Dependency parsing

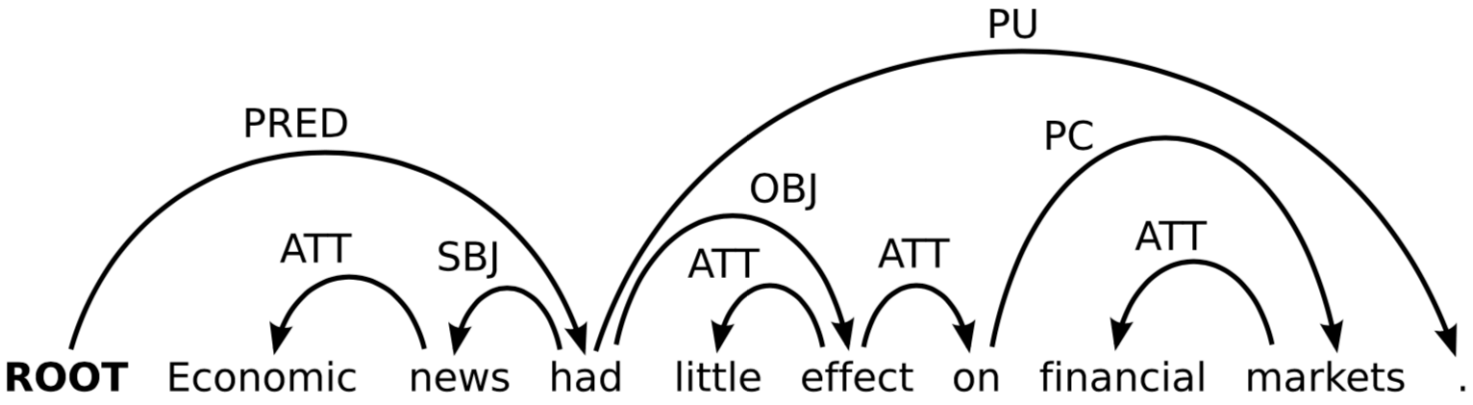
Benoît Sagot
Inria (ALMAnaCH)

MVA — Speech and Language Processing — Class #6 — 4th March, 2019

Credit and disclaimer: some of the following slides are taken from, illustrated or inspired by presentations and article figures by Nivre, Dyer, Ballesteros, Mooney, Rasooli and Tetreault

Introduction

- **Syntactic parsing of natural language**
  - Building the structure of natural language sentences
- **Dependency-based syntactic representations**
  - Long tradition in descriptive and theoretical linguistics
  - Have become popular in computational linguistics



Strategies for dependency parsing

- Graph-based parsing
- Transition-based parsing
- Other strategies

Graph-based parsing

- MSTParser (McDonald et al. 2005)
  - http://www.seas.upenn.edu/~strctlrn/MSTParser/MSTParser.html
- Simplified version of the underlying idea:
  - Create all possible dependencies
  - Weigh them
  - Extract the optimal dependency tree
    - I.e. the tree that covers all words and minimises the overall weight of all retained dependencies

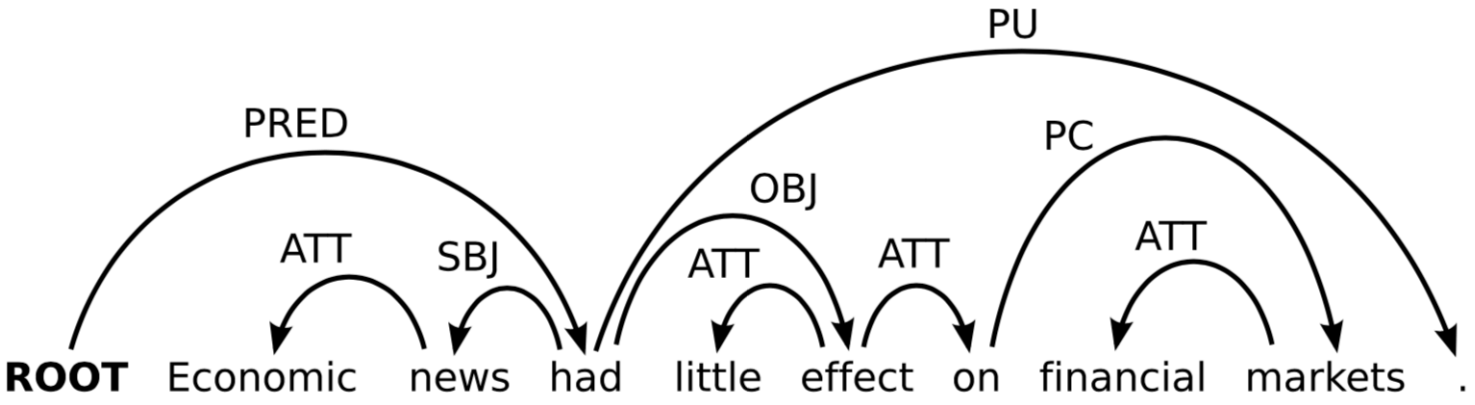
Arc-standard Transition-Based Parsing

Starting point

- **The basic idea:**
  - Define a transition system for dependency parsing
  - Learn a model for scoring possible transitions
  - Parse by searching for the optimal transition sequence
- **Advantages:**
  - Highly efficient parsing with low complexity
  - Rich history-based feature models for disambiguation
- Cf. Nivre (et al.)
  - http://www.maltparser.org

Formalising dependency trees

- A **dependency tree** is a **labelled directed tree** *T* with
  - a set *V* of nodes, labelled with wordforms (including the special "wordform" **ROOT**)
  - a set *A* of arcs, labelled with dependency types
  - a linear precedence order < on *V*
- **Notation:**
  - Arc (w_i, *l*, w_j) connects head w_i to dependent w_j with label *l*
  - Node w_0 (labeled **ROOT**) is the unique root of the tree



Parser configurations

- A **parser configuration** is a triple $c = (S, Q, A)$, where
  - S = a stack $[\dots, w_i]_S$ of partially processed nodes,
  - Q = a queue $[w_j, \dots]_Q$ of remaining input nodes,
  - A = a set of labelled arcs (w_i, l, w_j).
- **Initialisation**:

  $([w_0]_S, [w_1, \dots, w_n]_Q, \{\})$

  (recall that w_0 = **ROOT**)
- **Termination**: $([w_0]_S, []_Q, A)$

Transitions for the “arc-standard algorithm”

- **Left-Arc**(l)

$$\frac{([\ldots, w_i, w_j]_S, Q, A)}{([\ldots, w_j]_S, Q, A \cup \{(w_j, l, w_i)\})} \quad [i \neq 0]$$

- **Right-Arc**(l)

$$\frac{([\ldots, w_i, w_j]_S, Q, A)}{([\ldots, w_i]_S, Q, A \cup \{(w_i, l, w_j)\})}$$

- **Shift**

$$\frac{([\ldots]_S, [w_i, \ldots]_Q, A)}{([\ldots, w_i]_S, [\ldots]_Q, A)}$$

Example Transition Sequence

$[\text{ROOT}]_S$ [Economic, news, had, little, effect, on, financial, markets, .]$_Q$

ROOT Economic news had little effect on financial markets .

Example Transition Sequence

$[\text{ROOT, Economic}]_S$ [news, had, little, effect, on, financial, markets, .]$_Q$

action: Shift

ROOT Economic news had little effect on financial markets .

Example Transition Sequence

[ROOT, Economic, news]$_S$ [had, little, effect, on, financial, markets, .]$_Q$

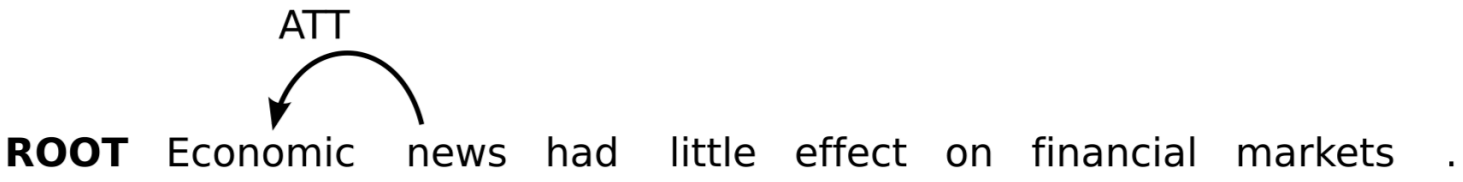
action: Shift

ROOT Economic news had little effect on financial markets .

Example Transition Sequence

[ROOT, Economic, news]$_S$ [had, little, effect, on, financial, markets, .]$_Q$

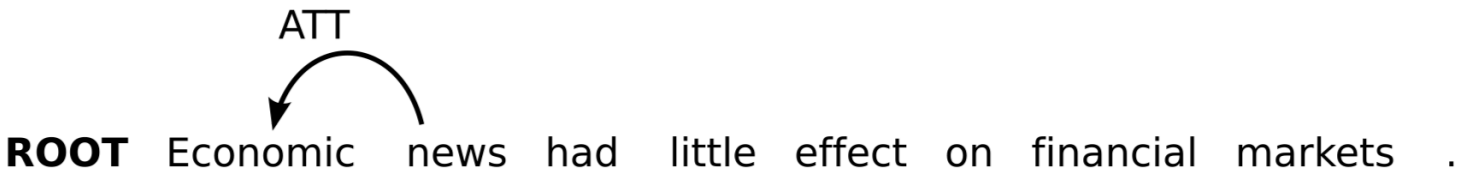
action: Left-Arc(ATT)



Example Transition Sequence

$[\text{ROOT, news, had}]_S\ [\text{little, effect, on, financial, markets, .}]_Q$

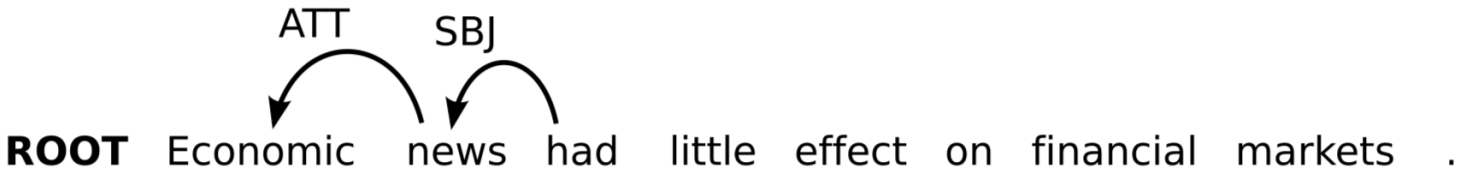
action: Shift



Example Transition Sequence

[ROOT, news, had]$_S$ [little, effect, on, financial, markets, .]$_Q$

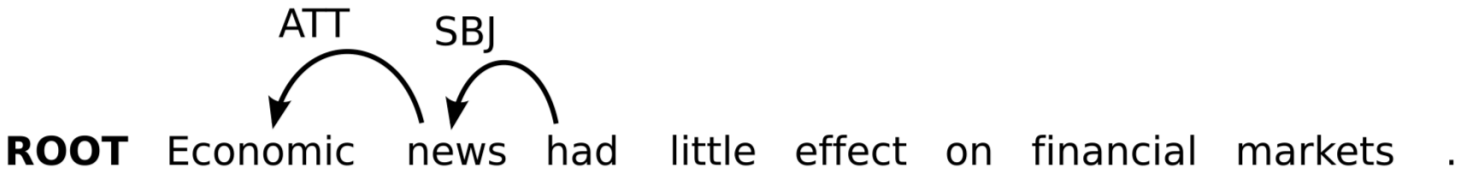
action: Left-Arc(SBJ)



Example Transition Sequence

$[\text{ROOT, had, little}]_S\ [\text{effect, on, financial, markets, .}]_Q$

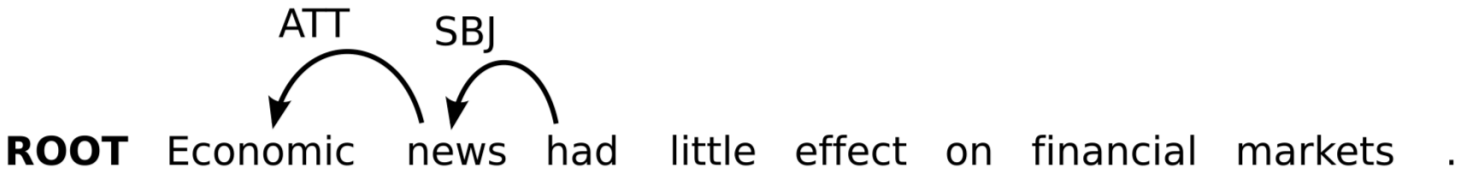
action: Shift



Example Transition Sequence

$[\text{ROOT, had, little, effect}]_S\ [\text{on, financial, markets, .}]_Q$

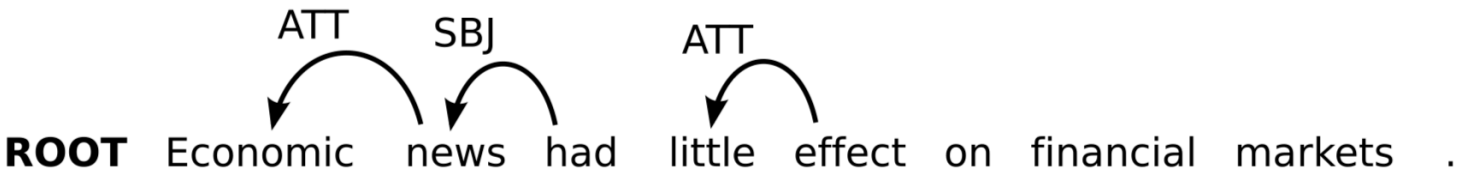
action: Shift



Example Transition Sequence

[ROOT, had, little, effect]$_S$ [on, financial, markets, .]$_Q$

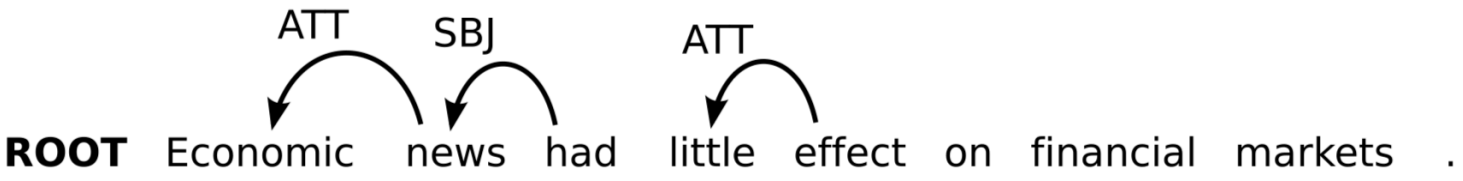
action: Left-Arc(ATT)



Example Transition Sequence

$[\text{ROOT, had, effect, on}]_S\ [\text{financial, markets, .}]_Q$

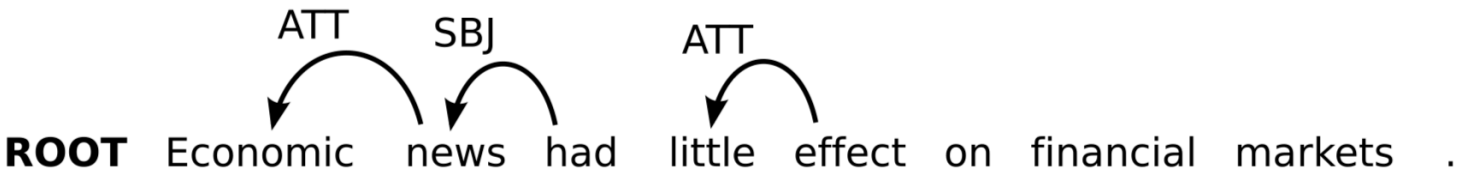
action: Shift



Example Transition Sequence

$[\text{ROOT, had, effect, on, financial}]_S\ [\text{markets, .}]_Q$

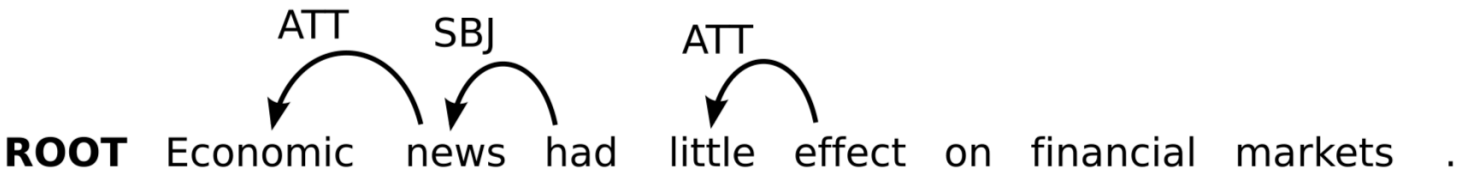
action: Shift



Example Transition Sequence

$[\text{ROOT}, \text{had}, \text{effect}, \text{on}, \text{financial}, \text{markets}]_S\ [.]_Q$

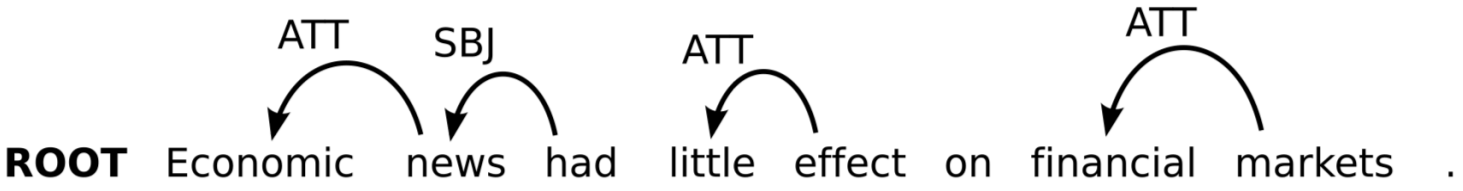
action: Shift



Example Transition Sequence

[ROOT, had, effect, on, financial, markets]$_S$ [.]$_Q$

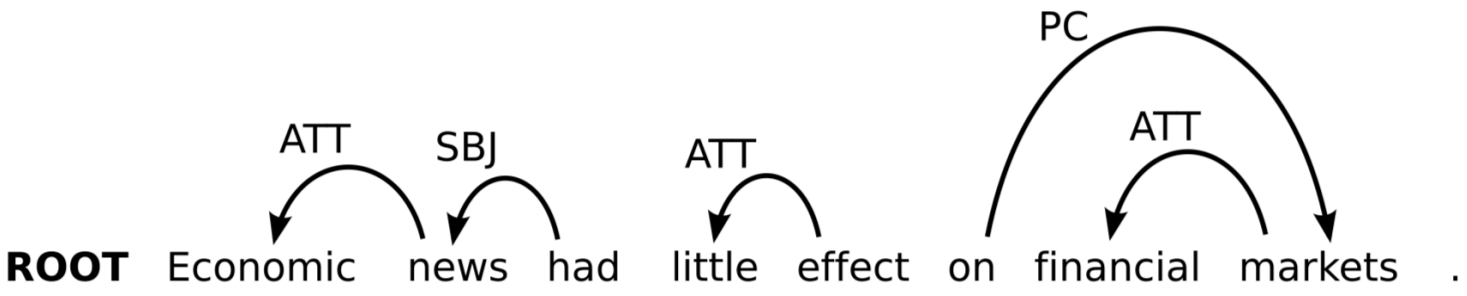
action: Left-Arc(ATT)



