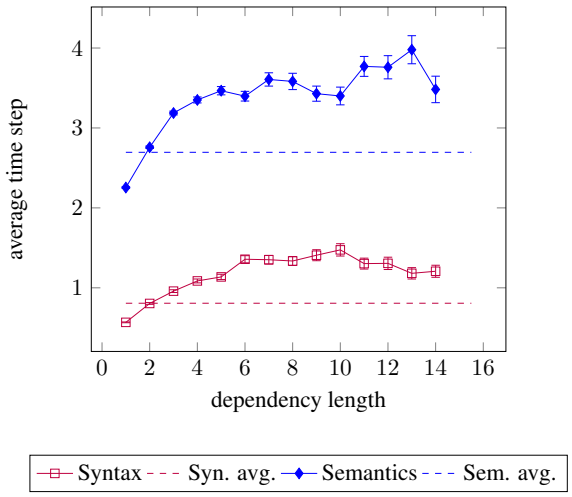

Figure 2: PAS semantic formalism: average time step (from 0) at which syntactic and semantic dependencies are created as a function of their length. Error bars indicate the standard error of the mean.

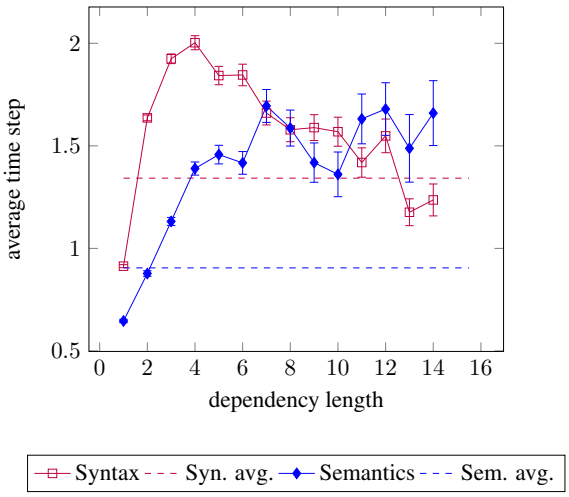

Figure 3: DM semantic formalism: average time step (from 0) at which syntactic and semantic dependencies are created as a function of their length. Error bars indicate the standard error of the mean.

to perform syntactic before semantic parsing[15], at least when dealing with this specific semantic formalism. Indeed, figure 3 shows that the strategy inferred when training on the DM formalism is radically different: short dependencies are still created before long ones (note that the number of dependencies decreases very rapidly with their length), but the syntactic and the semantic structures are built much more concurrently, with semantic structures being, on average, generated *before* syntactic ones of the same length. The strategy inferred for the PSD formalism, not depicted here, is qualitatively similar to the one for DM. In light of these observations, we can better understand why both tasks benefit from MTI: it is *not* true, in general, that syntactic parsing should be performed (even on average) before semantic parsing.

4.5 Comparison with other parsers

Finally, table 4 compares the performance of our free+RL model with state-of-the-art SDP parsers that, for fairer comparison, do not use ELMo or BERT embeddings, nor rely on lemma input: Peng et al. (2017)'s FREDA3, Peng et al. (2018a)'s SPIGOT, Wang et al. (2019)'s MF model and a model by Kurita and Søgaard (2019).[16] Note that our model is the only one that does *not* rely on the provided POS tags at inference time.[17]

Model	DM	PAS	PSD	all	avg.
FREDA3	90.4	92.7	78.5	88.0	87.2
K&S (-lem.)	91.2	92.9	78.8	88.5	87.6
SPIGOT	91.6		78.9		
MF	**93.0**	**94.3**	**80.9**		89.4
free+RL	91.1	91.7	79.0	88.1	87.3

Table 4: Semantic F1 on the in-domain test set for each of the three formalisms, along with the F1 over all formalisms (*all*) and the average of the F1 over all formalisms (*avg.*). *All* tends to be higher than *avg.* because PAS happens to be both the most simple formalism and the one introducing the highest number of dependencies, while the opposite is true of PSD.

Wang et al. (2019)'s second-order model, based on mean field variational inference, is clearly ahead in terms of performance. Among the other four, our model achieves competitive performance on the DM and PSD formalisms. It is, however, somewhat behind when it comes to the PAS formalism. Understanding why this is so is left as a subject of future research.

5 Conclusion and future work

We have defined Multiple Tasks Integration as a set of principles for joint processing, orthogonal to weight sharing. The essence of MTI is to process the input iteratively but concurrently on multiple

[15] On average only, as some semantic dependencies are still created before syntactic ones.

[16] Dozat and Manning (2018) report very high scores, but as indicated in the Q&A session of their ACL 2018 talk (https://vimeo.com/285804230) — a fact pointed out by Kurita and Søgaard (2019) —, these scores correspond to a macro-F1 instead of the micro-F1 computed by the shared task tool, thus giving more relative weight to shorter sentences.

