The Reading Machine: a Versatile Framework for Studying Incremental Parsing Strategies

Franck Dary, Alexis Nasr
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{franck.dary,alexis.nasr}@lis-lab.fr

Abstract

The Reading Machine, is a parsing framework that takes as input raw text and performs six standard NLP tasks: tokenization, POS tagging, morphological analysis, lemmatization, dependency parsing and sentence segmentation. It is built upon Transition Based Parsing, and allows implementing a large number of parsing configurations, among which a fully incremental one. Three case studies are presented to highlight the versatility of the framework. The first one explores whether an incremental parser is able to take into account top-down dependencies (i.e. the influence of high level decisions on low level ones), the second compares the performances of an incremental and a pipe-line architecture and the third quantifies the impact of the right context on the predictions made by an incremental parser.

1 Introduction

Syntactic parsers usually take as input text that has been processed at several levels. It has generally been segmented in sentences, tokenized, POS tagged, and possibly lemmatized and morphologically analyzed. All such steps are realized by other NLP modules that are usually organized in a sequential pattern, called a pipe-line. The pipe-line imposes a rational order on the modules: word boundaries, for example, have to be determined before a word can be associated to a POS tag and syntactic parsing usually comes after POS tagging, for POS tags group words that have close syntactic properties.

The pipe-line architecture offers many advantages, among which, the independence of the modules that compose it. The only constraint for their inter-operability is the compatibility of their inputs and outputs. Once this constraint is verified, each module can be built on any kind of model considered more suitable for the task to perform. Besides, in a pipe-line architecture, every module narrows down the search space of the following modules: a parser, for example, does not have to consider different tokenization hypotheses nor different POS tags for a word. Considering all such decisions can lead to a combinatorial explosion problem and yields huge search spaces.

The pipe-line architecture nevertheless has its limits. It is well known that some low level decisions (made by early modules of the pipe-line) can benefit from high level ones (made by late modules). Some tokenization decisions, for example, can depend on the syntactic structure of the sentence to parse, such as complex prepositions in French, as noted by Nasr et al. (2015). Likewise, sentence segmentation can depend on syntactic structures, especially when punctuation is absent or unreliable, such as in speech transcriptions. Such *top down dependencies* cannot be taken into account in a strict pipe-line architecture. But they are arguably less numerous than *bottom up dependencies* and this is the reason why the pipe-line architecture usually yields good results. One aim of this paper is to propose a framework, called the Reading Machine (RM), that is flexible enough to define several patterns for combining different NLP modules and explore different ways to link decisions made by these modules. We use in this paper the RM framework to define and compare several non sequential machines that model top down dependencies.

There is another, less immediate, reason for studying non sequential architectures, in link with human cognition. Psycholinguistic studies have shown that human language processing is incremental, i.e., people do not wait to see the entire sentence before they start trying to understand it (see Keller (2010) for more details). Such findings have developed interest in using NLP tools to implement cognitively-plausible models of hu-

Proceedings of the 17th International Conference on Parsing Technologies (IWPT 2021), pages 26–37
Bangkok, Thailand (online), August 6, 2021. ©2021 Association for Computational Linguistics

man sentence processing (see Hale (2017) for a review). Various "linking hypotheses" have been proposed to relate the models' intermediate states when parsing a given sentence to the behavior of human subjects trying to understand that same sentence. The RM offers a framework to investigate such linking hypotheses by defining machines that implement them and observe their behaviour on human data. In this perspective, the RM has already been used for predicting eye-movements during reading: different RM architectures have been compared on their ability to accurately predict fixation time (Dary et al., 2021a,b). In order to illustrate the kind of experiments that can be conducted in such a perspective, we will define and compare machines that model different perceptual fields and measure their influence on an incremental parsing process.

Technically, RM is an extension of the transition-based parsing algorithm (TBP) (Yamada and Matsumoto, 2003; Nivre, 2003). The reason for this choice is mainly that TBP implements an incremental parsing strategy (Nivre, 2008). We propose to extend this model to define a complete incremental NLP parser that integrates six tasks: tokenization, POS tagging, morphological analysis, lemmatization, syntactic parsing and sentence segmentation. RM borrows from TBP the two key notions of *Configurations* and *Actions* (also called *Transitions*), as well as a greedy algorithm that performs syntactic parsing. We extend these notions by defining an enriched version of a configuration and a richer set of actions. All linguistic decisions, such as word and sentence boundaries detection, POS tagging, lemmatization and, of course, syntactic parsing are realized by actions that are predicted based on configurations. The RM takes as input raw text and greedily predicts a sequence of actions that perform the six tasks mentioned above.

2 Related Work

Solving the circular dependencies that exist between parsing and other pre-processing steps, is an active area of research in the parsing literature. The solution that has been mainly investigated consists in jointly performing syntactic parsing and other pre-parsing steps. If we restrict ourselves to recent approaches to dependency parsing, solutions have been proposed both for graph-based parsing (Yan et al., 2020; Lee et al., 2011; Nguyen and Verspoor, 2018; Nasr et al., 2015; Li et al., 2011; Zhang et al., 2015) and transition-based parsing (Yoshikawa et al., 2016; Bohnet and Nivre, 2012; Alberti et al., 2015; Hatori et al., 2012; Honnibal and Johnson, 2014; Constant and Nivre, 2016; Kurita et al., 2017; Wan et al., 2018).