Example Transition Sequence

[ROOT, had, effect, on, markets]$_S$ [.]$_Q$

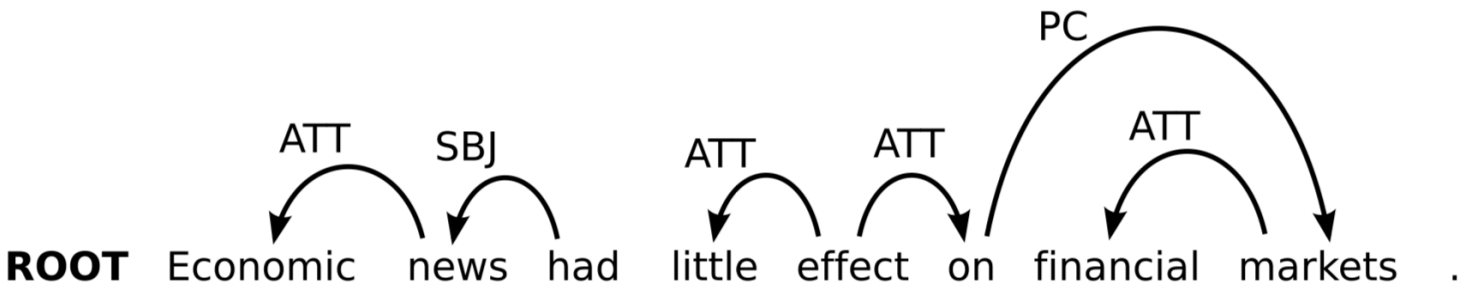
action: Right-Arc(PC)



Example Transition Sequence

[ROOT, had, effect, on]$_S$ [.]$_Q$

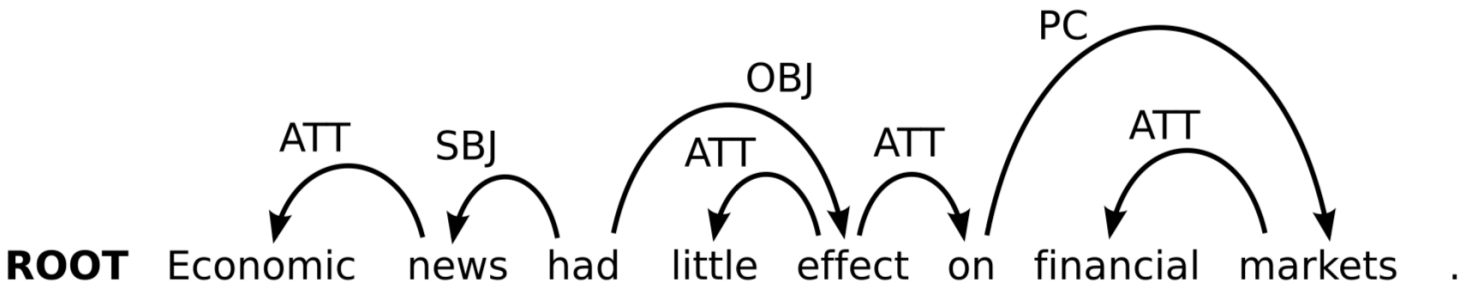
action: Right-Arc(ATT)



Example Transition Sequence

$[\text{ROOT}, \text{had}, \text{effect}]_S\ [.]_Q$

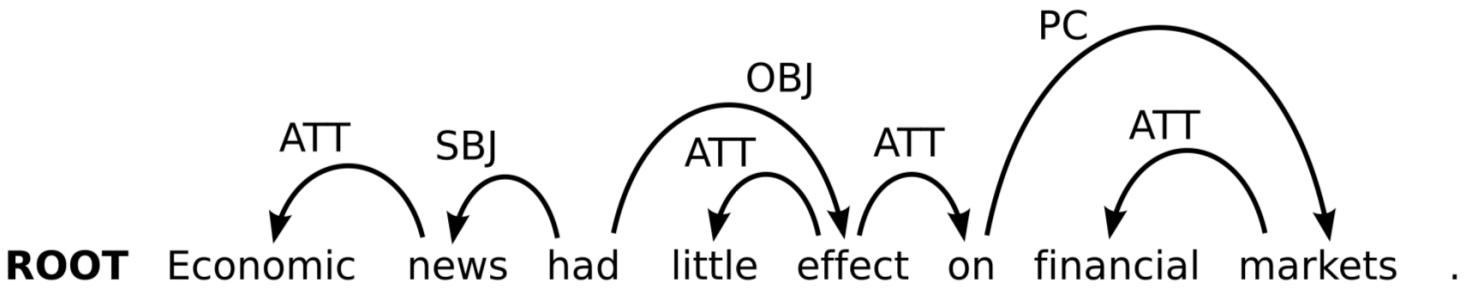
action: Right-Arc(OBJ)



Example Transition Sequence

$[\text{ROOT}, \text{had}, .]_S\ []_Q$

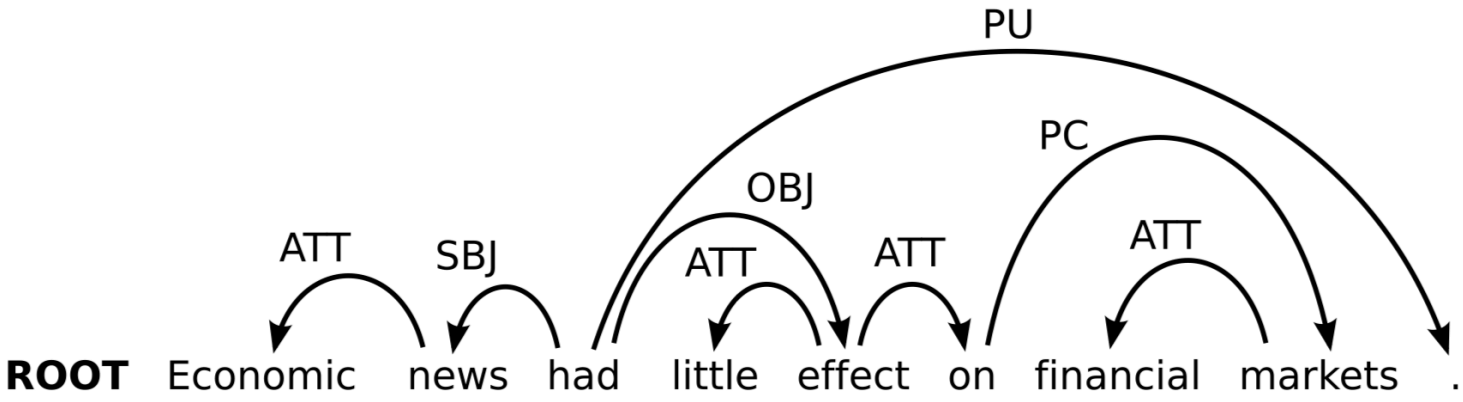
action: Shift



Example Transition Sequence

$[\text{ROOT}, \text{had}, .]_S\ []_Q$

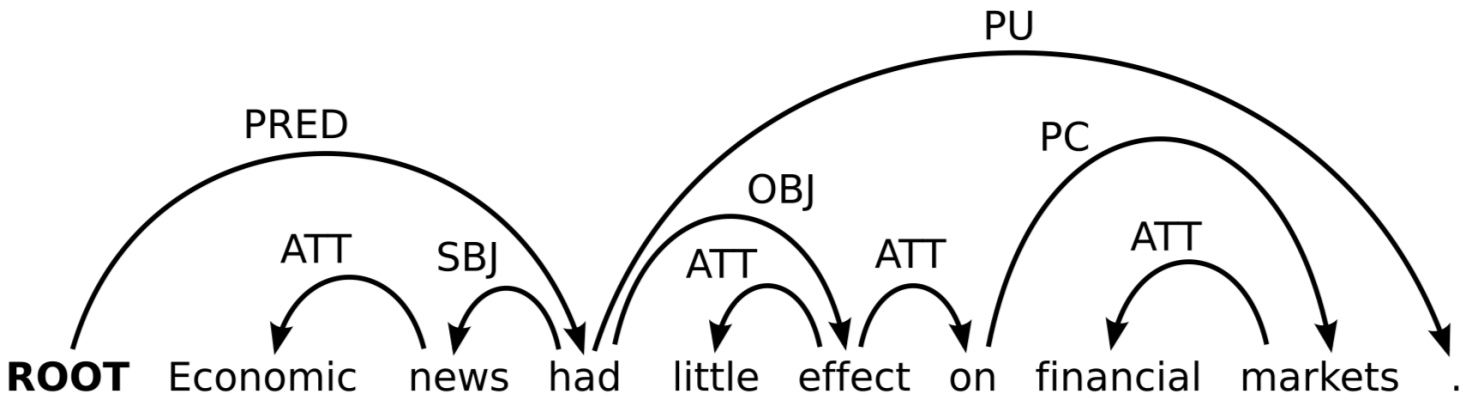
action: Right-Arc(PU)



Example Transition Sequence

$[\text{ROOT}, \text{had}]_S\ []_Q$

action: Right-Arc(PRED)



Properties of the algorithm

- Every transition sequence outputs a projective dependency tree (**soundness**).
- Every projective dependency tree is output by some transition sequence (**completeness**).
- There are exactly $2n$ transitions in a sentence with n words.

Deterministic parsing

- If we have an **oracle** that correctly predicts the next transition *o(c)*, parsing is deterministic:

PARSE($w_1, \ldots, w_n$)
1 $c \leftarrow ([w_0]_S, [w_1, \ldots, w_n]_Q, \{\})$
2 **while** $Q_c \neq [\,]$ **or** $|S_c| > 1$
3 $t \leftarrow o(c)$
4 $c \leftarrow t(c)$
5 **return** $T = (\{w_0, w_1, \ldots, w_n\}, A_c)$

Oracles as classifiers

- An oracle can be approximated by a (linear) **classifier**:

$$o(c) = \text{argmax}_t \, \mathbf{w} \cdot \mathbf{f}(c, t)$$

- History-based feature representation $\mathbf{f}(c, t)$:
  - Features over input tokens relative to S and Q
  - Features over the (partial) dependency tree defined by A
  - Features over the (partial) transition sequence
- Weight vector **w** learned from treebank data:
  - Reconstruct oracle transition sequence for each sentence
  - Construct training data set $D = \{(c, t) \mid o(c) = t\}$
  - Maximise accuracy of local predictions $o(c) = t$

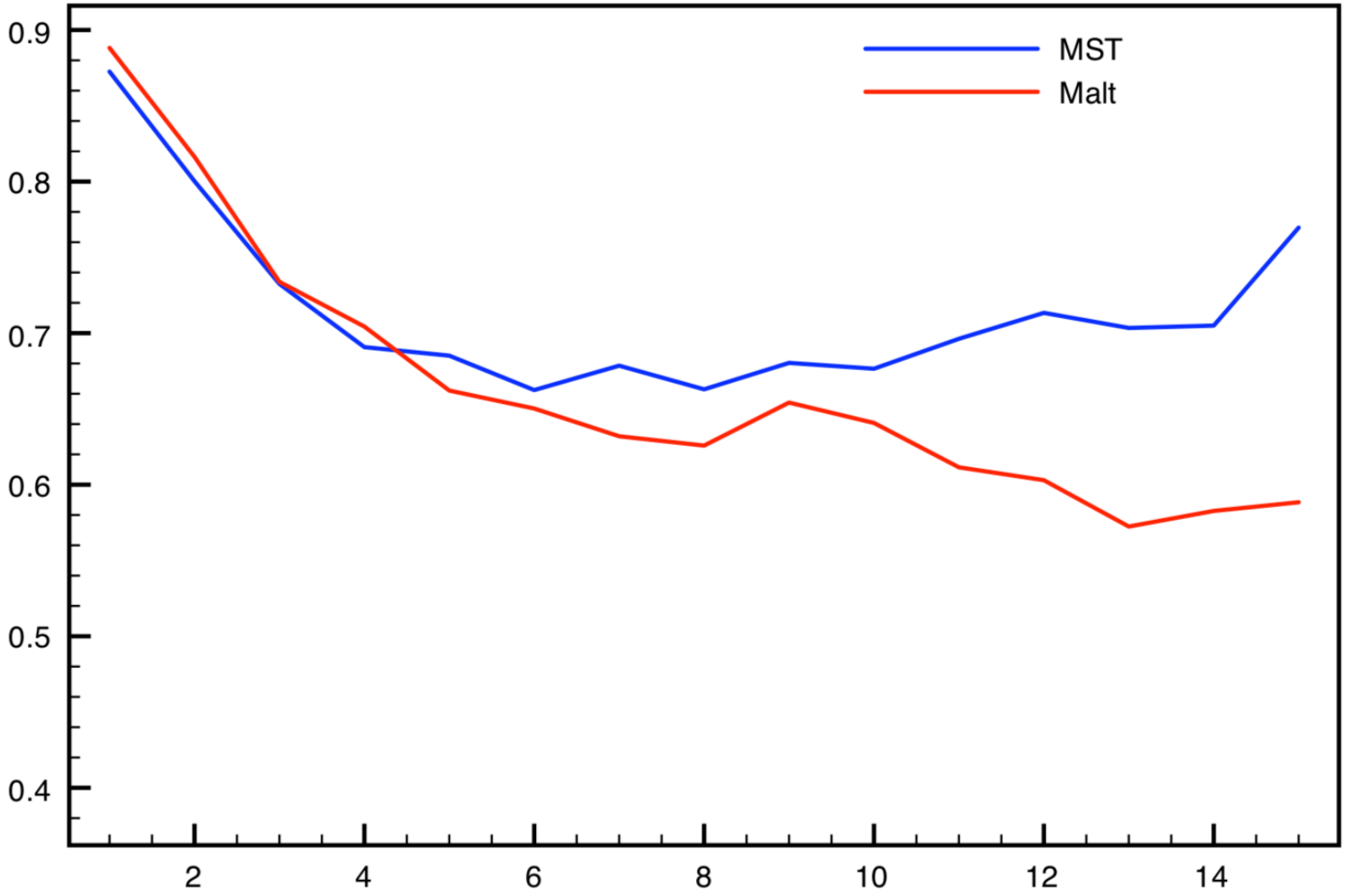
Deterministic classifier-based parsing

- Advantages:
  - **Highly efficient parsing** – linear time complexity with constant time oracles and transitions
  - **Rich history-based feature representations** – no rigid constraints from inference algorithm
- Drawback:
  - Sensitive to **search errors** and **error propagation** due to deterministic parsing and local learning

Empirical results: the CoNLL 2006 shared task

- CoNLL 2006 shared task (Buchholz and Marsi 2006):
  - **MaltParser** (Nivre et al. 2006) – transition-based, deterministic, local learning
  - **MSTParser** (McDonald et al. 2006) – graph-based, exact, global learning
  - Same average parsing accuracy over 13 languages
- Comparative **error analysis** (McDonald and Nivre2007):
  - MaltParser more accurate on short dependencies and disambiguation of core grammatical functions
  - MSTParser more accurate on long dependencies and dependencies near the root of the tree
- Hypothesised **explanation for MaltParser results**:
  - **Rich features counteracted by error propagation**

Precision by dependency length



Beam search and structured prediction

Beam search

- **Maintain the *k* best hypotheses** (Johansson and Nugues 2006):

PARSE($w_1, \ldots, w_n$)
1 BEAM $\leftarrow \{([w_0]_S, [w_1, \ldots, w_n]_Q, \{\})\}$
2 **while** $\exists c \in$ BEAM $[Q_c \neq [\,]$ **or** $|S_c| > 1]$
3 **foreach** $c \in$ BEAM
4 **foreach** t
5 ADD($t(c)$, NEWBEAM)
6 BEAM $\leftarrow$ TOP(k, NEWBEAM)
7 **return** $T = (\{w_0, w_1, \ldots, w_n\}, A_{\text{TOP}(1, \text{BEAM})})$

- **Note:**
  - Score($c_0, \ldots, c_m$) = $\sum_{i=1}^{m} \mathbf{w} \cdot \mathbf{f}(c_{j-1}, t_j)$
  - Simple combination of locally normalised classifier scores
  - Marginal gains in accuracy

Online question 1

What is the complexity of transition-based parsing extended with beam search?

1. $O(n)$
2. $O(n \log n)$
3. $O(n^2)$

https://api.socrative.com/rc/BbvH97

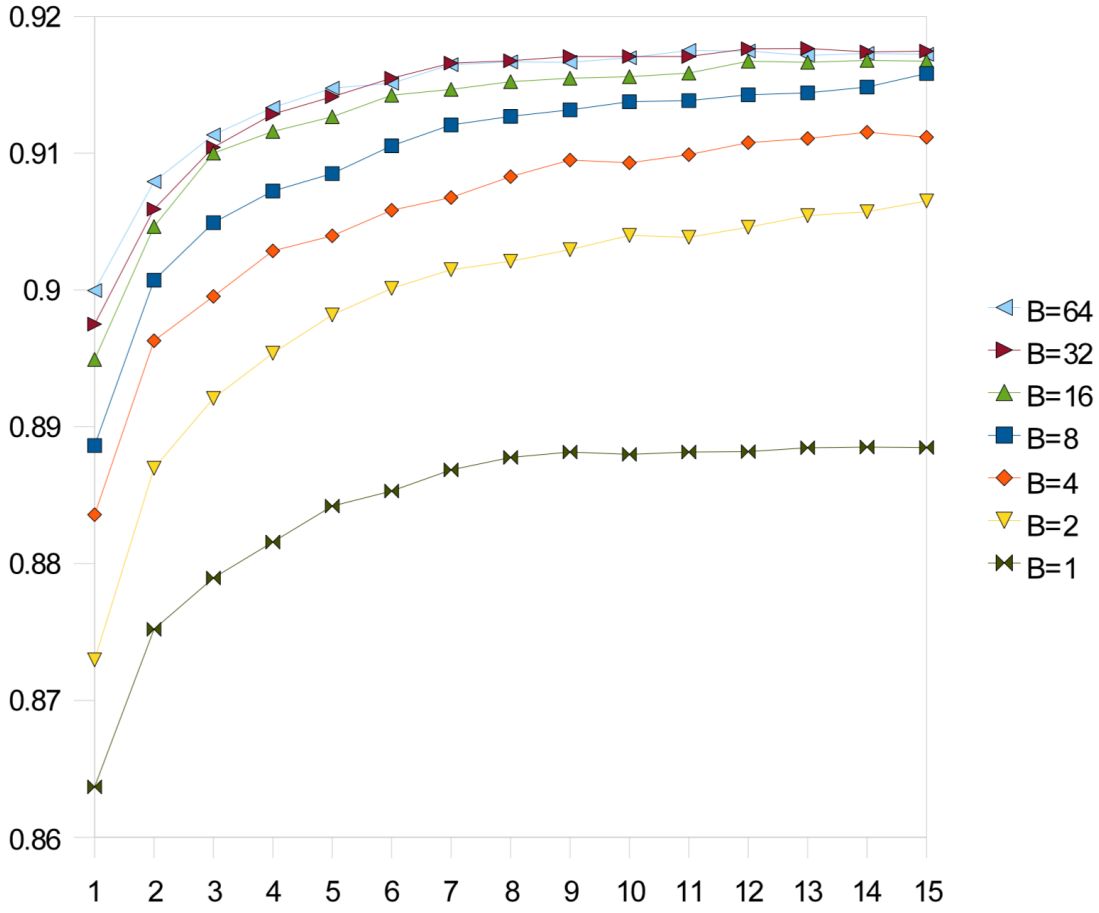
Structured prediction

- **Parsing as structured prediction** (Zhang and Clark 2008)**:**
  - Minimise loss over entire transition sequence
  - Use beam search to find highest-scoring sequence
- Factored feature representations:

$$\mathbf{f}(c_0, \ldots, c_m) = \sum_{i=1}^{m} \mathbf{f}(c_{i-1}, t_i)$$

- Online learning from oracle transition sequences:
  - Structured perceptron (Collins 2002)
  - Early updates (Collins and Roark 2004)