[17] In the English data, the PAS and PSD graphs come with the tags of the PTB (Marcus et al., 1993) while the DM ones come with the tags of DeepBank (Flickinger et al., 2012).

levels of analysis, basing each decision on all of the structures already inferred and free from usual ordering constraints. This way, the different tasks can interact in the full sense of the term. To train such a system, we propose using reinforcement learning algorithms, thus allowing it to infer its own ordering strategy.

In practice, we have trained a system to perform part-of-speech tagging, syntactic dependency parsing and semantic dependency parsing. We have observed that both the use of reinforcement learning and the release from sequential constraints are beneficial, not only to the (seemingly) highest level task (i.e. semantic parsing), but also to some intermediate ones (i.e. syntactic parsing). If the inferred strategies are interpreted as being easy-first — which is supported by the fact that shorter dependencies have a strong tendency to be generated before longer ones —, then we have observed that syntactic parsing is not necessarily simpler than semantic parsing and that both benefit from being executed concurrently.

While our model is not yet as effective as today's most complex systems, it is still competitive with most of the parsers presented in the recent literature, even though it uses a poorer input signal for inference (consisting of the raw tokens only). Furthermore, several aspects of the current system are open to developments that seem likely to improve performance. For instance, we do not use here the full potential of reinforcement learning, as what we optimise (the expected sum of rewards) is not the metric we are interested in (which would be either an average of the three F1 or the semantic one). For each of the three tasks, the sum of the rewards we have defined approaches, up to a multiplicative constant, the corresponding F1 of the system when the latter approaches 100%, but a better approximation might prove more successful. In a similar vein, Kurita and Søgaard (2019), who work with a similar architecture as far as SDP is concerned, penalise HALT actions when relevant dependencies are still missing, which intuitively boosts recall. Note also that our network is still rather simple, in that it does not use any form of regularisation nor any advanced technique to handle out-of-vocabulary words.

We would also be interested in applying MTI to other tasks and in studying how well it can learn from incomplete annotations. Our architecture can be straightforwardly adapted to any labelling or graph building task, as long as all nodes are tokens of the input sentence. In contrast, work remains to be done in order to handle formalisms such as Abstract Meaning Representation (AMR, Banarescu et al., 2013) or on how to integrate a generation component with, for example, the goal of translating the sentence being analysed.

Acknowledgments

The author thanks Marie Candito for discussion on this paper.

This paper is based on results obtained from project JPNP15009, commissioned by the New Energy and Industrial Technology Development Organization (NEDO). The computational resources of the AI Bridging Cloud Infrastructure (ABCI), provided by the National Institute of Advanced Industrial Science and Technology (AIST), Japan, were used for this work.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract Meaning Representation for Sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186. Association for Computational Linguistics. Event-place: Sofia, Bulgaria.

Xavier Carreras and Lluís Màrquez. 2005. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 152–164, Ann Arbor, Michigan. Association for Computational Linguistics.

Rich Caruana. 1997. Multitask Learning. *Machine Learning*, 28(1):41–75.

Danqi Chen 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. Association for Computational Linguistics.

Matthieu Constant and Joakim Nivre. 2016. A transition-based system for joint lexical and syntactic analysis. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 161–171. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but More Accurate Semantic Dependency

Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. 2018. The Hitchhiker's Guide to Testing Statistical Significance in Natural Language Processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1383–1392, Melbourne, Australia. Association for Computational Linguistics.

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. DeepBank. A dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, pages 85–96.

Yoav Goldberg and Michael Elhadad. 2010. An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*, pages 1–18, Boulder, Colorado. Association for Computational Linguistics.

James Henderson, Paola Merlo, Gabriele Musillo, and Ivan Titov. 2008. A Latent Variable Model of Synchronous Parsing for Syntactic and Semantic Dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 178–182, Manchester, England. Coling 2008 Organizing Committee.

James Henderson, Paola Merlo, Ivan Titov, and Gabriele Musillo. 2013. Multilingual Joint Parsing of Syntactic and Semantic Dependencies with a Latent Variable Model. *Computational Linguistics*, 39(4):949–998.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Shuhei Kurita and Anders Søgaard. 2019. Multi-Task Semantic Dependency Parsing with Policy Gradient for Learning Easy-First Strategies. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2420–2430, Florence, Italy. Association for Computational Linguistics.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182, Brussels, Belgium. Association for Computational Linguistics.

Zachary C. Lipton. 2018. The Mythos of Model Interpretability. *ACM Queue*, 16(3):1–27.