Solutions to this problem differ vastly for these two approaches, mainly because of the different parsing strategies they adopt. This is why we have decided to restrict ourselves to TBP based papers in the remainder of this section.

There have been many propositions to realize simultaneously several linguistic tasks in TBP. Both Bohnet and Nivre (2012) and Alberti et al. (2015), for example, show how a transition system can be extended and trained to jointly predict POS tags and the dependency tree, improving both the accuracy of tagging and parsing. These systems are not strictly incremental across the tasks they realize for they process text that has already been segmented in words and sentences. Besides, the text has been pre-tagged in order to limit the size of the search space.

The closest approach to ours is Kurita et al. (2017), which is based on the work of Hatori et al. (2012). The authors propose an extension of the arc-standard transition system that is able to perform a fully joint prediction of word segmentation, POS tagging and dependency parsing. Their system takes a queue of symbols as input, and process it in an incremental fashion, consuming one symbol at a time. They conducted their experiments on Chinese, and were the first to use a neural network architecture using both word and character based embeddings to achieve fully joint prediction of these three tasks. They showed that their joint architecture was competitive with a pipeline architecture for word segmentation and POS tagging, but fell short on parsing.

For their participation in the 2018 CoNLL Shared Task (Zeman et al., 2018), Wan et al. (2018) also defined an extension of TBP that is close to ours. It is based on the arc-standard system enriched with a swap transition (Nivre, 2009), and is able to jointly perform word segmentation, POS tagging, morphological tagging and dependency parsing. They showed that such a system could get better scores than the shared task's baseline, while still being quite far from the top scoring systems. Their paper focuses on low or even zero resources languages (what the shared task was mainly about), and did not test whether or not the joint prediction of multiple tasks was improving the performances.

Our model has the following characteristics that distinguishes it form the approaches cited above.

The RM performs simultaneously six NLP tasks with the notable inclusion of sentence segmentation which is almost always pre-processed in parsing systems.

The RM allows us to build a machine that is strictly incremental across the six tasks it realizes. There are two reasons for this choice. The first is theoretical, we are interested to know how much information is present in top-down dependencies and whether they can be captured in an incremental setup. The second is related to psycholinguistics: we believe that the definition of a fully incremental RM offers a useful model for simulating human behaviour during reading.

The RM is flexible enough to design machines that implement different strategies in order to compare them. We propose, in section 5, a high-level description format that allows us to define machines that differ on a specific dimension and study the effect of this dimension on the performances of the machines.

3 The Reading Machine

As mentioned in the introduction, the reading machine is an extension of the TBP[1] framework. The details of this algorithm are well known and do not need to be repeated here. We will only introduce the terms that are important for the rest of the paper.

TBP is an algorithm that predicts the dependency syntactic structure of sentences. This task can be viewed as selecting for each word w of a sentence its syntactic governor (another word w' of the sentence) as well as its syntactic function f. A directed arc, referred to as a *dependency*, is built from w' to w, labeled with function f. The graph built at the end of the parsing process is a tree. The TBP algorithm builds the dependency tree by scanning the sentence word by word, in reading order. At each step, an *action*, is predicted and applied to the current *configuration* of the parser and yields a new configuration. The prediction is realized by a classifier that takes as input a configuration and computes a score for every possible action. The parser makes use of a stack containing words that need to be linked to words yet undiscovered.

Four types of actions are defined: SHIFT, pushes the current word on the stack, REDUCE, pops the stack, LEFT$_l$, creates a left dependency, labeled l, whose governor is the current word and dependent is top element of the stack and RIGHT$_l$, which creates a right dependency, labeled l, whose dependent is the current word and governor is the top element of the stack.

<table>
<tr><th>SEG</th><td colspan="4">NO</td><td colspan="4">NO</td><td colspan="5">NO</td><td colspan="4">NO</td><td colspan="4">NO</td><td>YES</td></tr>
<tr><th>SYN</th><td colspan="4">DET</td><td colspan="4">SUB</td><td colspan="5">ROOT</td><td colspan="4">DET</td><td colspan="4">OBJ</td><td>PCT</td></tr>
<tr><th>GOV</th><td colspan="4">+1</td><td colspan="4">+1</td><td colspan="5">0</td><td colspan="4">+1</td><td colspan="4">-2</td><td>-3</td></tr>
<tr><th>LEM</th><td colspan="4">@</td><td colspan="4">@</td><td colspan="5">s@</td><td colspan="4">@</td><td colspan="4">@</td><td>@</td></tr>
<tr><th>MRF</th><td colspan="4">DEF</td><td colspan="4">SG</td><td colspan="5">P3S</td><td colspan="4">DEF</td><td colspan="4">SG</td><td>-</td></tr>
<tr><th>POS</th><td colspan="4">DET</td><td colspan="4">N</td><td colspan="5">V</td><td colspan="4">DET</td><td colspan="4">N</td><td>PCT</td></tr>
<tr><th>TOK</th><td colspan="4">the</td><td colspan="4">boy</td><td colspan="5">hits</td><td colspan="4">the</td><td colspan="4">ball</td><td>.</td></tr>
<tr><th>INPUT</th><td>t</td><td>h</td><td>e</td><td></td><td>b</td><td>o</td><td>y</td><td></td><td>h</td><td>i</td><td>t</td><td>s</td><td></td><td>t</td><td>h</td><td>e</td><td></td><td>b</td><td>a</td><td>l</td><td>l</td><td>.</td></tr>
</table>

Table 1: Input and output tapes of a RM after processing the text *The boy hits the ball*.