Beam size and training iterations

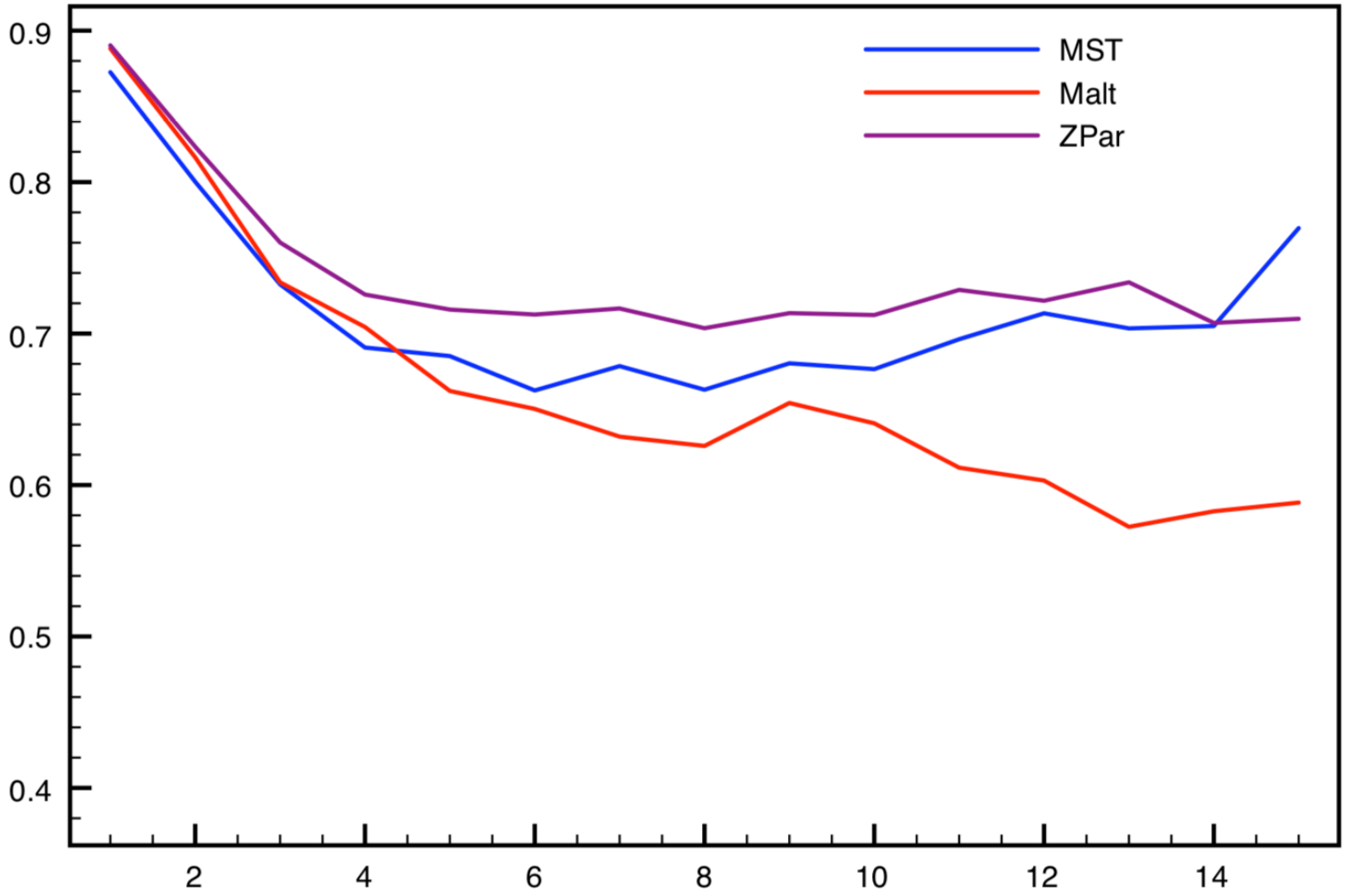


Yue Zhang and Stephen Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-Based and Transition-Based Dependency Parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, 562–571.

The best of two worlds?

- Like graph-based dependency parsing (MSTParser):
  - Global learning – minimise loss over entire sentence
  - Non-greedy search – accuracy increases with beam size
- Like deterministic transition-based parsing (MaltParser):
  - Highly efficient – complexity still linear for fixed beam size
  - Rich features – no constraints from parsing algorithm
- Example ZPar parser (Zhang and Clark 2011)
  - “Most heavily developed for English and Chinese”

Precision by dependency length, again



Even richer feature models

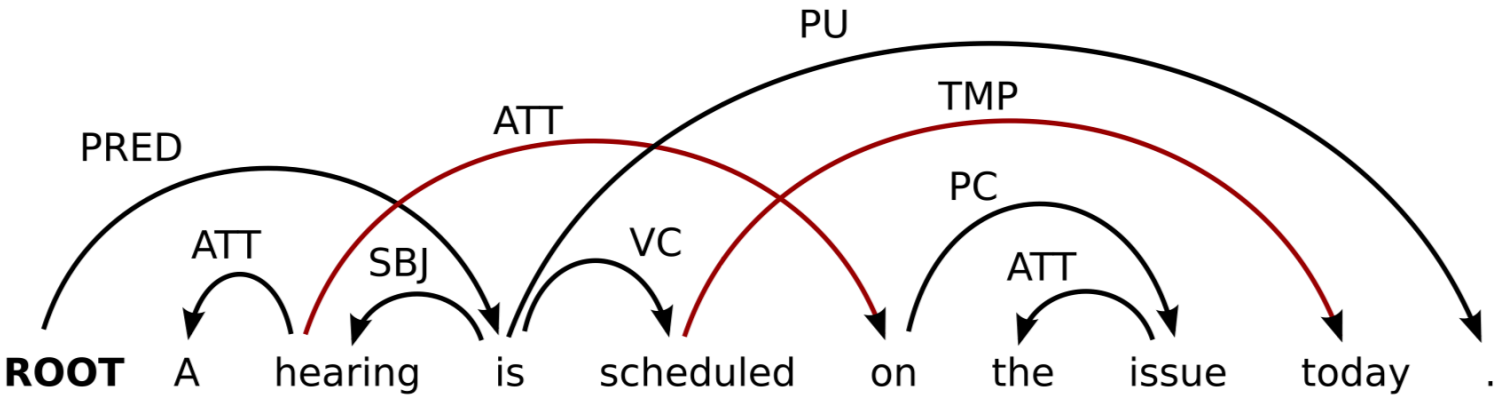
	ZPar	Malt
Baseline	92.18	89.37
+distance	+0.07	–0.14
+valency	+0.24	0.00
+unigrams	+0.40	–0.29
+third-order	+0.18	0.00
+label set	+0.07	+0.06
Extended	93.14	89.00

Yue Zhang and Joakim Nivre. 2011. Transition-Based Dependency Parsing with Rich Non-Local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 188–193.

Online reordering for non-projectivity

Projectivity

- A dependency arc is **projective** if the head (transitively) dominates all intervening words
- Most dependency grammar theories do not assume projectivity (but many parsers do)

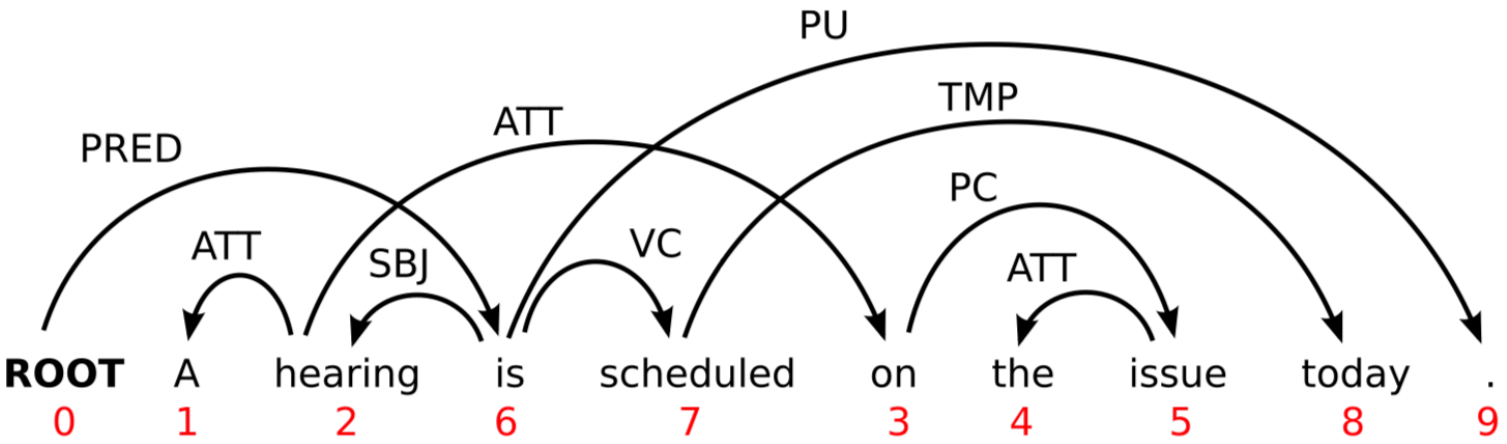


Non-projectivity in natural languages

Language	Trees	Arcs
Arabic (Hajič et al. 2004)	11,2 %	0,4 %
Basque (Aduriz et al. 2003)	26,2 %	2,9 %
Czech (Hajič et al. 2001)	23,2 %	1,9 %
Danish (Kromann 2003)	15,6 %	1,0 %
Greek (Prokopidis et al. 2005)	20,3 %	1,1 %
Russian (Boguslavsky et al. 2000)	10,6 %	0,9 %
Slovene (Džeroski et al. 2006)	22,2 %	1,9 %
Turkish (Oflazer et al. 2003)	11,6 %	1,5 %

Projectivity and word order

- Projectivity is a property of a dependency tree only in relation to a particular word order
  - Words can always be reordered to make the tree projective
  - Given a dependency tree $T = (V, A, <)$, let the projective order $<_p$ be the order defined by an in-order traversal of T with respect to $<$ (Veselá et al. 2004)



Parsing with online reordering

- Add transition for reordering words (Nivre 2009):
  - **Swap**

$$\frac{([\ldots, w_i, w_j]_S, [\ldots]_Q, A)}{([\ldots, w_j]_S, [w_i, \ldots]_Q, A)} \qquad [0 < i < j]$$

- Transition-based parsing with two interleaved processes:
  - Sort words into projective order $<_p$
  - Build dependency tree T by connecting adjacent subtrees
    - *T* is always projective with respect to $<_p$
    - *T* may be non-projective with respect to $<$

Example Transition Sequence

$[\text{ROOT}]_S$ [A, hearing, is, scheduled, on, the, issue, today, .]$_Q$

ROOT A hearing is scheduled on the issue today .

Example Transition Sequence

[ROOT, A]$_S$ [hearing, is, scheduled, on, the, issue, today, .]$_Q$

action: Shift

ROOT A hearing is scheduled on the issue today .

Example Transition Sequence

$[\text{ROOT, A, hearing}]_S$ $[\text{is, scheduled, on, the, issue, today, .}]_Q$

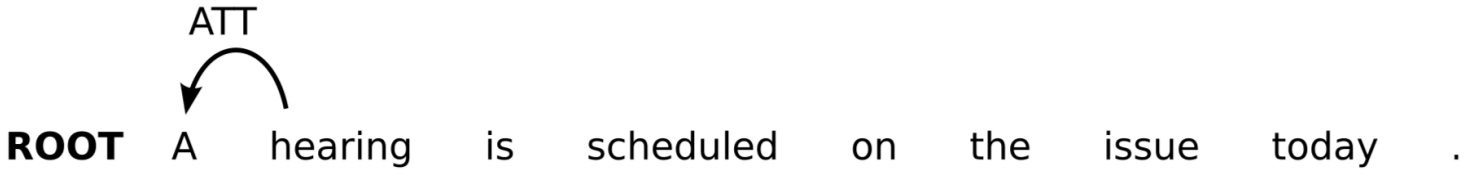
action: Shift

ROOT A hearing is scheduled on the issue today .

Example Transition Sequence

[ROOT, A, hearing]$_S$ [is, scheduled, on, the, issue, today, .]$_Q$

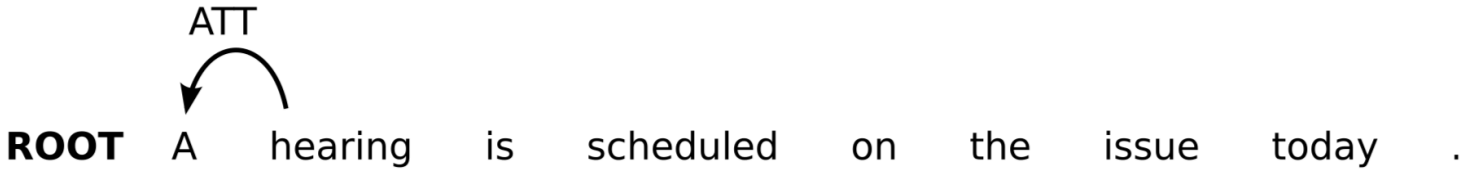
action: Left-Arc(ATT)



Example Transition Sequence

$[\text{ROOT, hearing, is}]_S\ [\text{scheduled, on, the, issue, today, .}]_Q$

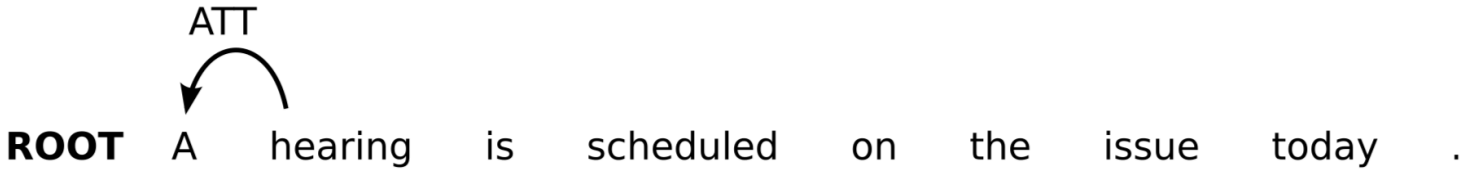
action: Shift



Example Transition Sequence

$[\text{ROOT}, \text{hearing}, \text{is}, \text{scheduled}]_S\ [\text{on}, \text{the}, \text{issue}, \text{today}, .]_Q$

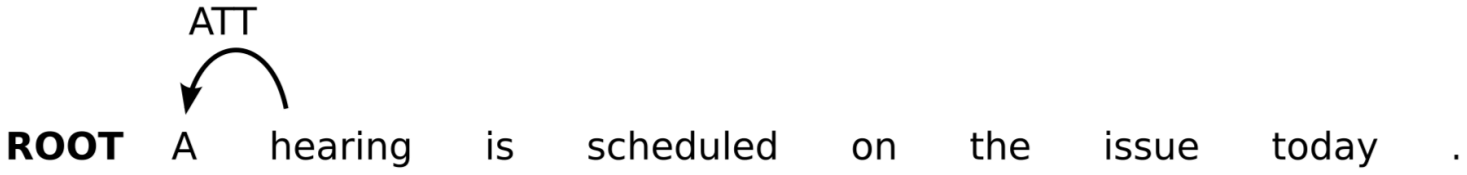
action: Shift



Example Transition Sequence

[ROOT, hearing, is, scheduled, on]$_S$ [the, issue, today, .]$_Q$

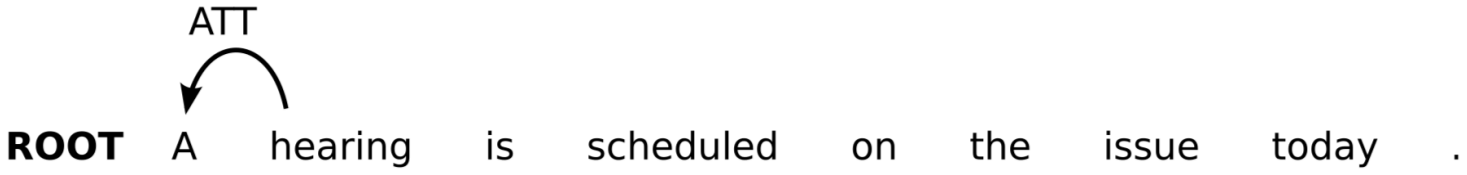
action: Shift



Example Transition Sequence

[ROOT, hearing, is, scheduled, on, the]$_S$ [issue, today, .]$_Q$

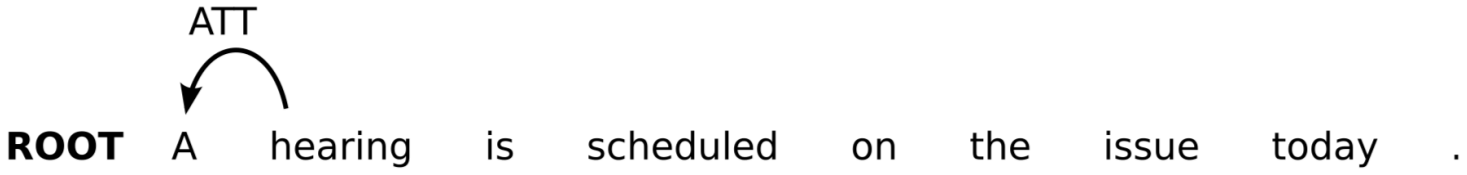
action: Shift



Example Transition Sequence

[ROOT, hearing, is, scheduled, on, the, issue]$_S$ [today, .]$_Q$

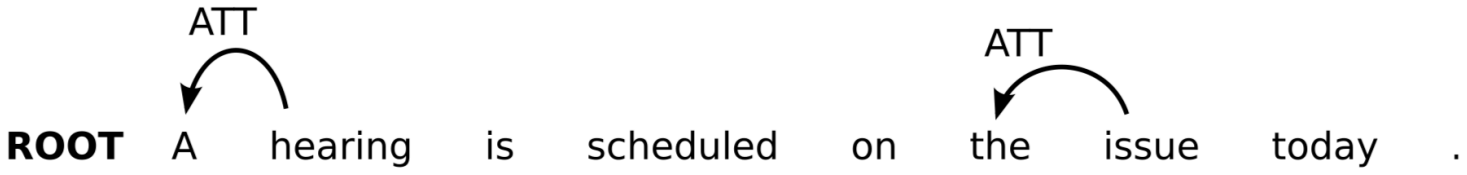
action: Shift



Example Transition Sequence

[ROOT, hearing, is, scheduled, on, the, issue]$_S$ [today, .]$_Q$

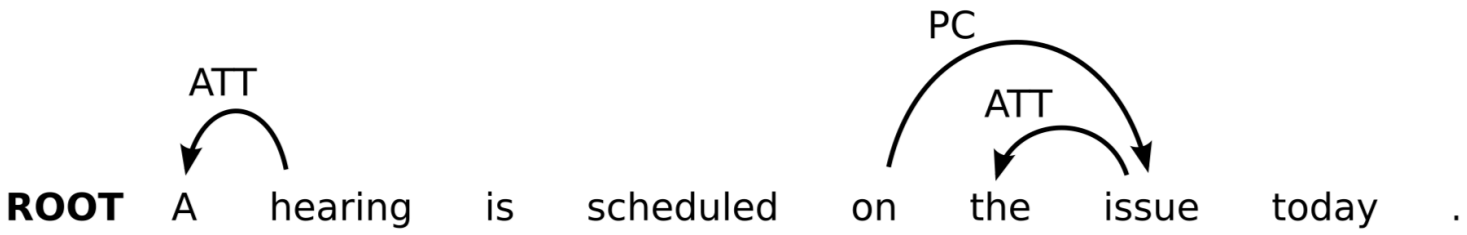
action: Left-Arc(ATT)