Bruce T. Lowerre. 1976. *The HARPY Speech Recognition System*. Ph.D. Thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Chunchuan Lyu, Shay B. Cohen, and Ivan Titov. 2019. Semantic Role Labeling with Iterative Structure Refinement. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1071–1082, Hong Kong, China. Association for Computational Linguistics.

Minh Lê and Antske Fokkens. 2017. Tackling Error Propagation through Reinforcement Learning: A Case of Greedy Dependency Parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 677–687, Valencia, Spain. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

André Martins, Noah Smith, Mário Figueiredo, and Pedro Aguiar. 2011. Dual Decomposition with Many Overlapping Components. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 238–249, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. SemEval 2015 Task 18: Broad-Coverage Semantic Dependency Parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926. Association for Computational Linguistics. Denver, Colorado.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 Task 8:

Broad-Coverage Semantic Dependency Parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 63–72, Dublin, Ireland. Association for Computational Linguistics.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep Multitask Learning for Semantic Dependency Parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2037–2048, Vancouver, Canada. Association for Computational Linguistics.

Hao Peng, Sam Thomson, and Noah A. Smith. 2018a. Backpropagating through Structured Argmax using a SPIGOT. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1863–1873.

Hao Peng, Sam Thomson, Swabha Swayamdipta, and Noah A. Smith. 2018b. Learning Joint Semantic Parsers from Disjoint Data. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1492–1502, New Orleans, Louisiana. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. 11283.

Sebastian Ruder. 2017. An Overview of Multi-Task Learning in Deep Neural Networks. *arXiv:1706.05098 [cs, stat]*.

Libin Shen, Giorgio Satta, and Aravind Joshi. 2007. Guided Learning for Bidirectional Sequence Classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 760–767, Prague, Czech Republic. Association for Computational Linguistics.

Veselin Stoyanov and Jason Eisner. 2012. Easy-first Coreference Resolution. In *Proceedings of COLING 2012*, pages 2519–2534, Mumbai, India. The COLING 2012 Organizing Committee.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-Informed Self-Attention for Semantic Role Labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038. Association for Computational Linguistics. 00015 event-place: Brussels, Belgium.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester, England. Coling 2008 Organizing Committee.

Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*, second edition edition. Adaptive Computation and Machine Learning series. MIT Press.

Swabha Swayamdipta, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Greedy, Joint Syntactic-Semantic Parsing with Stack LSTMs. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 187–197. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-Order Semantic Dependency Parsing with End-to-End Neural Networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4618, Florence, Italy. Association for Computational Linguistics.

Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2018. A Neural Transition-Based Approach for Semantic Dependency Graph Parsing. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5561–5568. AAAI Publications.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

Jun Xie, Chao Ma, Janardhan Rao Doppa, Prashanth Mannem, Xiaoli Fern, Thomas G. Dietterich, and Prasad Tadepalli. 2015. Learning Greedy Policies for the Easy-First Framework. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Sheng Zhang, Xutai Ma, Kevin Duh, and Benjamin Van Durme. 2019. Broad-Coverage Semantic Parsing as Transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3784–3796, Hong Kong, China. Association for Computational Linguistics.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency Parsing as Head Selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 665–676, Valencia, Spain. Association for Computational Linguistics.

Yu Zhang and Qiang Yang. 2018. A Survey on Multi-Task Learning. *arXiv:1707.08114 [cs]*.

Jie Zhou and Wei Xu. 2015. End-to-end learning of semantic role labeling using recurrent neural networks. 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 1127–1137, Beijing, China. Association for Computational Linguistics.

A Appendix: Base encodings

The base encoding of a given token (introduced in Section 3.3) is composed of the concatenation of four vectors and five binary values.

- The GloVe embeddings that we use are the 100-dimensional vectors of the 6B (uncased) release. We do not fine-tune them. All words present in the training set use their corresponding GloVe entry. All other words are considered unknown and are assigned the average of these embeddings.
- For POS tag embeddings, we use randomly initialised 50-dimensional vectors.
- We use a sum of prefix embeddings. We first consider all cased prefixes of length 1, 2 or 3 and then filter out all those that appear in less that a thousandth of the tokens and less that a thousandth of the word forms in the training set. The remaining prefixes are assigned a randomly initialised 32-dimensional vector. (Unknown prefixes correspond to a zero vector.)
- We use a sum of similar suffix embeddings.
- One binary value indicates whether the token starts with an upper case letter.
- One binary value indicates whether there is any upper case letter in the token.
- One binary value indicates whether the token is a number (matching the `\d+(\.\d+)?` regular expression).
- One binary value indicates whether the token is currently annotated as the (syntactic) root.
- One binary value indicates whether the token is currently annotated as a (semantic) top predicate.

In future work, we plan to substitute a more general character-level word embedding model for the tagging features (i.e., the prefix and suffix embeddings and the three first binary features).