Before giving a precise definition of the RM, in section 3.1, we describe the directions in which the TBP has been extended.

Tapes: The RM has one input tape, which is a read tape and an arbitrary number of output tapes which are read/write tapes.

The input tape contains the text to parse. It is character based: each cell of the tape contains a character. The text has not been linguistically pre-processed: it has not been segmented into sentences nor into words. The current position of the reading head of the input tape is called the *character index*.

Output tapes are word based: each cell of a tape refers to a word of the input text. Output tapes are used to write the predictions made by the machine, typically one tape per type of prediction. These tapes are synchronized: at all times, the head is at the same position for all tapes. This position is called the *word index*.

Table 1 represents the tapes of a machine after processing the text *The boy hits the ball*. The machine has 7 output tapes and one input tape, represented at the bottom.

Sliding Window: The reading head of the input tape takes the form of a sliding window. It is centered on the cell of the tape pointed by the *character index* and has access to an arbitrary number of cells to the left and to the right of this cell.

States and Transitions: TBP can be seen as a single state machine, in contrast RM are multi-state machines. Every state, or set of states, is devoted to a specific linguistic task and is linked to a classifier. The linking between states and classifiers can range from a single classifier for all states to one classifier

[1] Several sets of actions have been proposed in the literature, the one used here is known as *Arc Eager*.

per state. States are deterministically linked to each other through transitions that are labelled with action labels. At each step, the classifier of the current state predicts the next action to perform. The action is applied on the configuration and the transition labeled with this action is traversed to reach another state.

Actions: RM actions encompass standard *Arc Eager* actions for parsing and new actions have been defined for performing the other tasks.

Word tagging tasks (POS tagging and morphological tagging) are realized through a single action: TAG_L(t), which simply writes symbol t on tape L at the word index position. For example, action TAG_{POS}(DET) tags the current word as a determiner.

It is not straightforward to cast lemmatization as a classification task, due to the large number of classes (potentially all the lemmas of a language). Besides, lemmatization is, to a large extent, regular. In order to capture this regularity, the classifier that realizes the lemmatization task predicts editing rules of the form `s1@s2` where `s1` is a suffix of the word to lemmatize and `s2` the suffix of the lemma.[2] When applied to a word w, such a rule strips off suffix `s1` from w and appends `s2`, as in the following example *apply(*`s@`*, hits) = hit*.

The actions predicted by the tokenizer are of four types: ADD_n adds the n next characters of the input tape to the current word and moves the character index n positions to the right, IGNORE ignores the current character (typically spaces) and moves the character index to the right, WORD marks the current word as complete and SPLIT^w_W action moves the character index $|w|$ positions to the right and adds the word sequence W in the buffer. This last action is used to expand contractions such as *don't* → *do not*.

Programming an oracle function for actions IGNORE, WORD and SPLIT is straightforward because there is no ambiguity on which is correct at any time. However, the choice of ADD_n is ambiguous: if we consider that the 8 next characters on the input tape are "*academic*", we want to add them to the current word. This could be done using several action sequences, such as "ADD_8", "ADD_4,ADD_4" or "ADD_2,ADD_3,ADD_3,". In our experiments, we chose to limit the size of ADD_n to n=6 and to program the oracle so that it adds the largest number of characters at once. During training, the correct action sequence would then be: "ADD_6,ADD_2"

State	Action	Description
TOK	ADD_n	Adds the n next symbols to b.0.
TOK	IGNORE	Ignores the next symbol.
TOK	WORD	Marks b.0 as complete.
TOK	SPLIT^w_W	Consume symbol sequence w. Add word sequence W in buffer.
POS, MRF	TAG_L(t)	Writes tag t to b.0 on tape L.
LEM	s@s′	b.0 lemma := form $- s + s'$.
LEM	CASE_{ul}	b.0 lemma to upper/lower case.
SYN	REDUCE	Pop the the stack.
SYN	SHIFT	Push b.0 on the stack.
SYN	RIGHT_l	Adds arc (s.0,b.0,l). Push b.0 on the stack.
SYN	LEFT_l	Adds arc (b.0,s.0,l). Pop the stack.
SEG	EOS(Y/N)	Mark b.0 as an end of sentence, set sentence root, attach orphans to root then empty stack.

Table 2: Actions used in our RM architecture. b.0 stands for the current word and s.0 for the word on top of the stack.

Sentence segmentation is realized by a binary action EOS(YES/NO) which tags the current word as the end of the current sentence, or not. Once the end of a sentence has been detected, the deepest element in the stack is marked as the root of the sentence and the potential remaining elements of the stack are attached to the root, before emptying the stack. The complete set of actions is reported in Table 2.