Example Transition Sequence

[ROOT, hearing, is, scheduled, on, issue]$_S$ [today, .]$_Q$

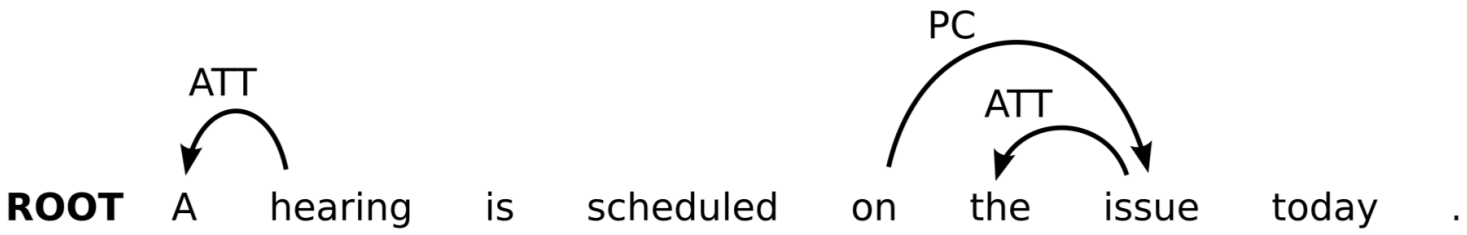
action: Right-Arc(PC)



Example Transition Sequence

[ROOT, hearing, is, on]$_S$ [scheduled, today, .]$_Q$

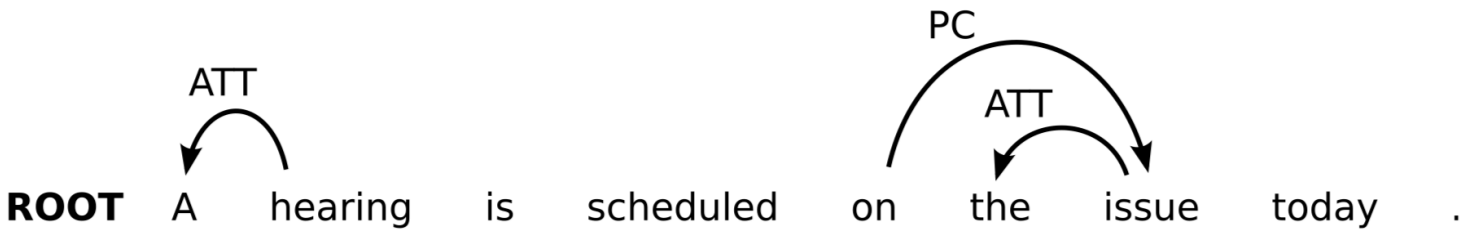
action: Swap



Example Transition Sequence

[ROOT, hearing, on]$_S$ [is, scheduled, today, .]$_Q$

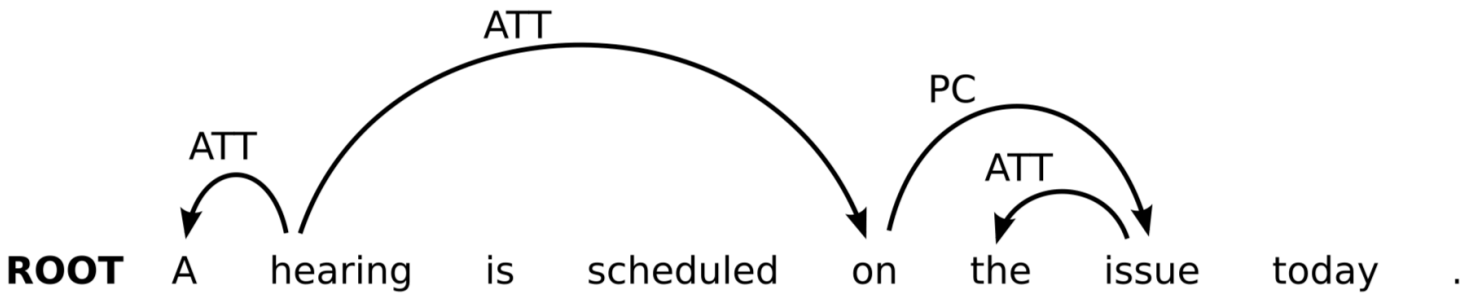
action: Swap



Example Transition Sequence

$[\text{ROOT}, \text{hearing}, \text{on}]_S\ [\text{is}, \text{scheduled}, \text{today}, .]_Q$

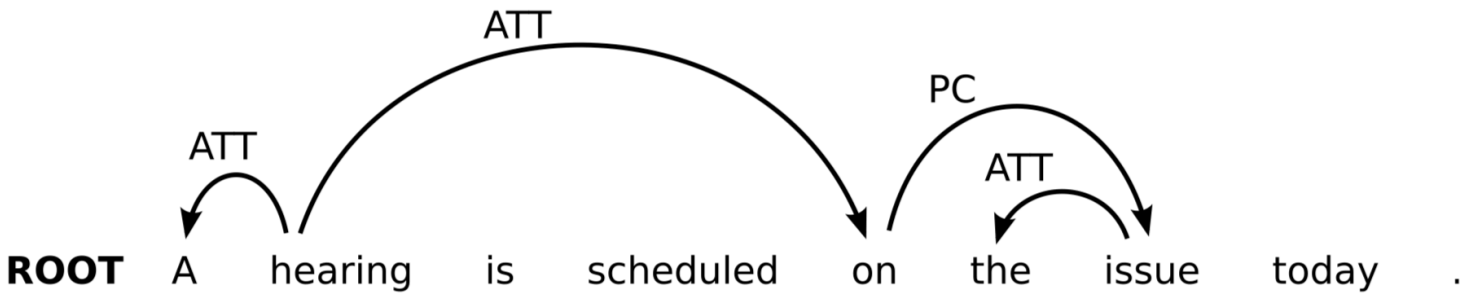
action: Right-Arc(ATT)



Example Transition Sequence

$[\text{ROOT, hearing, is}]_S\ [\text{scheduled, today, .}]_Q$

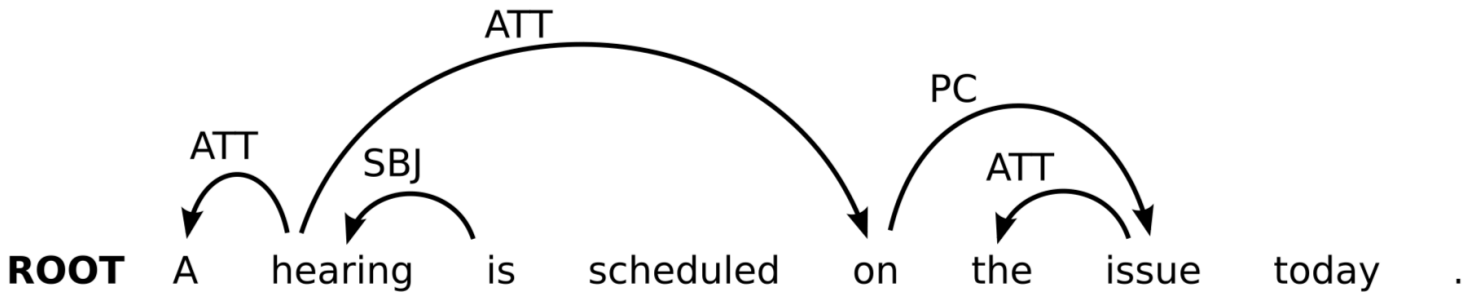
action: Shift



Example Transition Sequence

$[\text{ROOT}, \text{hearing}, \text{is}]_S\ [\text{scheduled}, \text{today}, .]_Q$

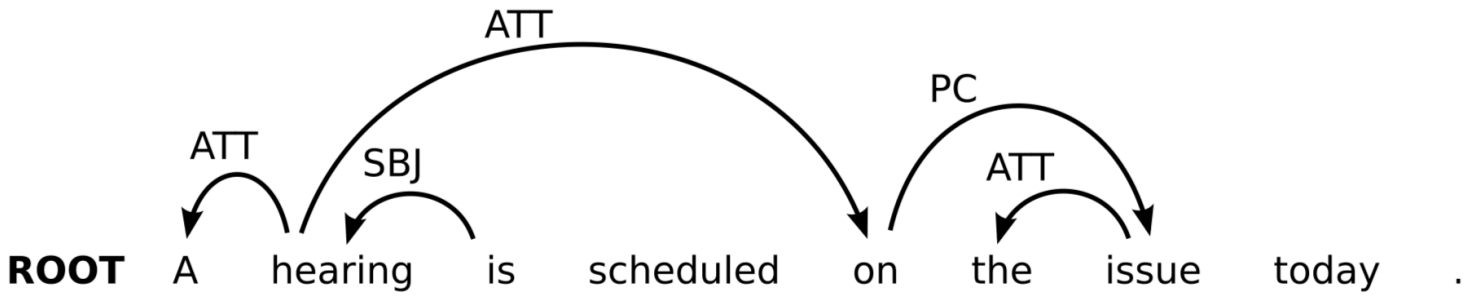
action: Left-Arc(SBJ)



Example Transition Sequence

$[\text{ROOT, is, scheduled}]_S\ [\text{today, .}]_Q$

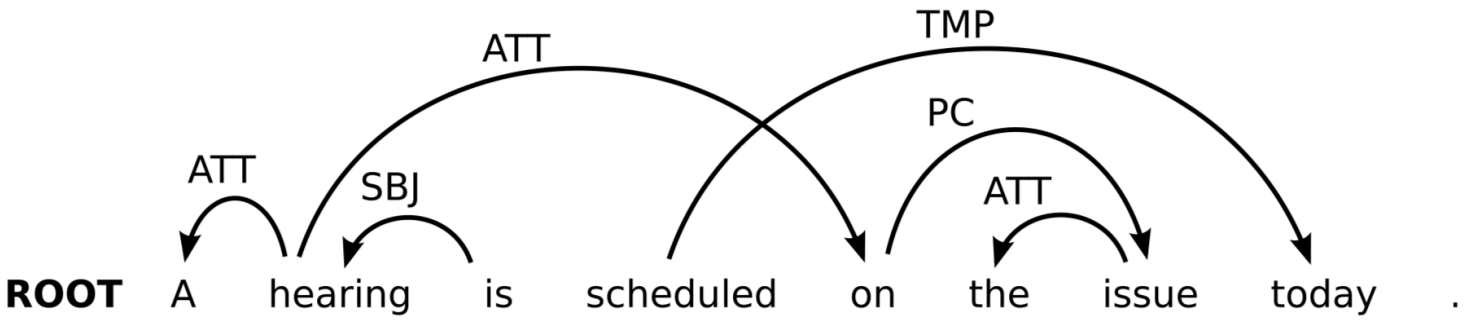
action: Shift



Example Transition Sequence

$[\text{ROOT, is, scheduled, today}]_S\ [.]_Q$

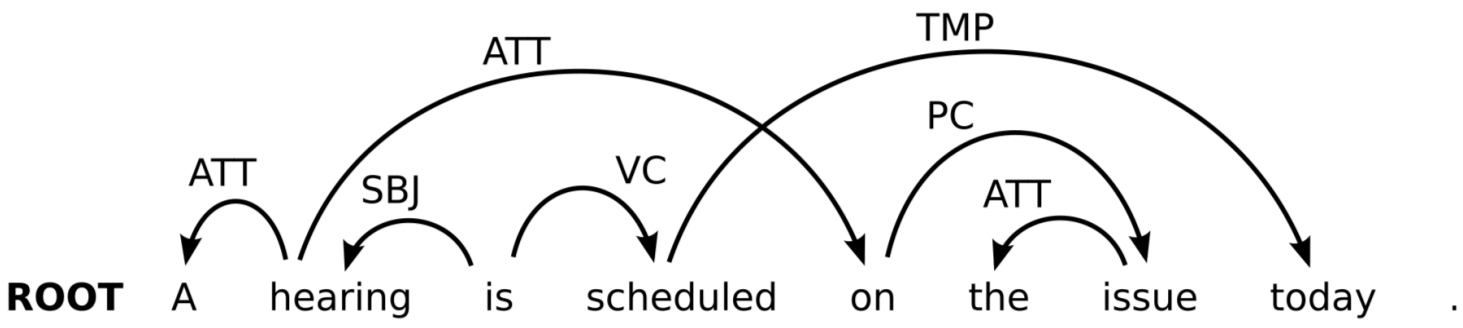
action: Right-Arc(TMP)



Example Transition Sequence

[ROOT, is, scheduled]$_S$ [.]$_Q$

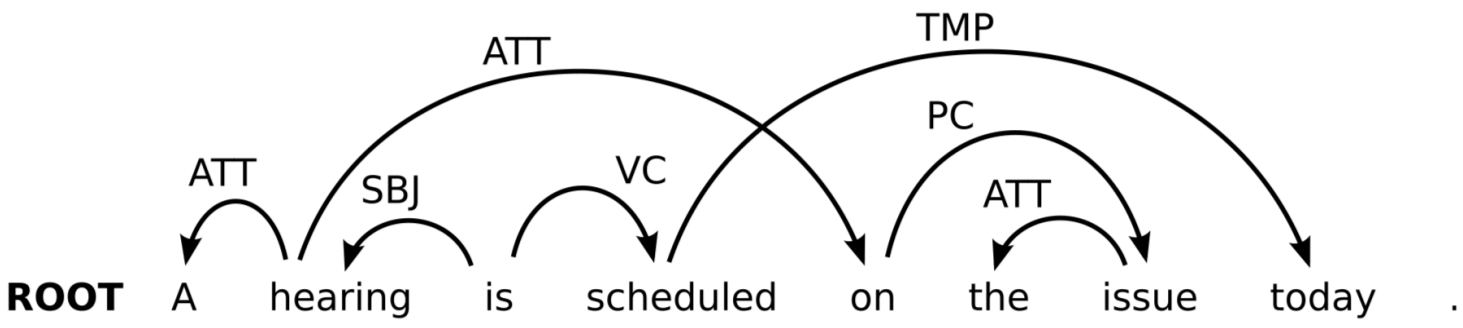
action: Right-Arc(VC)



Example Transition Sequence

$[\text{ROOT}, \text{is}, .]_S\ []_Q$

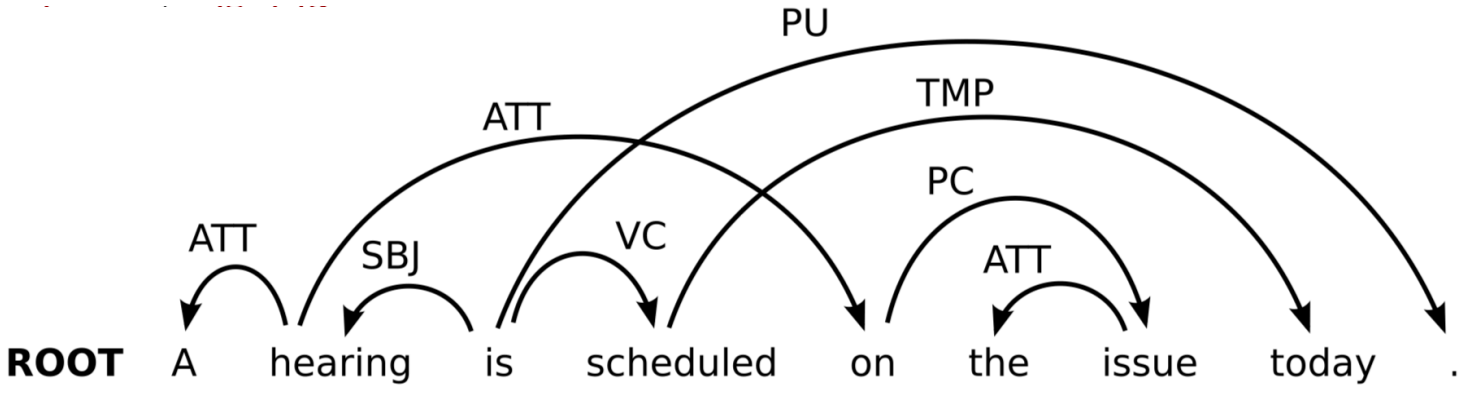
action: Shift



Example Transition Sequence

[ROOT, is, .]$_S$ []$_Q$

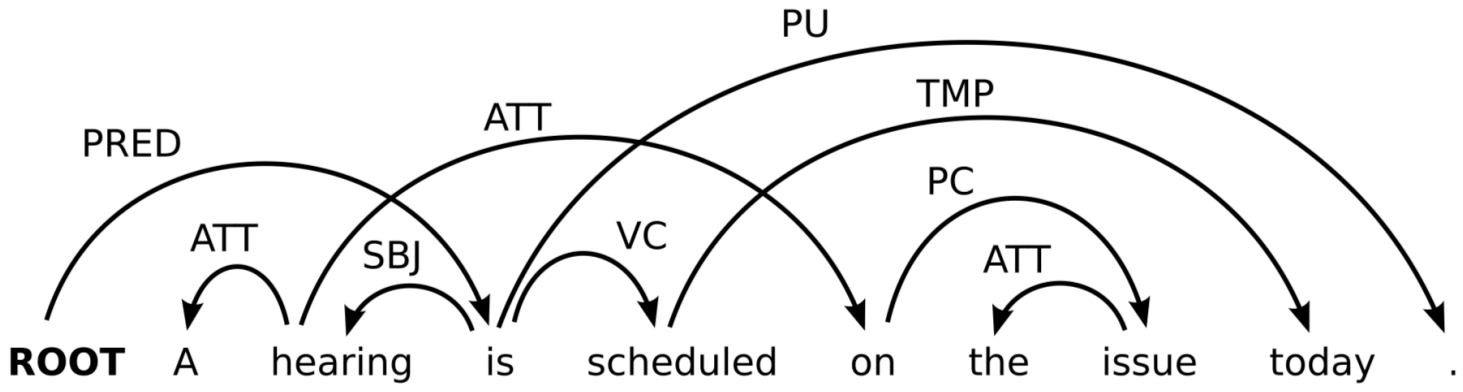
action: Right-Arc(PU)



Example Transition Sequence

$[\text{ROOT}, \text{is}]_S\ []_Q$

action: Right-Arc(PRED)



Empirical results

- Deterministic transition-based parsing (Nivre 2009):
  - Parsing in linear expected time (quadratic worst-case time)
  - Best results on Czech CoNLL 2006 data sets
- Beam search and structured prediction:
  - Evaluation on CoNLL 2009 data sets (dev sets)

	Czech		German	
	LAS	UAS	LAS	UAS
Projective	80.8	86.3	86.2	88.5
Online reordering	83.9	89.1	88.7	90.9

Online question 2

The arc-standard algorithm has several drawbacks. What is this its main drawback?

1. Segmenting the construction of the parse into individual Left-Arc()'s and Right-Arc()'s prevents a correct handling of compound words and idioms
2. The emission of Left-Arc()'s must be performed as soon as possible, which results in decisions being sometimes taken without the appropriate information
3. The emission of Right-Arc()'s is sometimes postponed, which results in decisions being sometimes taken without the appropriate information

https://api.socrative.com/rc/KBcrE4

Arc-eager Transition-Based Parsing

Limitations of the arc-standard algorithm

- The **arc-standard** system considered so far
  - builds a dependency tree strictly bottom-up
  - a dependency arc can only be added between two nodes if the dependent node has already found all its dependents.
  - As a consequence, it is often necessary to postpone the attachment of right dependents.
- This is a problem, as parsing decisions are easier to take when the governor and the governee of a dependency are immediately accessible

The problem of arc-standard on an example

[ROOT]$_S$ [La, température, a, un, très, gros, effet, sur, la, concentration]$_Q$

La température a un très gros effet sur la concentration.

The problem of arc-standard on an example

[ROOT, La]$_S$ [température, a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

La température a un très gros effet sur la concentration.

The problem of arc-standard on an example

[ROOT, La, température]$_S$ [a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

La température a un très gros effet sur la concentration.

The problem of arc-standard on an example

[ROOT, La, température]$_S$ [a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Left-Arc()

The problem of arc-standard on an example

[ROOT, température, a]$_S$ [un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, température, a]$_S$ [un, très, gros, effet, sur, la, concentration]$_Q$

action: Left-Arc()

The problem of arc-standard on an example

[ROOT, a, un]$_S$ [très, gros, effet, sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, a, un, très]$_S$ [gros, effet, sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, a, un, très, gros]$_S$ [effet, sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, a, un, très, gros]$_S$ [effet, sur, la, concentration]$_Q$

action: Right-Arc()

The problem of arc-standard on an example

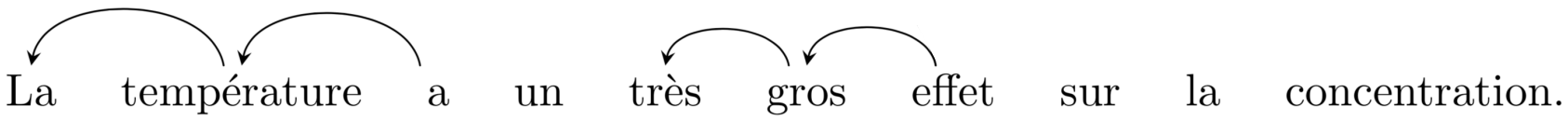
[ROOT, a, un, gros, effet]$_S$ [sur, la, concentration]$_Q$

action: Shift

The problem of arc-standard on an example

[ROOT, a, un, gros, effet]$_S$ [sur, la, concentration]$_Q$

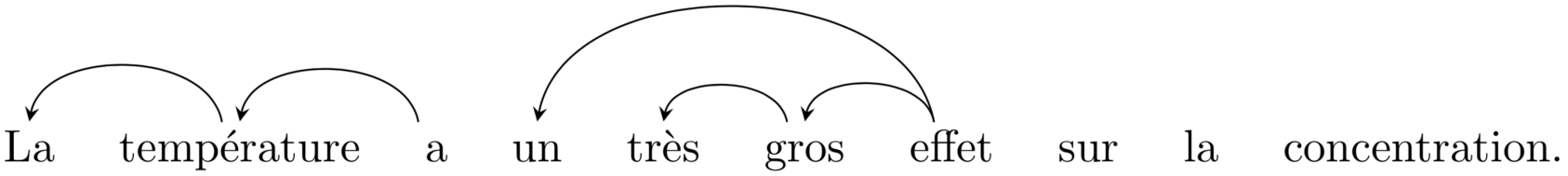
action: Left-Arc()



The problem of arc-standard on an example

[ROOT, a, un, effet]$_S$ [sur, la, concentration]$_Q$

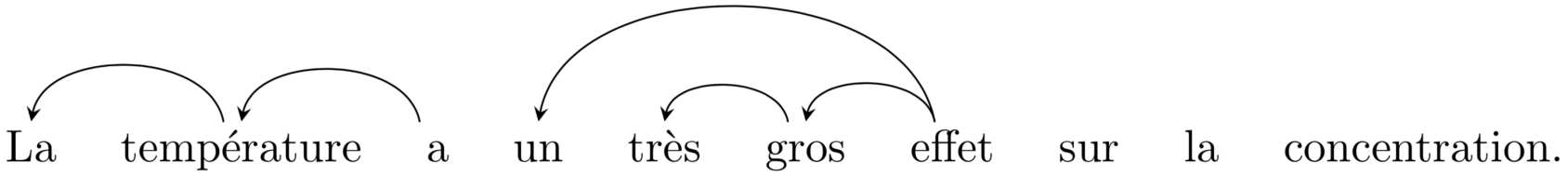
action: Left-Arc()



The problem of arc-standard on an example

$[\text{ROOT, a, effet, sur}]_S$ $[\text{la, concentration}]_Q$

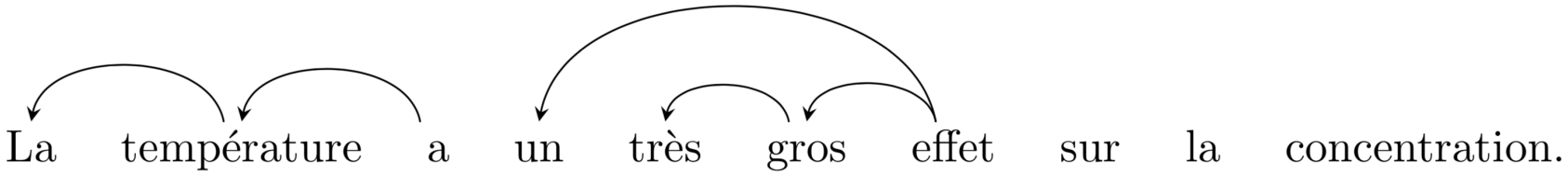
action: Shift



The problem of arc-standard on an example

[ROOT, a, effet, sur, la]$_S$ [concentration]$_Q$

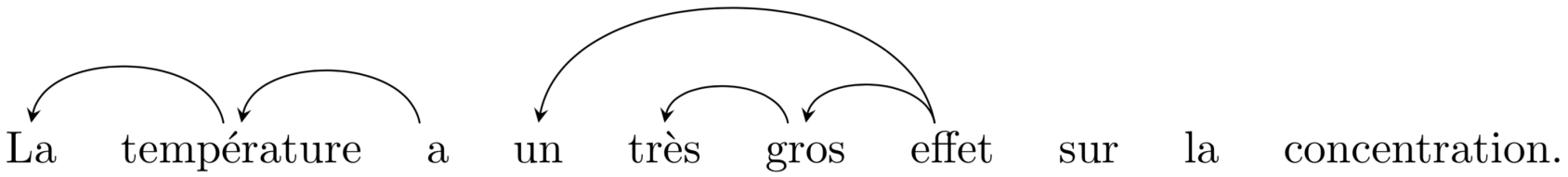
action: Shift



The problem of arc-standard on an example

[ROOT, a, effet, sur, la, concentration]$_S$ []$_Q$

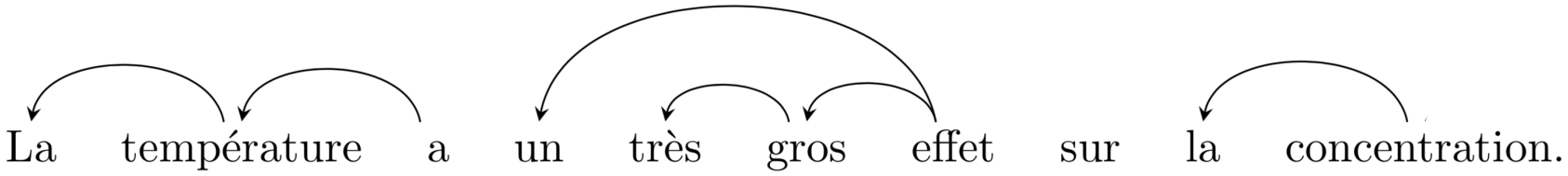
action: Shift



The problem of arc-standard on an example

[ROOT, a, effet, sur, la, concentration]$_S$ []$_Q$

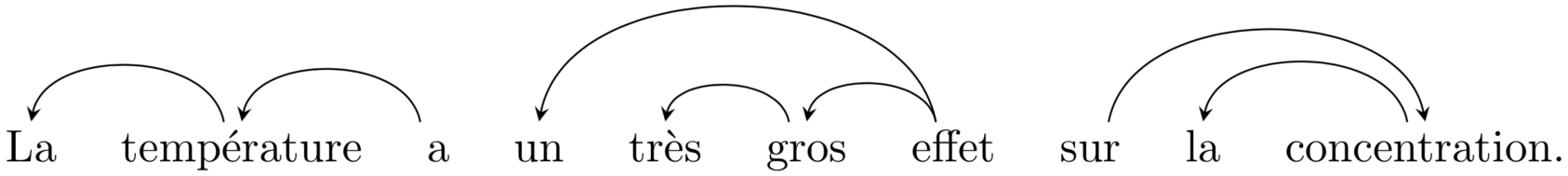
action: Left-Arc()



The problem of arc-standard on an example

[ROOT, a, effet, sur, concentration]$_S$ []$_Q$

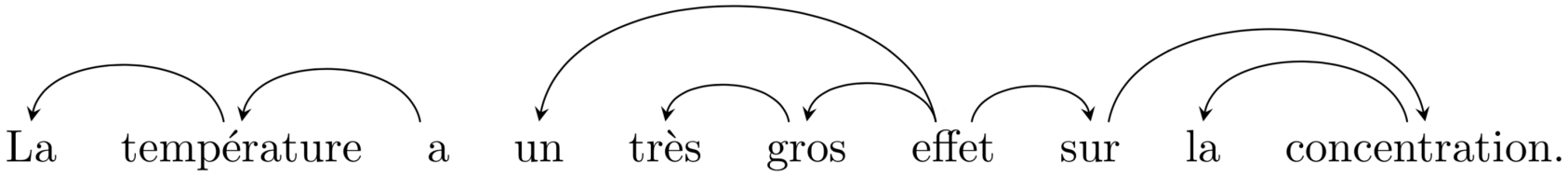
action: Right-Arc()



The problem of arc-standard on an example

[ROOT, a, effet, sur]$_S$ []$_Q$

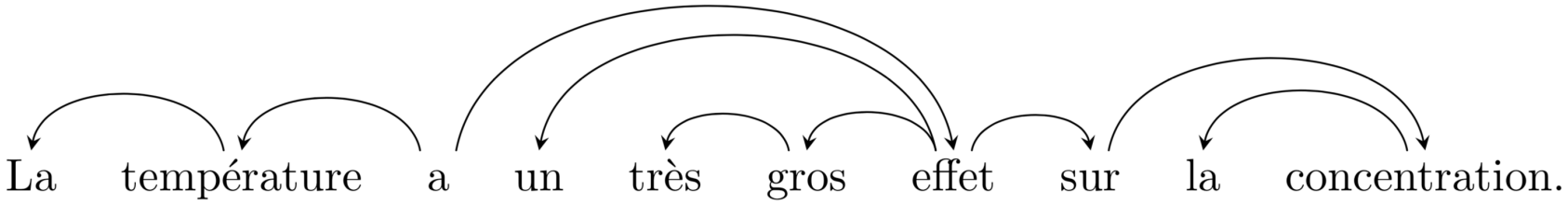
action: Right-Arc()



The problem of arc-standard on an example

[ROOT, a, effet]$_S$ []$_Q$

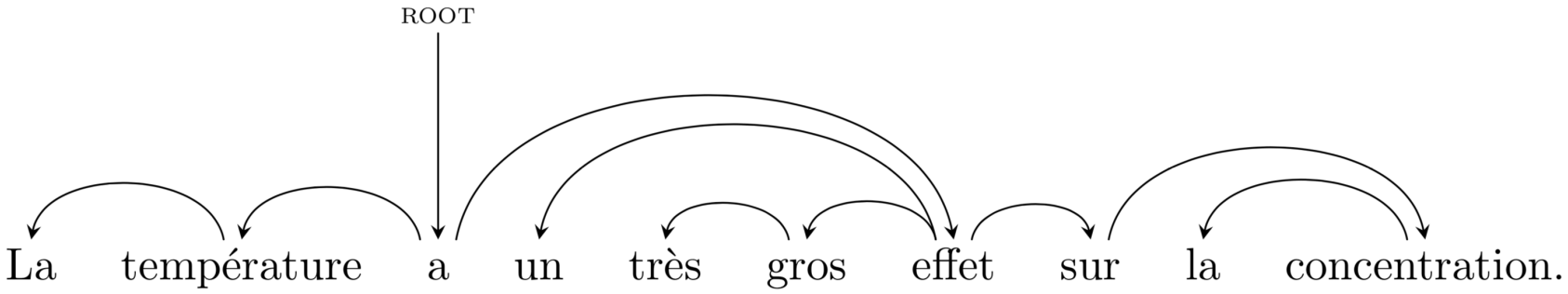
action: Right-Arc()



The problem of arc-standard on an example

[ROOT, a]$_S$ []$_Q$

action: Right-Arc()



The arc-eager system

- We will modify the basic set of actions in order to always add an arc at the earliest possible opportunity:
  - we will now build parts of the tree top-down instead of bottom-up
- Shift remains the same
- **Left-Arc is rewritten and subjected to a stricter condition** (allowed only if the dependent is not the root and has no incoming arcs)

$$\frac{([\ldots, w_i]_S, [w_j, \ldots]_Q, A)}{([\ldots]_S, [w_j, \ldots]_Q, A \cup \{(w_j, l, w_i)\})} \; [i \neq 0 \wedge \nexists(k, l') \mid (k, l', i) \in A]$$

The arc-eager system

- **Right-Arc is changed**: it does not discard w_i anymore:

$$\frac{([\ldots, w_i]_S, [w_j, \ldots]_Q, A)}{([\ldots, w_i, w_j]_S, [\ldots]_Q, A \cup \{(w_i, l, w_j)\})}$$

- We postpone the reduction of w_i to another, new action:
- **Reduction**, only possible if the top of the stack already has a head

$$\frac{([\ldots, w_i]_S, Q, A)}{([\ldots]_S, Q, A)} \quad \text{only if } \exists (k, l') \mid (k, l', i) \in A$$

Arc-eager on an example

[ROOT]$_S$ [La, température, a, un, très, gros, effet, sur, la, concentration]$_Q$

La température a un très gros effet sur la concentration.

Arc-eager on an example

[ROOT, La]$_S$ [température, a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

La température a un très gros effet sur la concentration.

Arc-eager on an example

[ROOT, La]$_S$ [température, a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Left-Arc()

Arc-eager on an example

[ROOT, température]$_S$ [a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Shift

Arc-eager on an example

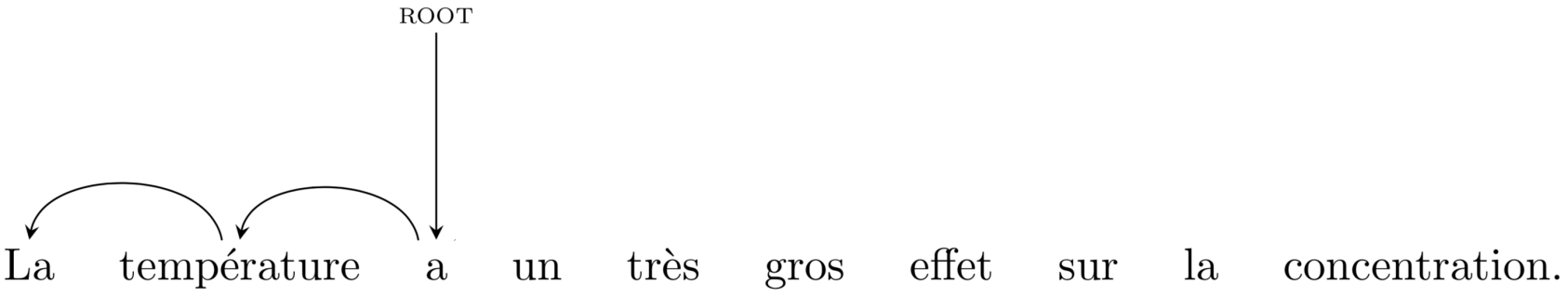
[ROOT, température]$_S$ [a, un, très, gros, effet, sur, la, concentration]$_Q$

action: Left-Arc()

Arc-eager on an example

[ROOT, a]$_S$ [un, très, gros, effet, sur, la, concentration]$_Q$

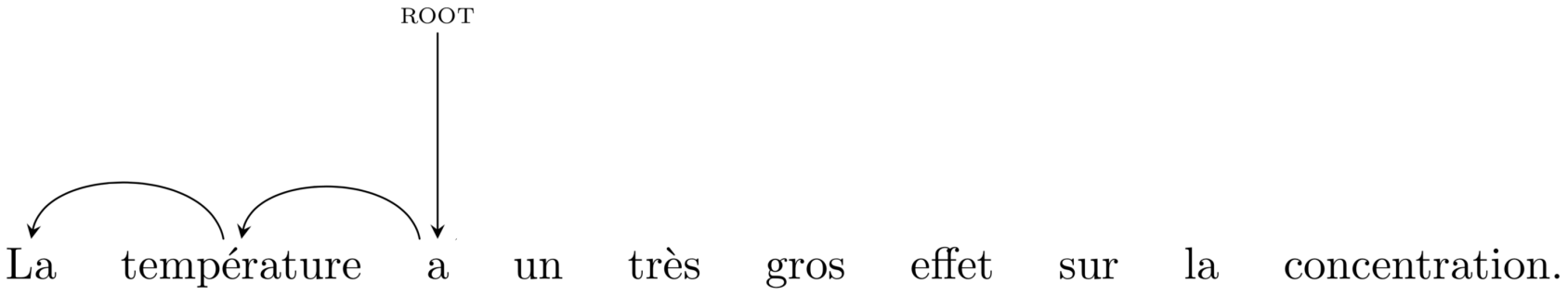
action: Right-Arc()



Arc-eager on an example

[ROOT, a, un]$_S$ [très, gros, effet, sur, la, concentration]$_Q$

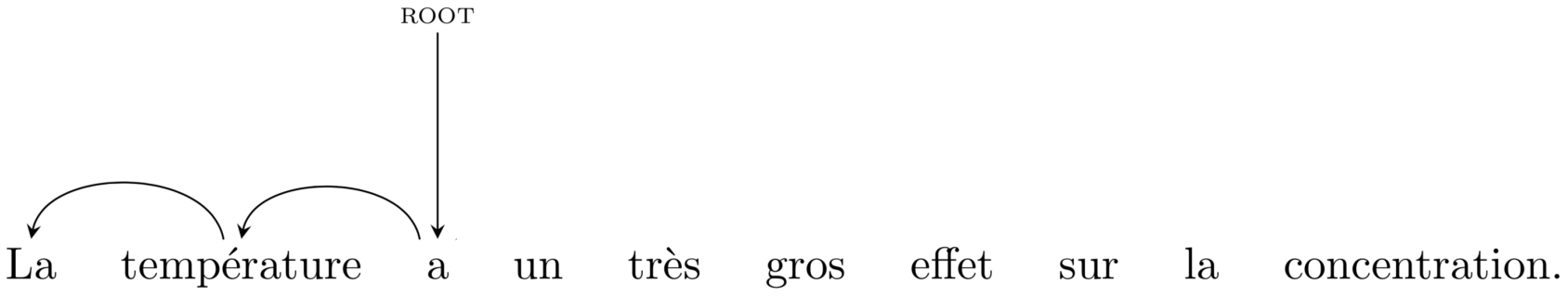
action: Shift



Arc-eager on an example

[ROOT, a, un, très]$_S$ [gros, effet, sur, la, concentration]$_Q$

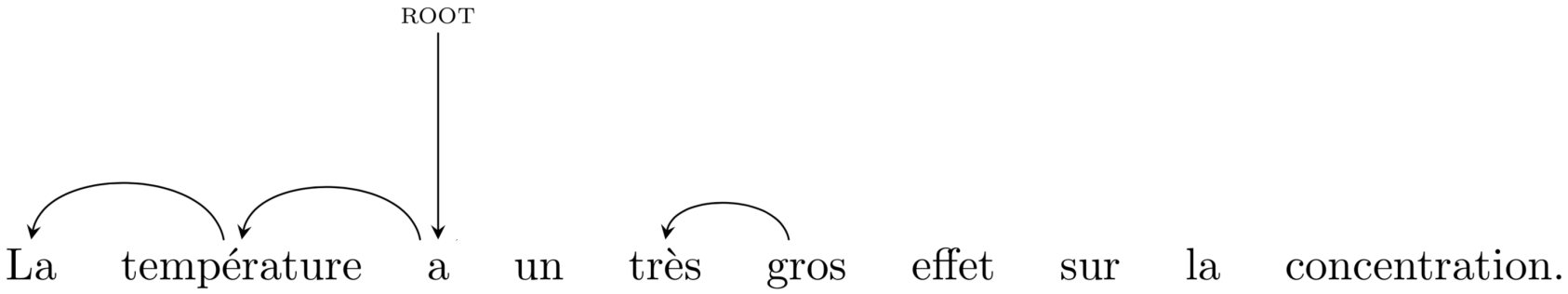
action: Shift



Arc-eager on an example

[ROOT, a, un, très]$_S$ [gros, effet, sur, la, concentration]$_Q$

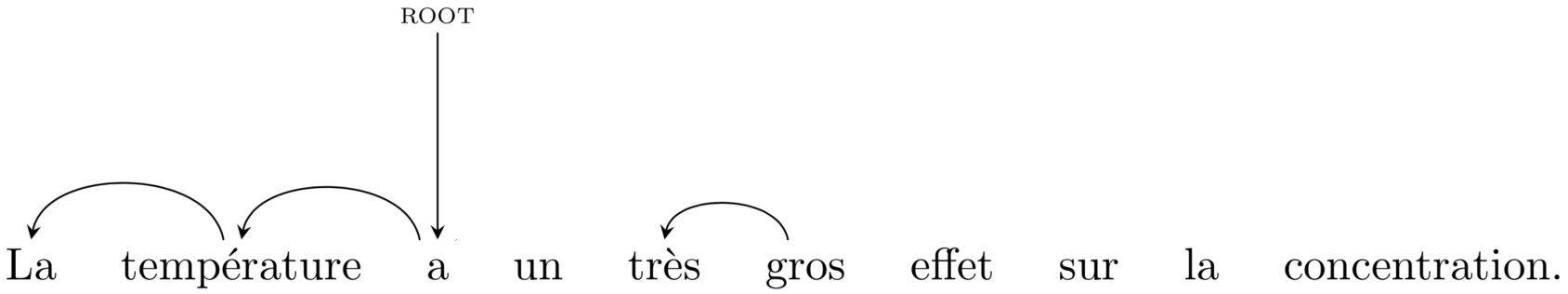
action: Left-Arc()