3.1 Formal Definition

A RM, as any formal automata, has an input alphabet Σ_T, which is a set of characters, a set of states $\mathcal{S}$, a transition function δ, an initial state s_0 and a set F of final states. It also has N output alphabets $\Sigma_1, \ldots \Sigma_N$ associated to its N output tapes. Transitions between states are labelled with actions. Moreover, each state is associated, via function γ, to a classifier, which maps configurations to actions, along with a score.

Formally, we define a RM as a tuple $T = (\Sigma_T, N, \Sigma, \mathcal{S}, s_0, \mathcal{A}, \mathcal{K}, \gamma, \delta, F)$. We develop below the elements of the machine that deserve more explanation.

[2] Of course, such a simple form of rules can only deal with suffixal flexional morphology. More complex morphological phenomena, in templatic morphology for example, ask for more elaborate types of rules.

- $\mathcal{A}$ is a set of actions. Each action can write a symbol on an output tape, move the character head or the word head either to the left or to the right and push or pop the stack (see Table 2).
- $\mathcal{K}$ is a set of classifiers, described in section 3.3
- $\delta : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is a deterministic transition function, also called a *Strategy*. Given the current state and its associated classifier, the action selected by the classifier is performed and control jumps to the destination state of the transition. The strategy defines the order in which the predictions are made. For instance a strategy could force the lemmatization task to happen after the dependency parsing task. Two different strategies are described in section 5.
- $\gamma : \mathcal{S} \rightarrow \mathcal{K}$ is a function that maps each state of the RM to a classifier. This mapping allows several states to share a single classifier which allows in turn jointly training several processes.

3.2 Configuration

Configurations for a RM M and a text T are defined as $(S, T, c, \beta_{1,N}, w, \sigma, H)$, where:
- S is the current state of M.
- T is the input tape
- c is the character index
- $\beta_{1,N}$ is a collection of output tapes.
- w is the word index
- σ is a stack of word indexes, its purpose is the same as the stack in TBP.
- H is the sequence of transitions that have been predicted till now.

The set of all configurations for text T is noted $\mathcal{C}_T$.

An *initial configuration* for a text T is defined as follows: $(s_0, T, 0, \beta, 0, [], [])$, where all tapes in β are empty.

An *accept configuration* is defined as $(s \in F, T, n_c, \beta, n_w, [], H)$, where n_c and n_w are respectively the number of characters and words in T.

3.3 Classifier

The classifiers that constitute the set $\mathcal{K}$ are functions that map a configuration to actions and scores.

The classifiers are independent of each other but they all take as input a configuration which contains all aspects of the current state of the RM, in particular, the content of all the tapes of the RM. The tapes contain all predictions already realized. Each classifier defines its own *Feature Function* which extracts from the input configuration all features considered useful for the type of prediction it realizes. We will not delve into the set of all possible features, let us just mention that most of them allow to access the content of a specific cell of a specific tape.

4 Training the RM

The training process of a RM is close to TBP training. It starts with a dependency tree that is decomposed into a sequence of (*state*, *configuration*, *action*) triples, by a static oracle. The difference with TBP is that the set of actions is considerably larger since it encompasses actions for all six tasks. This sequence is used to train the classifiers: every classifier receives examples corresponding to the states it is related to. The RM is trained using only correct examples, predicted by the oracle. In order to increase the robustness to error propagation, we use a dynamic oracle (Goldberg and Nivre, 2012) to extract a new set of training examples where the actions applied were the one predicted by the network.[3] The RM is therefore trained to predict the next action given potentially incorrect configurations.

Four epochs are devoted to the first part of the training process, followed by 26 more epochs in dynamic oracle regime.[4] At each epoch, the machine is used to decode the development set, and is saved if its score (mean score across all 6 levels) is the best so far. We used cross entropy as a loss function and Adagrad (Duchi et al., 2011) for optimization.

The classifiers used to predict the actions can be decomposed into three parts: the first part is devoted to feature extraction, it is composed of several encoders applied to different parts of the RM configuration. The output of these encoders are then concatenated yielding a dense vector representation of the current configuration, to which we apply a dropout of 0.5 and feed it into a Multi Layer Perceptron (MLP) composed of 2 hidden layers of respective sizes 3200 and 1600 with dropout 0.4 and ReLU activation. The output of the MLP is then fed into a decision layer, producting a probability distribution over the possible actions. The most probable action is predicted then applied to the configuration. A classifier can have up to 6 decision layers, one for every task. A schematic representation of the classifier structure can be found in

[3] Except for tokenization and sentence segmentation actions, because our dynamic oracle is not able to deal with incorrect segmentation.

[4] All hyper-parameters have been optimized for the development set described in section 6.

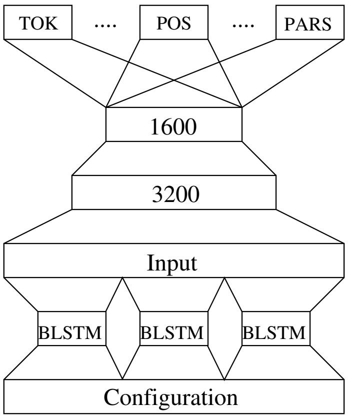

Figure 1: General structure of the classifiers of a RM. They take as input a configuration and predict an action for up to 6 tasks.

Figure 1.

The most complex part of the classifier is the transformation of the configuration into a vector: the input of the MLP. Depending of its feature function, the classifier extracts from the configuration elements that can be of different natures: the current state of the RM, tags or words read from the tapes, characters read from the input tape, previous action present in the history and words from the stack. All these elements are fed to specific Bi-LSTM that produce contextual representations, as in Kiperwasser and Goldberg (2016), that are in turn concatenated in the MLP input layer.

All feature values are represented by trainable embeddings of size 128. These embeddings are randomly initialized, except for word embeddings that are pretrained[5] exclusively on the train set, in order to produce an embedding for unknown words (using words occurring only once in the train set).

The Bi-LSTM that take sequences of these embeddings as input are made of only one layer of size 64.

5 Designing Reading Machines

The precise definition of RM, introduced in section 3, allows us to design a large number of machines. Every machine is defined by a large number of features and comparing machines is not always easy. In this section we define a more abstract description that is based on four high-level features, and define seven machines that are compared in section 6.

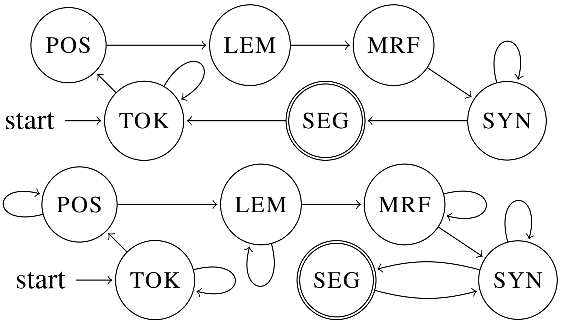

Figure 2: Two RM strategies that correspond to the order of predictions. Above, the INCR strategy and below the SEQ strategy.

We first describe the four high-level features then give a description of the machines.

5.1 Strategy

As mentioned in section 3, the strategy of a RM is the structure of the underlying automaton: its number of states and transition function. A strategy dictates the order in which the predictions are made. Two strategies: INCR and SEQ, are defined, they are represented in Figure 2. The actions that label the transitions have been omitted for readability reasons.

The main difference between the two strategies comes from the loops on all states of the SEQ strategy. These loops model the sequential behaviour of the RM: the whole text is processed at a given level before switching to the upper level. In contrast, the INCR strategy processes a word at a given level then performs a prediction at the next higher level for the same word. This difference can be illustrated in the way the matrix of Table 1 is filled: the SEQ machine fills it line by line, bottom-up, while the INCR machine fills it column by column, from left to right.

5.2 Feature Span

The *Feature Span* of a machine specifies the part of the tapes that are accessible to each classifier in order to make a prediction. Three feature spans have been defined, they are represented in Figure 3. Each of the three rectangles represents, schematically, the tapes of an RM, as in Table 1. The black square corresponds to the current prediction and the hatched area to the content of the tapes available for the current prediction. The past-low (PA-LO) feature span only sees the tapes content for the lower level past predictions. Future-low (FU-LO) sees the past, current and future predictions for lower levels.

[5] Using GloVe (Pennington et al., 2014), implementation: https://github.com/stanfordnlp/GloVe.

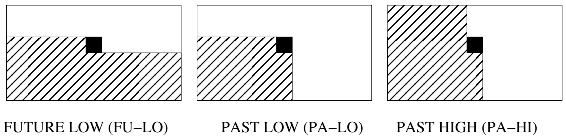

Figure 3: Three feature spans: the black square is the current prediction and the hatched area, the available features.

Past-high (PA-HI) have access to low, current and high-level predictions from the past.

In the PA-HI configuration, when the tagger, for example, has to select the POS tag of a word, it has access to the predictions made by all the modules, including the parser, for preceding words. The PA-HI feature span therefore offers an *explicit* way to model top down dependencies.

5.3 Number of Classifiers

As mentioned above, several states of a machine can share a single classifier to predict the actions associated to these states. The sharing of a single classifier by several states amounts to perform multi-task training, which "*uses the domain information contained in the training signals of related tasks as an inductive bias. It does this by learning tasks in parallel while using a shared representation; what is learned for each task can help other tasks be learned better*" (Caruana, 1997).

In our case, when a single classifier is used, for example, to perform both POS tagging and parsing, the representation of a configuration built by this classifier is obtained by optimizing both tasks. Decisions made by the parser can therefore *implicitly* influence the tagger. The Number of Classifiers is, with the Feature Span, the two ways that will be tested to take into account top down dependencies.

Two extreme choices have been made with respect to this dimension, in the first one, all states share a single classifier while in the second each state defines its own classifier, yielding two different values: 1 and 6.

5.4 Window Span

The *Window Span* of a machine is simply the span of the sliding window that gives access to the text. A window span is defined by a pair of integers that indicate how many characters, to the left and to the right of the character index, are accessible. The window span corresponds to the sliding window introduced by McConkie and Rayner (1975) used to model the perceptual span of a human reader.

Four different window spans have been defined: [-5,2], [-5, 5], [-5, 10] and [-5, 15]. We followed McConkie and Rayner (1976) in choosing asymmetric windows.