Arc-eager on an example

$[\text{ROOT, a, un, gros}]_S\ [\text{effet, sur, la, concentration}]_Q$

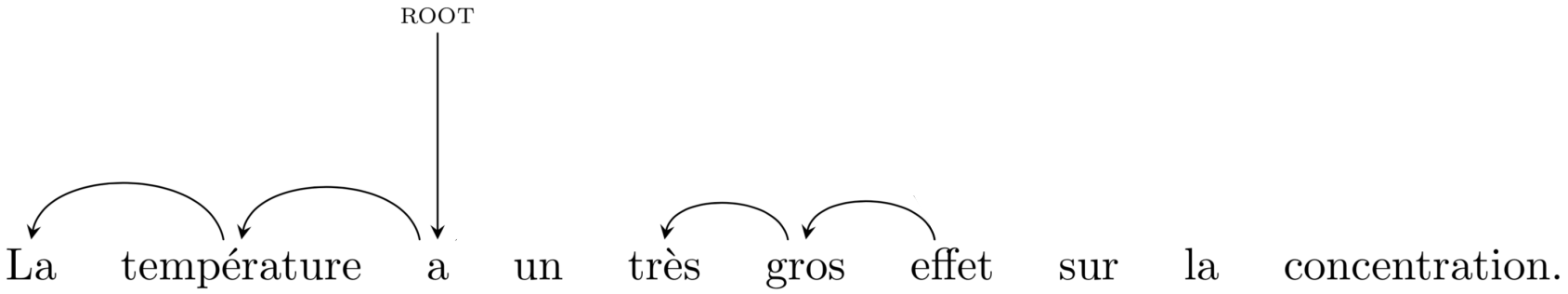
action: Shift



Arc-eager on an example

[ROOT, a, un, gros]$_S$ [effet, sur, la, concentration]$_Q$

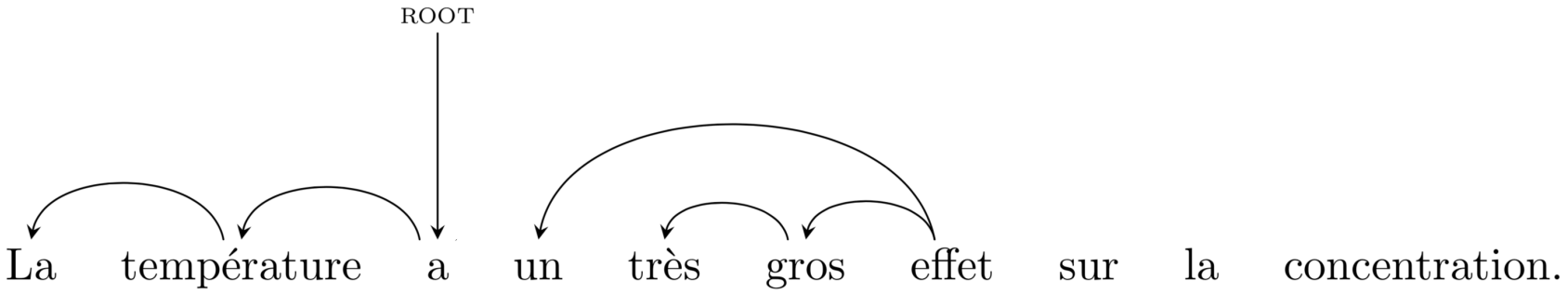
action: Left-Arc()



Arc-eager on an example

[ROOT, a, un]$_S$ [effet, sur, la, concentration]$_Q$

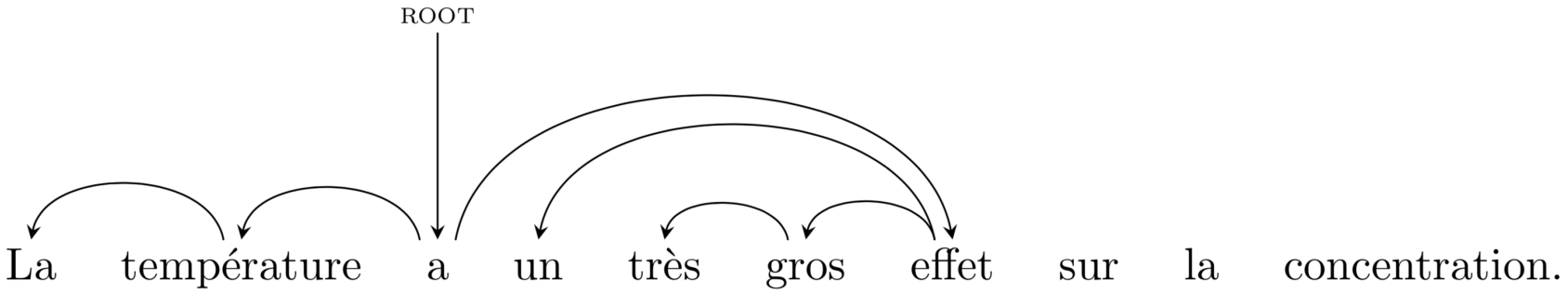
action: Left-Arc()



Arc-eager on an example

$[\text{ROOT, a, effet}]_S\ [\text{sur, la, concentration}]_Q$

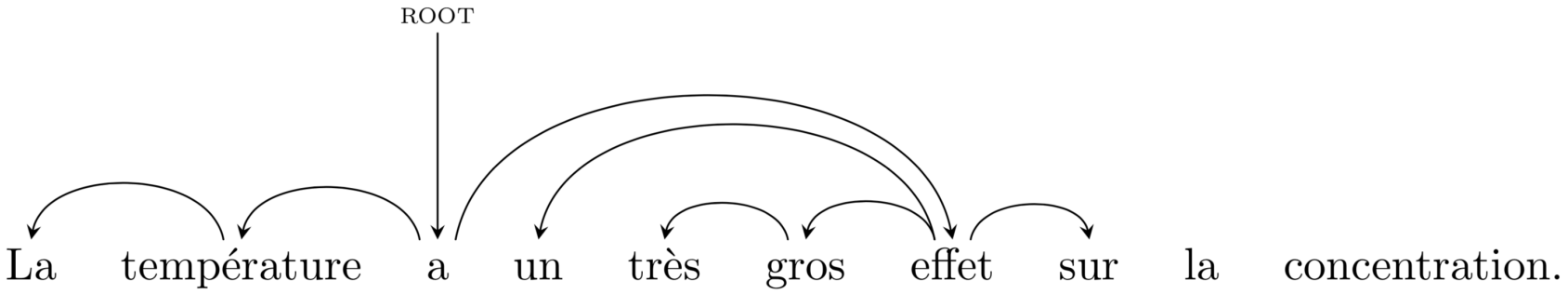
action: Right-Arc()



Arc-eager on an example

[ROOT, a, effet, sur]$_S$ [la, concentration]$_Q$

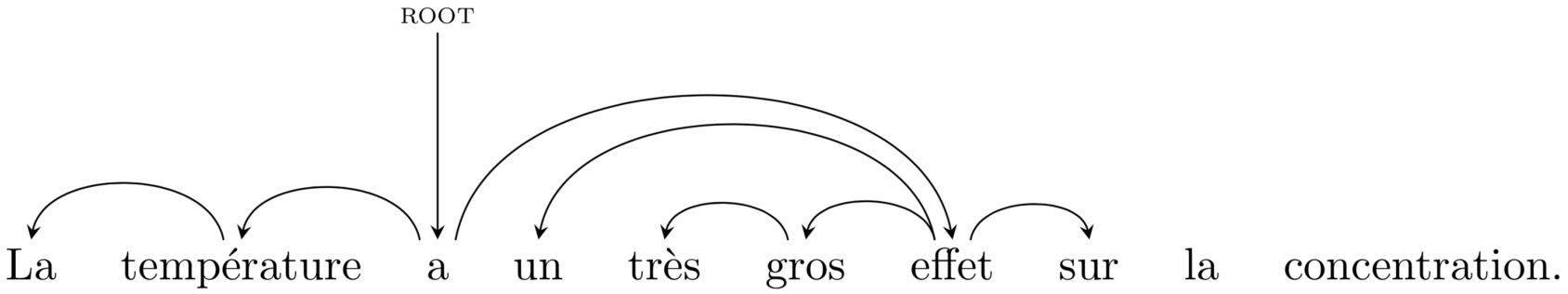
action: Right-Arc()



Arc-eager on an example

$[\text{ROOT, a, effet, sur, la}]_S\ [\text{concentration}]_Q$

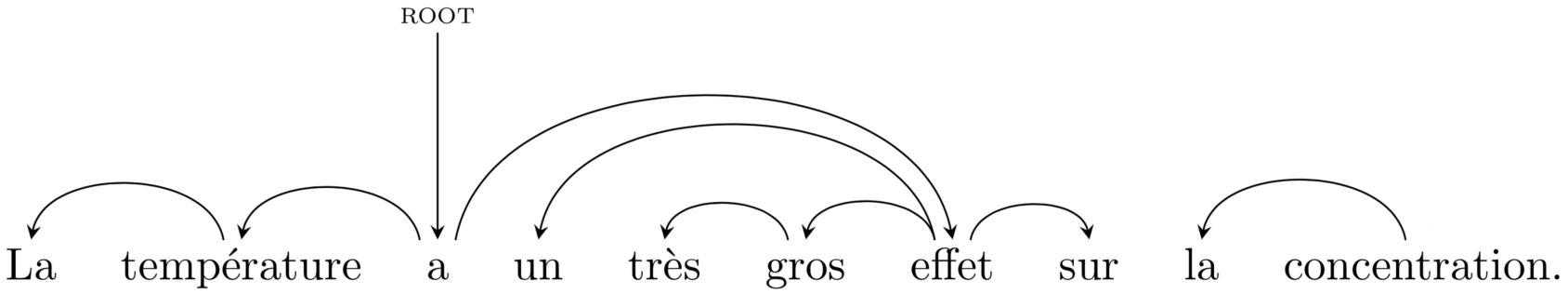
action: Shift()



Arc-eager on an example

[ROOT, a, effet, sur, la]$_S$ [concentration]$_Q$

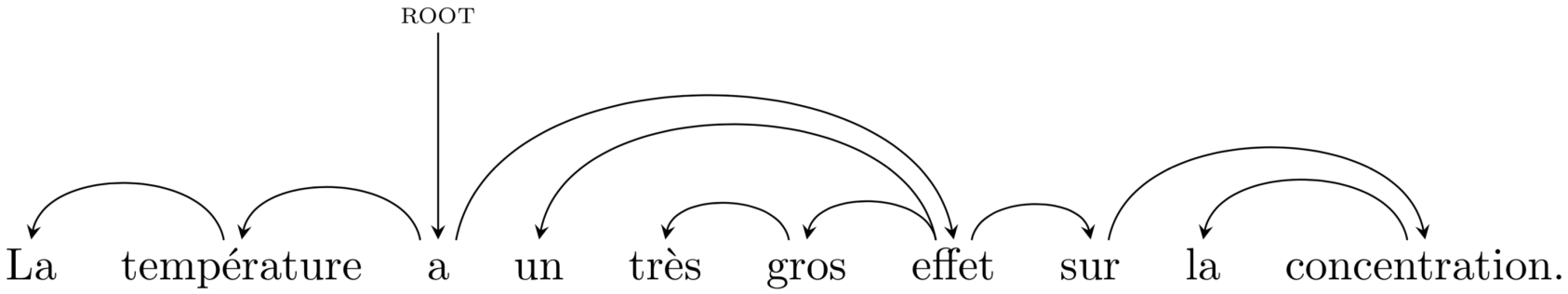
action: Left-Arc()



Arc-eager on an example

[ROOT, a, effet, sur, concentration]$_S$ []$_Q$

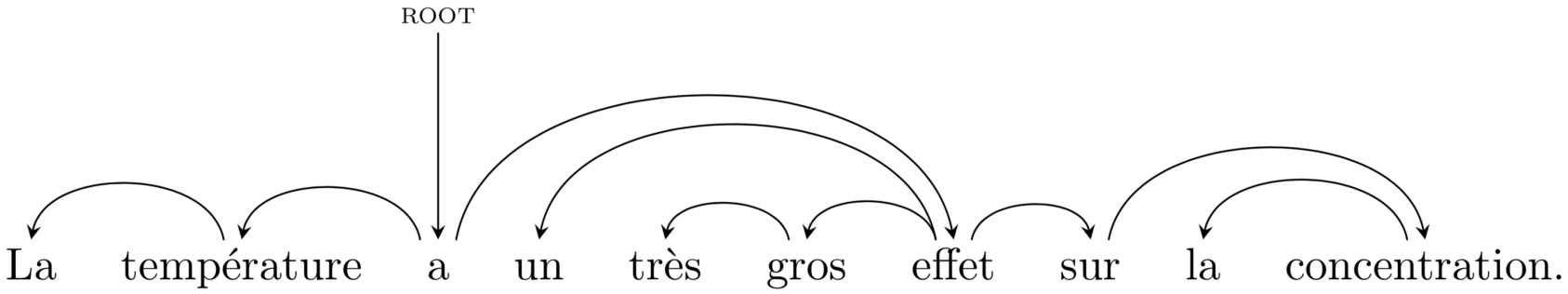
action: Right-Arc()



Arc-eager on an example

[ROOT, a, effet, sur, concentration]$_S$ []$_Q$

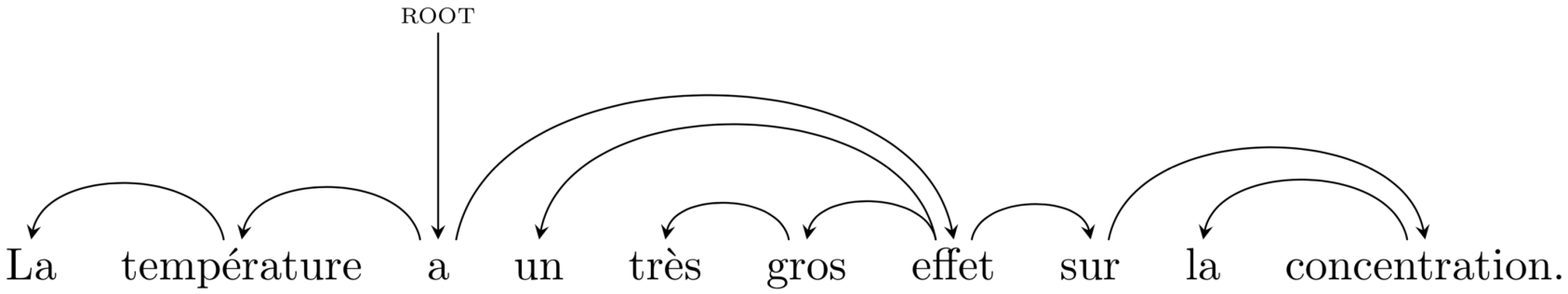
action: Reduce



Arc-eager on an example

[ROOT, a, effet, sur]$_S$ []$_Q$

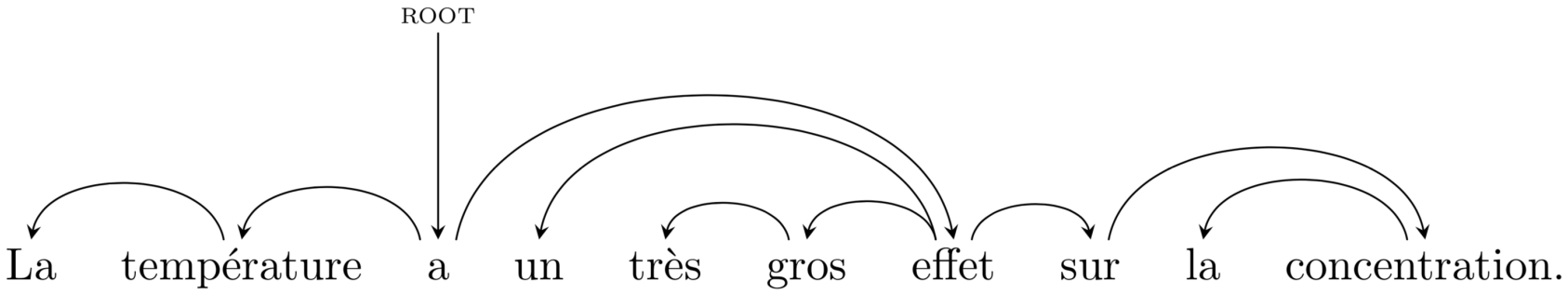
action: Reduce



Arc-eager on an example

[ROOT, a, effet]$_S$ []$_Q$

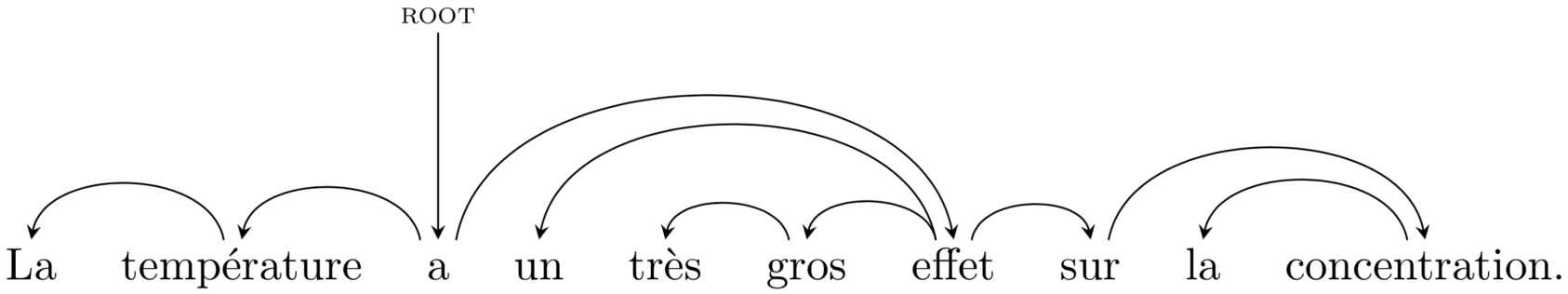
action: Reduce



Arc-eager on an example

[ROOT, a]$_S$ []$_Q$

action: Reduce



Drawbacks of the arc-eager algorithm

- The arc-eager system has a weaker soundness result than the arc-standard system
- It does not guarantee the output to be a dependency tree, only a sequence of (unconnected) trees (a forest).
- In the best case, this is a sequence of length 1, meaning that the tree is in fact a tree.
- In the worst case, this is a sequence of length n, meaning that each word is its own tree.
- The arc-eager parsers normally have a last step that attaches everything that remains in the stack to the root