5.5 Seven Machines

Given the four dimensions defined above and the number of values per dimension, a total of 48 different machines can be defined. We have selected, among these, seven machines that can be grouped in four subsets, as shown in Table 3. Machines in a subset generally differ from one another for a single dimension, the exception are the two machines of the first subset. These four subsets correspond to the four experiments that are described in the following section. The letter in the first column of the table indicates identical machines. They have been given several names in order to ease the comparisons in section 6. The feature function of the classifiers of each of these machines can be found in table 4.

	RM	Strat.	F.Span	W.Span	NC
A	PA-HI	INCR	**PA-HI**	[-5,10]	6
	PA-LO	INCR	**PA-LO**	[-5,10]	6
B	C1	INCR	PA-HI	[-5,10]	**1**
A	C6	INCR	PA-HI	[-5,10]	**6**
B	INCR	**INCR**	**PA-HI**	[-5,10]	1
	SEQ	**SEQ**	**FU-LO**	[-5,10]	1
	[-5,2]	INCR	PA-HI	**[-5,2]**	6
	[-5,5]	INCR	PA-HI	**[-5,5]**	6
A	[-5,10]	INCR	PA-HI	**[-5,10]**	6
	[-5,15]	INCR	PA-HI	**[-5,15]**	6

Table 3: Definition of the machines used in the experiments. The letter in the first column indicates identical machines. NC stands for Number of Classifiers.

6 Experiments

The experiments presented in this section aim at exploring three directions. The first one is the modelling of top-down dependencies. We have introduced two means for taking into account such dependencies in a RM: joint prediction of several tasks, using a single classifier, and the feature span of the classifiers. The two first experiments aim at studying whether these two techniques allow to effectively model such dependencies. The second direction compares the sequential and the incremental strategies and measure how much information a parser gets from the knowledge of the next words

Machine	Features
C1 INCR	FORM,ID,POS,MRF,SYN: b.-3 b.-2 b.-1 b.0 s.0 s.1 s.2 b.0.0 s.0.0 s.0.-1 s.1.0 s.1.-1 s.2.0 s.2.-1 Prefix&Suffix of size 5: b.0 Raw text: [-5,10] around *character index* History: past 10 actions Split: list of applicable SPLIT actions Name: name of the current state Distances: from s.0 s.1 s.2 to b.0.
SEQ	Same as INCR with the addition of right context: b.1 b.2.
PA-HI C6 [-5,10]	Each of the 6 classifiers has the same features as C1.
PA-LO	Each of the 6 classifiers is different. The classifier corresponding to a given linguistic level will only have the features of PA-HI corresponding to this linguistic level and inferior levels. For example, the classifier corresponding to MRF will only access columns FORM,ID,POS,MRF for targets b.-3 b.-2 b.-1 b.0 and will not have the distance feature.
[-5,2] [-5,5] [-5,15]	Same as [-5,10] but with window [-5,2]. Same as [-5,10] but with window [-5,5]. Same as [-5,10] but with window [-5,15].

Table 4: Features used in our RM architecture. "b.i" stands for the word at position *word index*+i in the buffer and "s.i" for the ith topmost word on the stack. Additional suffixes ".0" and ".-1" refer respectively to the leftmost and rightmost dependent.

POS, lemma and morphological analysis. The third direction aims at measuring the effect of the window span on the performances of an incremental RM.

The machines realize six types of predictions: tokenization, part of speech tagging, lemmatization, morphological analysis, syntactic parsing and sentence segmentation. Each of these predictions are evaluated by a specific metric using the evaluation script of the CoNLL 2018 shared task (Zeman et al., 2018). Besides the task specific metrics, we report the Morphology-Aware Labeled Attachment Score (MLAS) which takes into account word segmentation, morphological and POS tagging as well as syntax, as described in Zeman et al. (2018). The MLAS allows us to compare the general performances of two machines, while the task specific metrics allow for a finer comparison of these machines.

Experiments have been conducted on French[6] data, using the GSD corpora of the Universal Dependencies collection (Zeman et al., 2019) version 2.7. In the official distribution of this corpus, the train/dev/test split is of respective sizes 364,349/36,775/10,298 words. In preliminary experiments, we found out that the test split was way too small to meaningfully compare machines. That's why we decided to use 10 fold cross-validation: we realized ten different 80%/10%/10% train/dev/test splits and trained ten copies of each of our machines on these splits. The ten test sets were then decoded by the corresponding copy of the machine and the predictions were concatenated. This technique allowed us to compare the machines on their predictions on a test set of 427,763 words. We tested the significance of our comparisons using paired bootstrap resampling[7] (Koehn, 2004), and reported in our tables the corresponding p-value, estimating the probability that in a pair of models, the model that appear to perform better is in reality a worse model. Unfortunately, the resampling script we used (the one used in the CoNLL 2018 shared task) is not able to produce p-values for the task of sentence segmentation, that is why the corresponding cells in tables 5,6 and 7 are empty.

6.1 Wide *vs* Narrow Feature Span

This experiment aims at studying if past high level predictions can help current low level predictions, which is the explicit means we have proposed to model top-down dependencies. In order to test this hypothesis, we compare machines PA-LO and PA-HI. Both machines differ on their Feature Span. PA-HI has access to past high-level predictions while PA-LO does not.

The results, displayed in Table 5, show that PA-HI yields lower results than PA-LO. This is true for the general MLAS measure, as well as sentence segmentation and morphological tagging. The differences obtained by the two machines on the other tasks are not significant. Contrary to what we expected, using past high level predictions does not seem to increase the performances of low-level modules. We do not have for the moment an explanation for this result. It could be explained by the errors made by PA-HI on earlier predictions which, in turn, provoke errors on current word predictions.

6.2 Multi Task *vs* Mono Task

The aim of our second experiment is to test the second means we have proposed to model top-down

[6] The same experiments can be replicated on any language of the UD collection. Comparing the respective merits of the different machines across languages is a very interesting but very resource demanding task (mainly for optimizing the hyper-parameters for each language) and we leave this for future work.

[7] Using implementation of Popel et al. (2017).

Task	PA-LO	PA-HI	p-value
MLAS	**77.70**	77.30	0.014
Seg	96.65	96.57	
LAS	86.88	86.77	0.193
UAS	89.13	89.03	0.183
Lemma	98.04	98.01	0.22
UFeats	**97.02**	96.87	0.003
UPOS	97.02	96.93	0.051
Words	99.60	**99.64**	0.02

Table 5: Wide *vs* Narrow Feature Span

dependencies: multi task prediction. Two machines are compared: C1 which uses a single classifier to predict all tasks and C6 which uses a specific classifier for every task. The results are reported in Table 6. The table shows that C1 outperforms C6 on the MLAS metric: on average, a multi-task setup performs better than a mono-task one. C1 is significantly better than C6 for parsing, morphological and POS tagging. While the two machines are equivalent on the other tasks. These results show that predictions based on representations of the machine configurations that are optimised for all tasks are beneficial for all tasks. It is tempting to conclude that multi task learning is an effective way to model top down dependencies. It is unfortunately premature to draw such a conclusion for multi task learning is a complex process that models a large number of dependencies in the data. More investigation using, for example, probing, is in order to give a definite answer to this question.

It is worth noting that C1 has six times less parameters than C6, because each classifier of C6 has as many parameter as the classifier of C1, another argument in favor of multi-task training.

Task	C1	C6	p-value
MLAS	**77.93**	77.29	0.001
Seg	96.40	96.57	
LAS	**87.08**	86.77	0.008
UAS	**89.33**	89.03	0.004
Lemma	98.02	98.01	0.39
UFeats	**97.01**	96.87	0.004
UPOS	96.99	96.93	0.135
Words	99.65	99.64	0.315

Table 6: Multi Task *vs* Mono Task

6.3 Sequential *vs* Incremental

In this experiment, two machines are compared, SEQ, a sequential machine that implements a pipeline architecture, and INCR that implements an incremental architecture. These two machines differ on two dimensions, their strategy as well as their feature span. The aim of this experiment is to measure to which extend the information given to a parser by the next words low level analysis (POS tagging, morphological tagging and lemmatization) help a parser. The SEQ machine implements a two words look-ahead: the parser has therefore access to the form, POS, lemma, and morphological analysis of the next two words.

The results of this experiment are reported in Table 7. As one can see, SEQ outperforms INCR on the MLAS metric: on average, SEQ gets better results than INCR. As was expected, it is the parser that takes advantage of the sequential strategy: both LAS and UAS are increased by around one point. The two architectures achieve equivalent performances on the low level tasks, meaning that early processing steps do not take advantage of the sequential architecture.

Task	INCR	SEQ	p-value
MLAS	77.93	**78.60**	0.000
Seg	**96.40**	95.96	
LAS	87.08	**87.56**	0.000
UAS	89.33	**89.89**	0.000
Lemma	98.02	98.01	0.403
UFeats	97.01	**97.16**	0.002
UPOS	96.99	**97.09**	0.036
Words	99.65	99.63	0.115

Table 7: Sequential *vs* Incremental

6.4 Window Span

Our last experiment aims to study the influence of the Window Span on the performances of an incremental machine. Four machines are compared: [-5,2], [-5,5], [-5,10], [-5,15].

The best MLAS results are obtained by machine [-5,10] which defines a look-ahead of 10 characters for its predictions. It is interesting to note that low values of look ahead has a dramatic effect on the performances. This is partly due to the metrics used in which tokenization errors provoke errors on higher level modules. As expected, we observe diminishing returns as we increase the Window Span to the right, and going to 15 characters slightly decreases performances. These results are in line with McConkie and Rayner (1975) who determined experimentally that the span of the window for humans is about four characters to the left of the current character and twelve characters to the right.

Task	[-5,2]	[-5,5]	[-5,10]	[-5,15]
MLAS	48.53	76.76	**77.29**	**77.19**
Seg	79.75	96.44	**96.57**	**96.68**
LAS	59.39	86.28	**86.77**	**86.69**
UAS	61.05	88.56	**89.02**	**88.96**
Lemma	83.67	97.89	**98.01**	**97.99**
UFeats	82.58	96.74	**96.87**	**96.83**
UPOS	82.46	96.71	**96.93**	**96.90**
Words	85.12	99.55	**99.64**	**99.65**

Table 8: Different values of Window Span

7 Conclusion and Future Work

This paper introduced a versatile parsing framework, called the Reading Machine, that allows us to compare incremental parsers that implement different parsing configurations. We illustrated this with two cases. In the first one, we compared different ways to take into account top down dependencies across six tasks. In the second, we compared the effect of the look-ahead on the parsing performances.