Transition-Based Parsing with a neural classifier

Example of a neural arc-standard algorithm

- Chen and Manning (2014)
- Replace the action selection module by a neural network
  - Based on manually selected features provided as an input

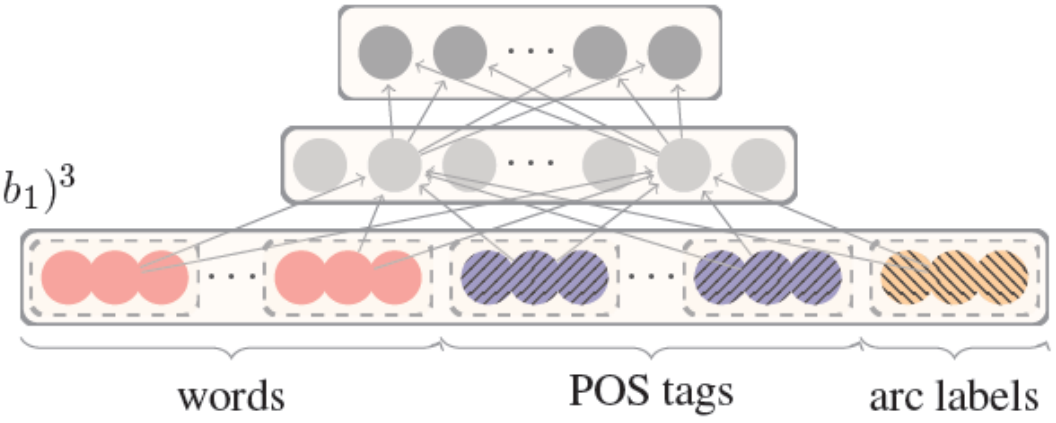
Softmax layer:

$$p = \texttt{softmax}(W_2 h)$$

Hidden layer:

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$

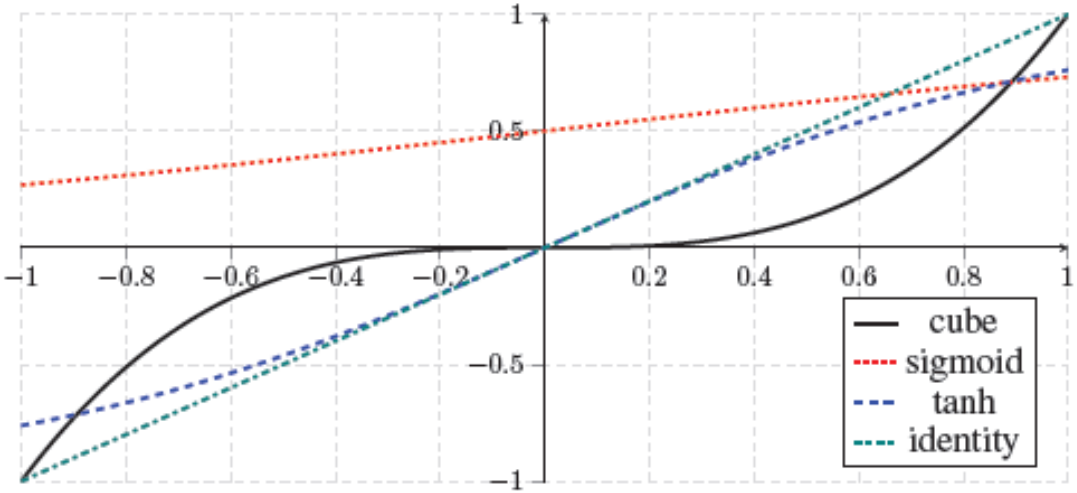
Input layer: $[x^w, x^t, x^l]$



Example of a neural arc-standard algorithm

- Chen and Manning (2014)
- Replace the action selection module by a neural network
  - Based on manually selected features provided as an input
- Cube activation function
  - It directly extracts feature combinations of up to three

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3$$



Example of a neural arc-standard algorithm

- Chen and Manning (2014)
- Replace the action selection module by a neural network
  - Based on manually selected features provided as an input
- Cube activation function
- POS tags and arc labels are discrete sets
  - Normally represented as one-hot vectors
  - Just like words, there should be similarities
    - NN (singular noun) should be similar to NNP (plural noun)
  - Dense embedding layer for POS tags and arc labels capture relationships
- **Better results in accuracy and parsing speed** compared with previous parsers with statistical classifiers

Example of a neural arc-standard algorithm: Experimental Results

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	89.9	88.7	89.7	88.3	51
eager	90.3	89.2	89.9	88.6	63
Malt:sp	90.0	88.8	89.9	88.5	560
Malt:eager	90.1	88.9	90.1	88.7	535
MSTParser	92.1	90.8	**92.0**	90.5	12
Our parser	**92.2**	**91.0**	**92.0**	**90.7**	**1013**

Table 4: Accuracy and parsing speed on PTB + CoNLL dependencies.

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	90.2	87.8	89.4	87.3	26
eager	89.8	87.4	89.6	87.4	34
Malt:sp	89.8	87.2	89.3	86.9	469
Malt:eager	89.6	86.9	89.4	86.8	448
MSTParser	91.4	88.1	90.7	87.6	10
Our parser	**92.0**	**89.7**	**91.8**	**89.6**	**654**

Table 5: Accuracy and parsing speed on PTB + Stanford dependencies.

Parser	Dev		Test		Speed
	UAS	LAS	UAS	LAS	(sent/s)
standard	82.4	80.9	82.7	81.2	72
eager	81.1	79.7	80.3	78.7	80
Malt:sp	82.4	80.5	82.4	80.6	420
Malt:eager	81.2	79.3	80.2	78.4	393
MSTParser	**84.0**	82.1	83.0	81.2	6
Our parser	**84.0**	**82.4**	**83.9**	**82.4**	**936**

Table 6: Accuracy and parsing speed on CTB.

Example of a neural arc-standard algorithm: Model comparison

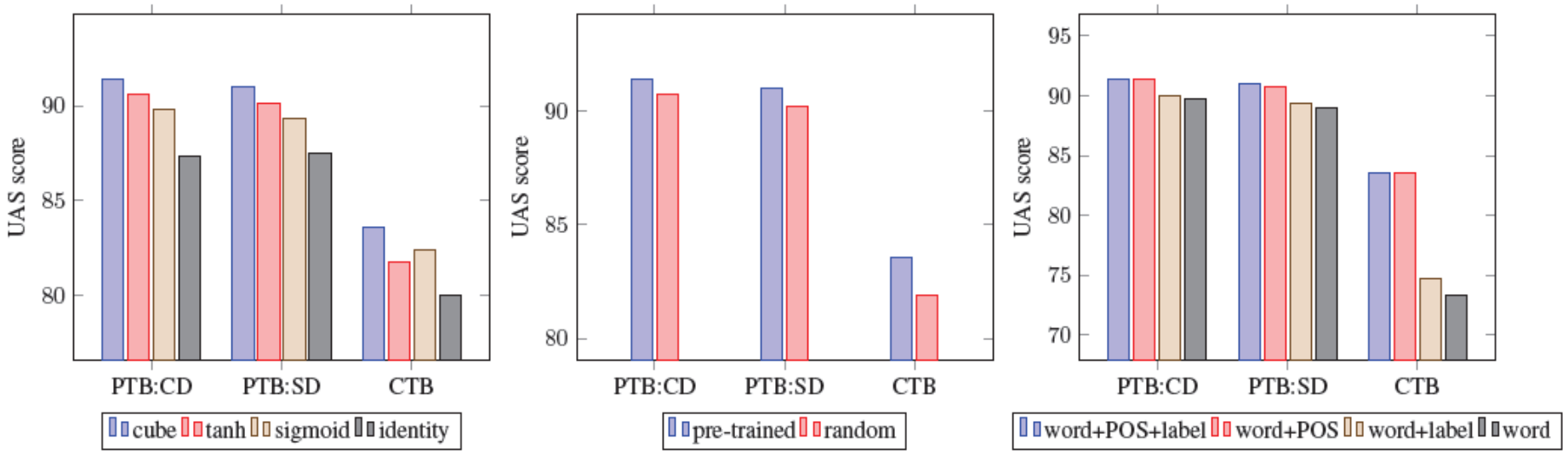
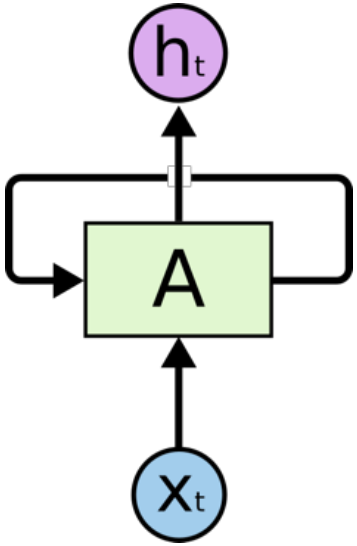


Figure 4: Effects of different parser components. Left: comparison of different activation functions. Middle: comparison of pre-trained word vectors and random initialization. Right: effects of POS and label embeddings.

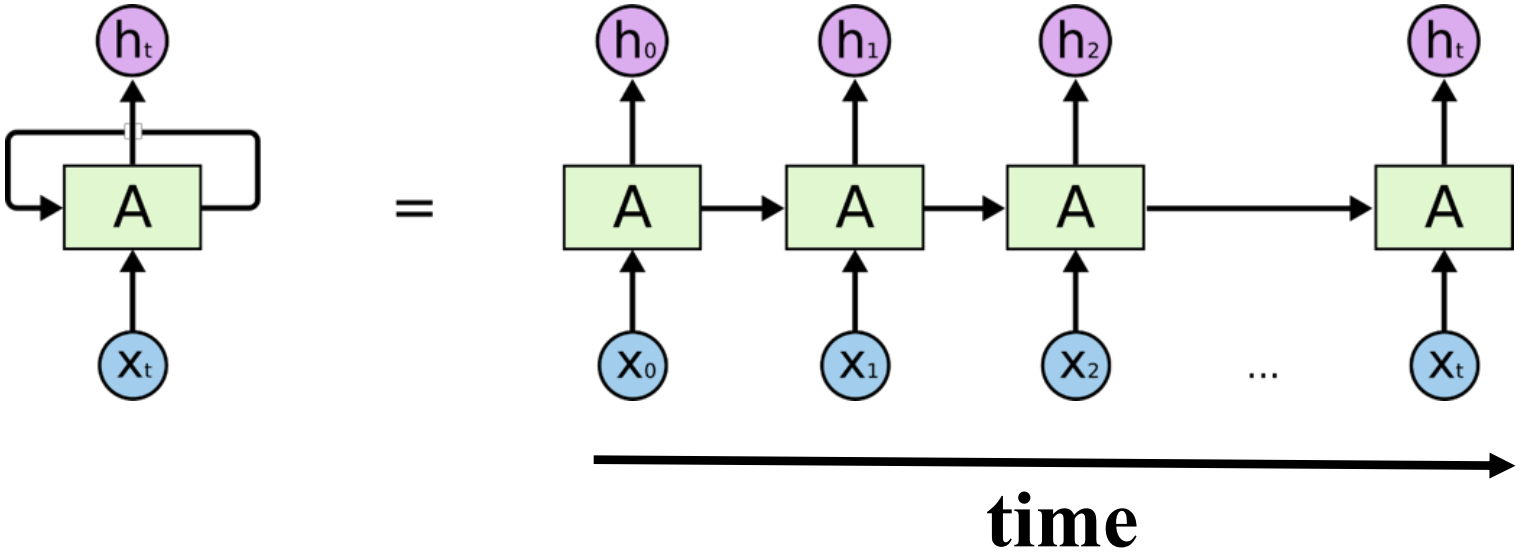
Recurrent Neural Networks and Long Short-Term Memory networks (LSTMs)

Recurrent Neural Networks

- Textual data is structurally sequential
- Using a neural network to build local classifiers is a limited way to use neural networks to make decisions over a sequence of elementary inputs
- We could define and train neural networks over full sequences
  - The idea behind RNNs is to add feedback loops where some units' current outputs determine some future network inputs
  - RNNs can model dynamic finite-state machines, beyond the static combinatorial circuits modelled by feed-forward networks



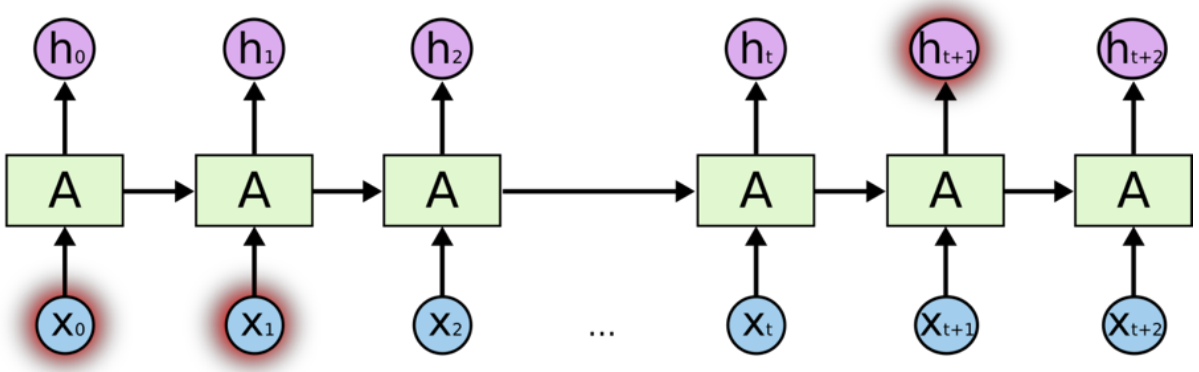
Recurrent Neural Networks



- RNNs can be trained using “backpropagation through time.”
- Can viewed as applying normal backpropagation to the unrolled network

Limitations of Recurrent Neural Networks

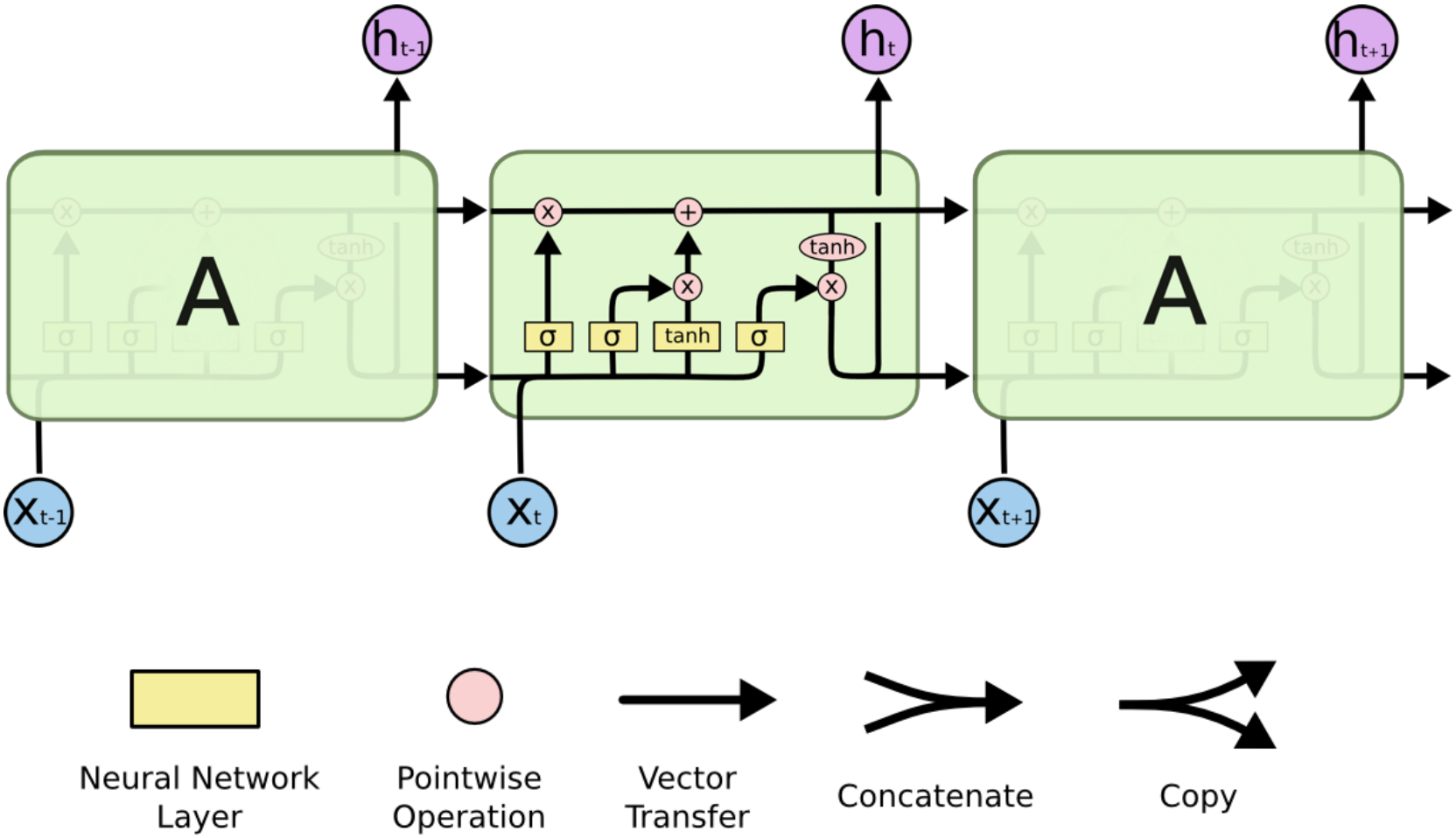
- Backpropagated errors multiply at each layer, resulting in exponential decay (if derivative is small) or growth (if derivative is large)
- Makes it very difficult to train deep networks, or simple recurrent networks over many time steps
- It is very difficult to train basic RNNs to retain information over many time steps
- This make is very difficult to learn basic RNNs that handle long-distance dependencies, such as subject-verb agreement



Long Short-Term Memory

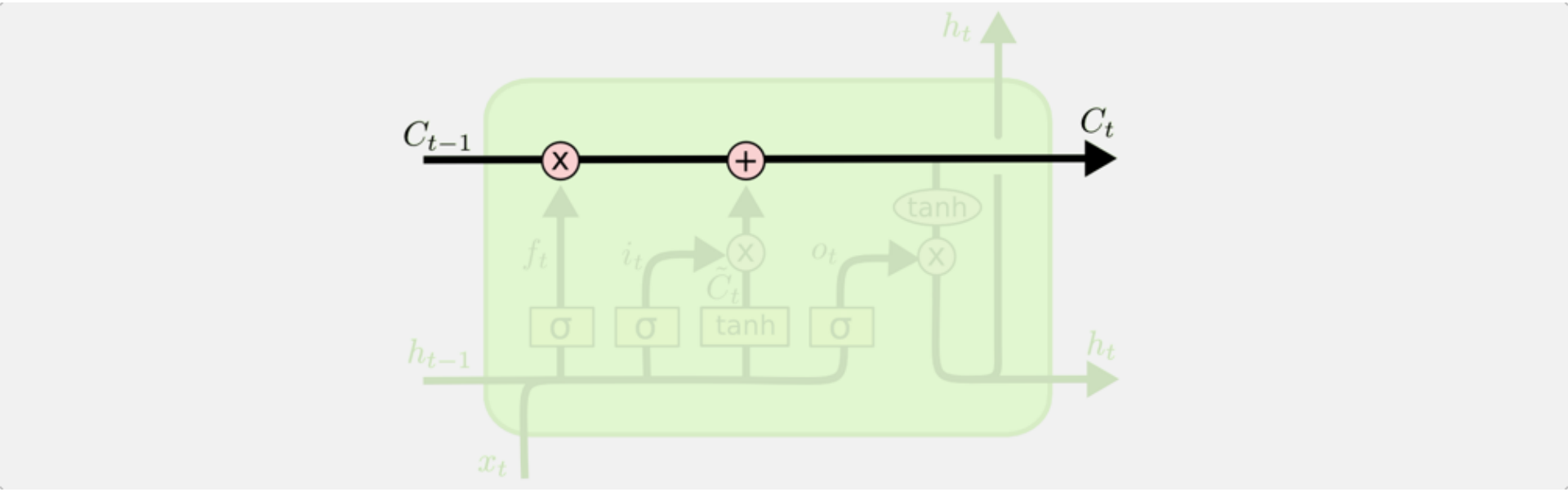
- LSTM networks, add additional gating units in each memory cell.
  - Forget gate
  - Input gate
  - Output gate
- Prevents vanishing/exploding gradient problem and allows network to retain state information over longer periods of time.