This work will be extended in several directions.

The first one concerns the analysis of the results obtained in our two first experiments. More investigation is needed to explain the reason why using past high level predictions cannot help low level current ones. Likewise, we would like to understand if the better results of multi task learning comes from the modelling of top down dependencies. Two means will be used in order to answer this question: probing and the development of (small) specialized test sets that focus on such phenomena.

The sequential machines that have been proposed are greedy: a local decision taken by a machine is never questioned even when contradicted by future events. This behaviour is not appealing from an NLP perspective nor from a psycholinguistic one. In order to tackle this problem, we will introduce in the Reading Machine a backtrack mechanism that can predict left movements leading to the re-analysis of a previously analyzed part of the text.

The third direction consists in evaluating the model against human experimental data. The Reading Machine has already been used to predict reading time. We will continue further in this direction and use the Reading Machine to predict saccades.

The strategies implemented in the RM described in this paper completely specify the order in which predictions are made. We plan to relax this constraint and let the RM learn strategies that optimize its performances. A RM would have the ability to decide the order in which to perform some tasks (specifically, POS tagging, lemmatization and morphological analysis).

8 Acknowledgments

This work was granted access to the HPC resources of IDRIS under the allocation 2020-AD011011708 made by GENCI.

References

Chris Alberti, David Weiss, Greg Coppola, and Slav Petrov. 2015. Improved transition-based parsing and tagging with neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1354–1359.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465. Association for Computational Linguistics.

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

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, Berlin, Germany. Association for Computational Linguistics.

Franck Dary, Abdellah Fourtassi, and Alexis Nasr. 2021a. On the role of low-level linguistic tasks for reading time prediction. In *Proceedings of the 43th Annual Meeting of the Cognitive Science Society.* In press.

Franck Dary, Alexis Nasr, and Abdellah Fourtassi. 2021b. TALEP at CMCL 2021 shared task: Non linear combination of low and high-level features for predicting eye-tracking data. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*, pages 108–113, Online. Association for Computational Linguistics.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976.

John Hale. 2017. Models of human sentence comprehension in computational psycholinguistics.

Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2012. Incremental joint approach to word segmentation, POS tagging, and dependency parsing in Chinese. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1045–1053, Jeju Island, Korea. Association for Computational Linguistics.

Matthew Honnibal and Mark Johnson. 2014. Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics*, 2:131–142.

Frank Keller. 2010. Cognitively plausible models of human language processing. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 60–67. Association for Computational Linguistics.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Philipp Koehn. 2004. Statistical significance tests for machine translation evaluation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 388–395, Barcelona, Spain. Association for Computational Linguistics.

Shuhei Kurita, Daisuke Kawahara, and Sadao Kurohashi. 2017. Neural joint model for transition-based Chinese syntactic analysis. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1204–1214, Vancouver, Canada. Association for Computational Linguistics.

John SY Lee, Jason Naradowsky, and David A Smith. 2011. A discriminative model for joint morphological disambiguation and dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human language technologies*, pages 885–894.

Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011. Joint models for Chinese POS tagging and dependency parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1180–1191, Edinburgh, Scotland, UK. Association for Computational Linguistics.

George W McConkie and Keith Rayner. 1975. The span of the effective stimulus during a fixation in reading. *Perception & Psychophysics*, 17(6):578–586.

George W McConkie and Keith Rayner. 1976. Asymmetry of the perceptual span in reading. *Bulletin of the psychonomic society*, 8(5):365–368.

Alexis Nasr, Carlos Ramisch, José Deulofeu, and André Valli. 2015. Joint dependency parsing and multiword expression tokenisation. In *Annual Meeting of the Association for Computational Linguistics*, pages 1116–1126.

Dat Quoc Nguyen and Karin Verspoor. 2018. An improved neural network model for joint POS tagging and dependency parsing. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 81–91, Brussels, Belgium. Association for Computational Linguistics.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the eighth international conference on parsing technologies*, pages 149–160.

Joakim Nivre. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359.

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.

Martin Popel, Zdeněk Žabokrtskỳ, and Martin Vojtek. 2017. Udapi: Universal api for universal dependencies. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 96–101.

Hui Wan, Tahira Naseem, Young-Suk Lee, Vittorio Castelli, and Miguel Ballesteros. 2018. Ibm research at the conll 2018 shared task on multilingual parsing. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 92–102.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206. Nancy, France.

Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2020. A Graph-based Model for Joint Chinese Word Segmentation and Dependency Parsing. *Transactions of the Association for Computational Linguistics*, 8:78–92.

Masashi Yoshikawa, Hiroyuki Shindo, and Yuji Matsumoto. 2016. Joint transition-based dependency parsing and disfluency detection for automatic speech recognition texts. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1036–1041, Austin, Texas. Association for Computational Linguistics.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

Daniel Zeman et al. 2019. Universal dependencies 2.5. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Yuan Zhang, Chengtao Li, Regina Barzilay, and Kareem Darwish. 2015. Randomized greedy inference for joint segmentation, POS tagging and dependency parsing. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 42–52, Denver, Colorado. Association for Computational Linguistics.