Long Short-Term Memory



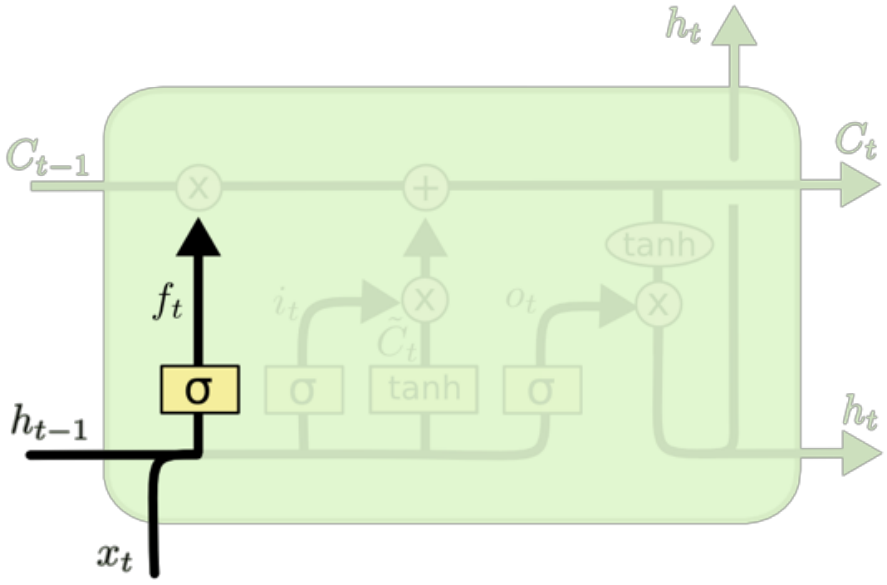
LSTM cell state

- Maintains a vector C_t that is the same dimensionality as the hidden state, h_t
- Information can be added or deleted from this state vector via the forget and input gates.



LSTM forget gate

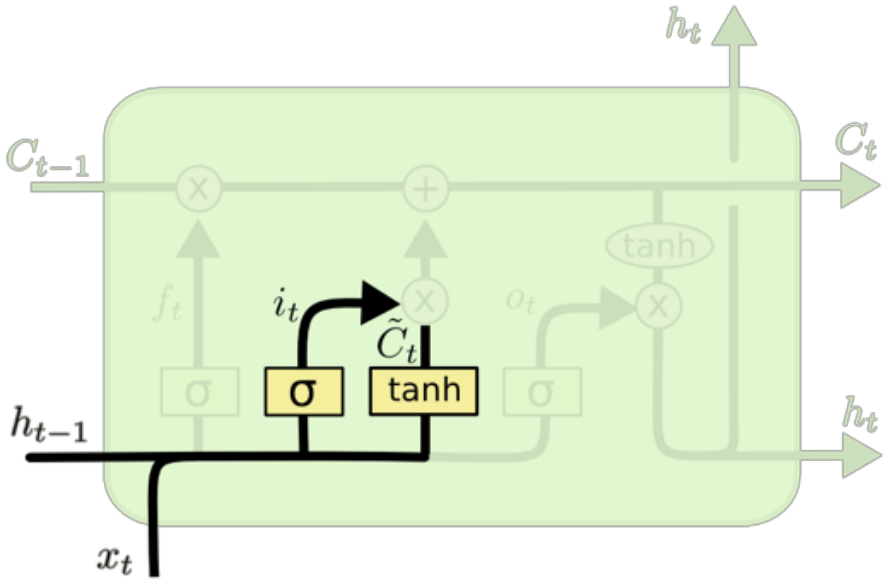
- Forget gate computes a 0-1 value using a logistic sigmoid output function from the input, x_t, and the previous hidden state, h_{t-1}:
- Multiplicatively combined with cell state, "forgetting" information where the gate outputs something close to 0.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM input gate

- First, determine which entries in the cell state to update by computing 0-1 sigmoid output.
- Then determine what amount to add/subtract from these entries by computing a tanh output (valued –1 to 1) function of the input and hidden state.

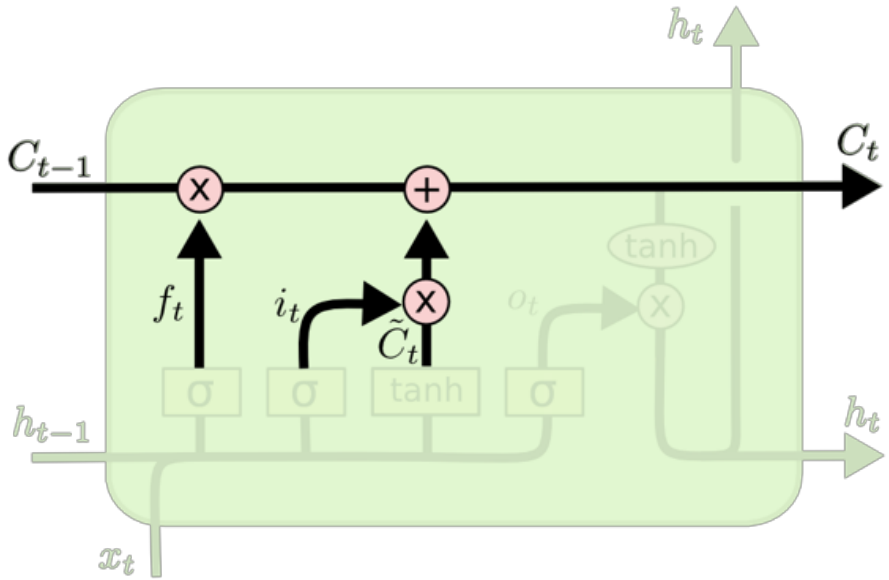


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Updating the cell state

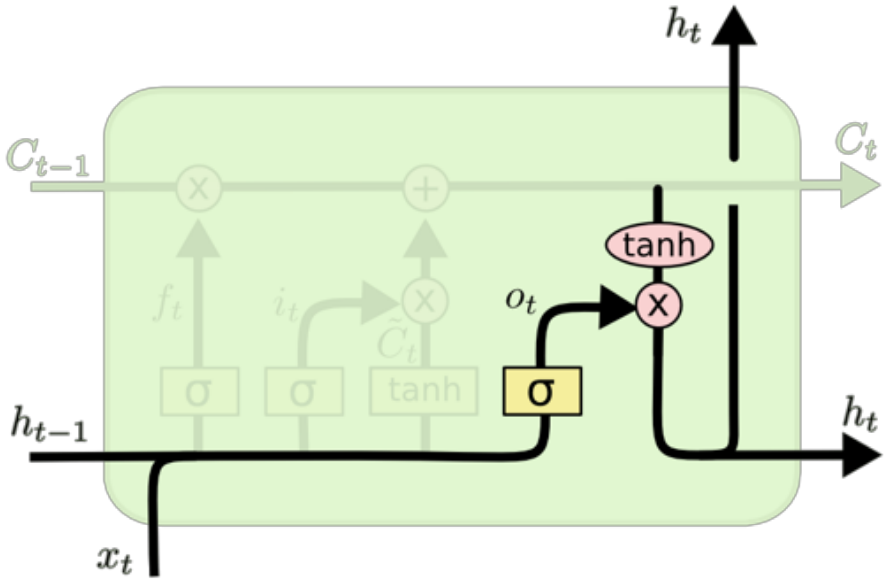
- Cell state is updated by using component-wise vector multiplication to "forget" and vector addition to "input" new information.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM output gate

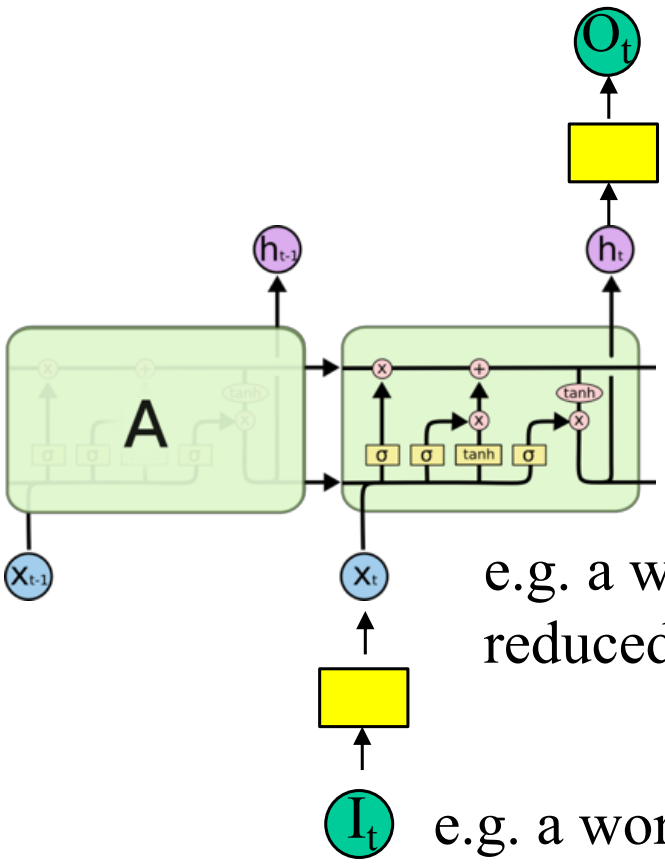
- Hidden state is updated based on a "filtered" version of the cell state, scaled to –1 to 1 using tanh.
- Output gate computes a sigmoid function of the input and previous hidden state to determine which elements of the cell state to "output".



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Overall LSTM architecture



e.g. a POS tag as a “one hot” vector

e.g. a word “embedding” with reduced dimensionality

e.g. a word as a “one hot” vector

Properties

- Drawbacks
  - Many parameters to learn
    - Needs more training data
    - Requires more (GPU) computing power
  - (GRUs are a simplified version of LSTMs with fewer parameters and performance almost as good)
- Advantages
  - Can capture long-distance dependencies between outputs
    - Successful for tasks such as sequence labelling (e.g. POS tagging), language modelling, sequence classification (e.g. text classification)

LSTM architectures

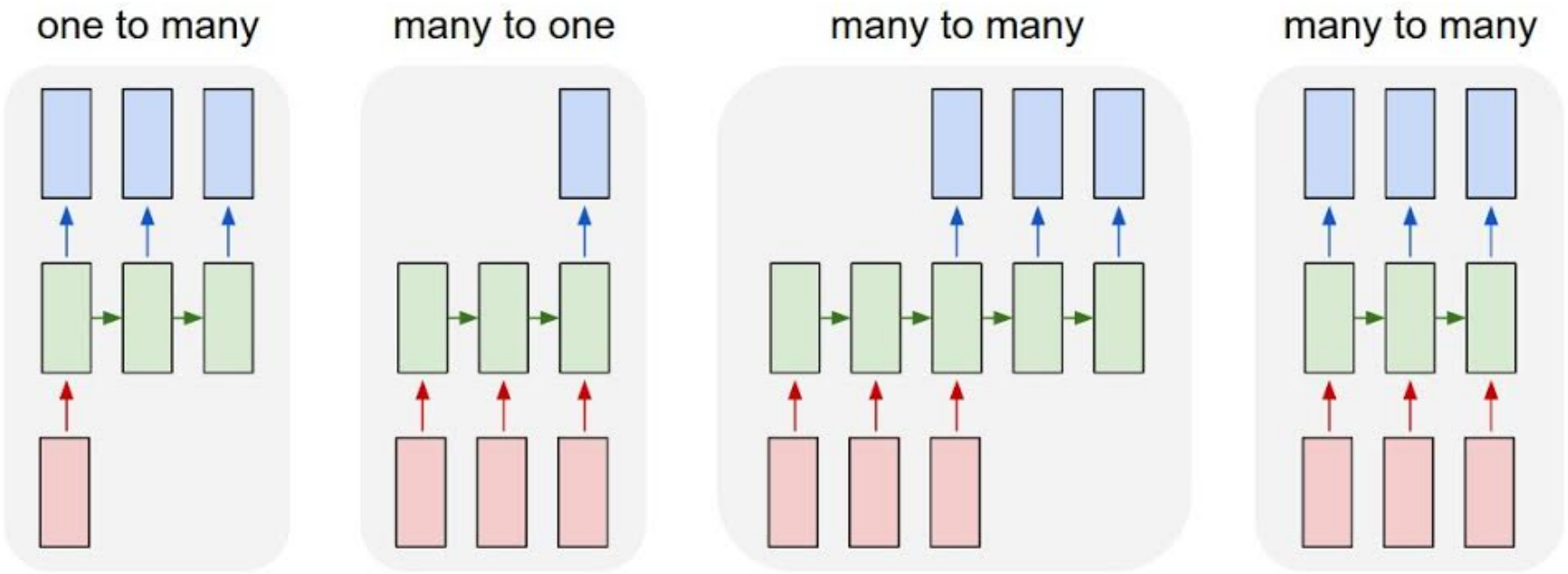


Image Captioning

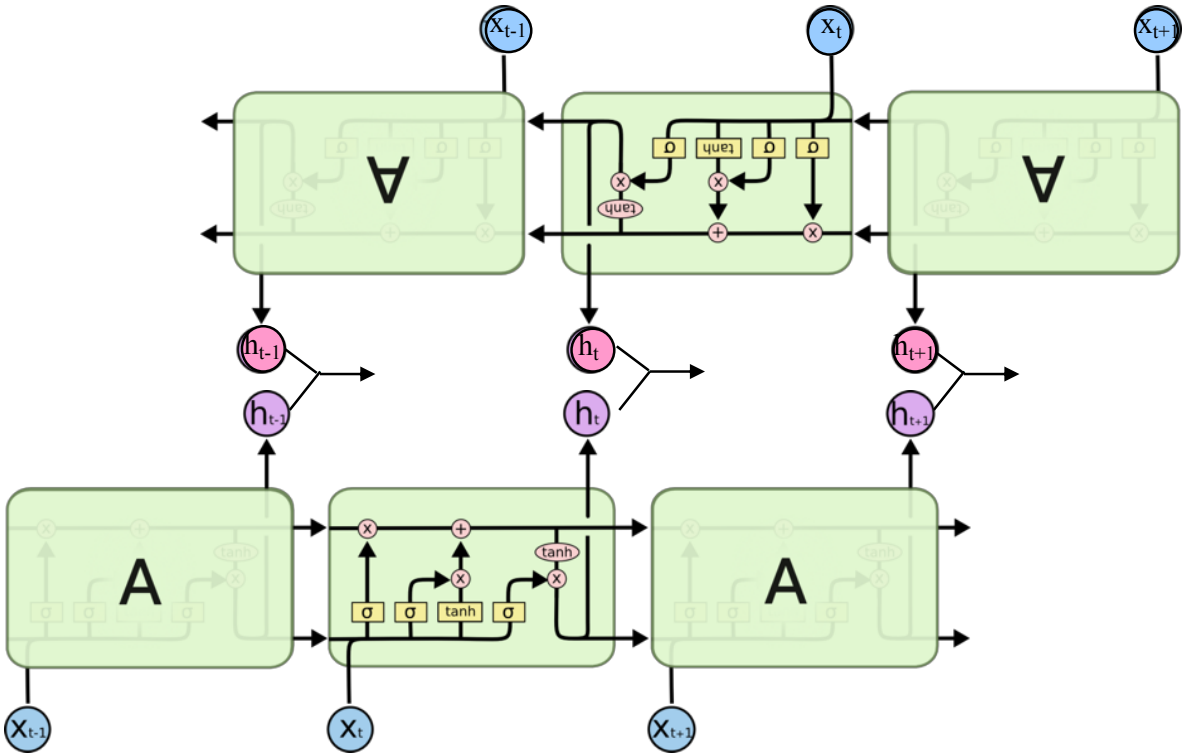
Video Activity Recog
Text Classification

Video Captioning
Machine Translation

POS Tagging
Language Modeling

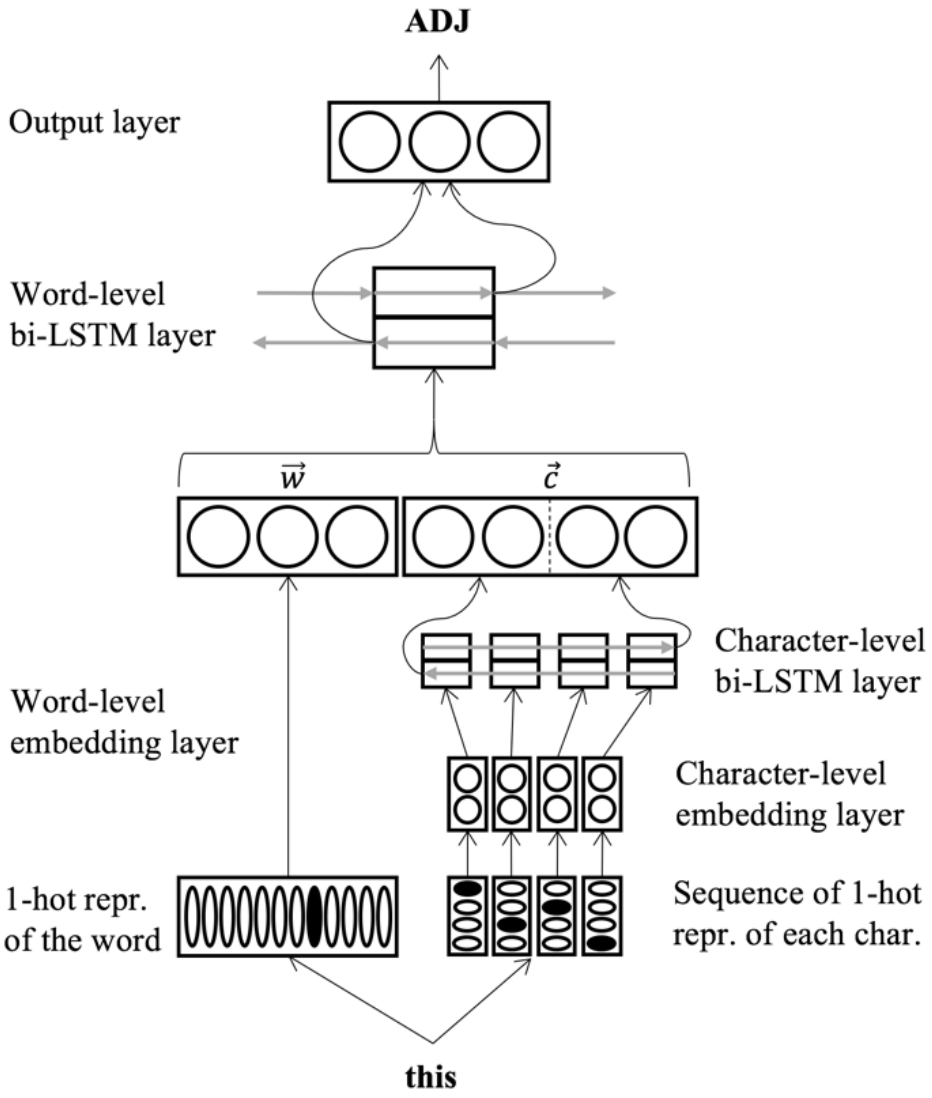
Bi-directional LSTMs

- Two separate LSTMs process the sequence forwards and backwards; hidden layers at each time step are concatenated to form the cell output



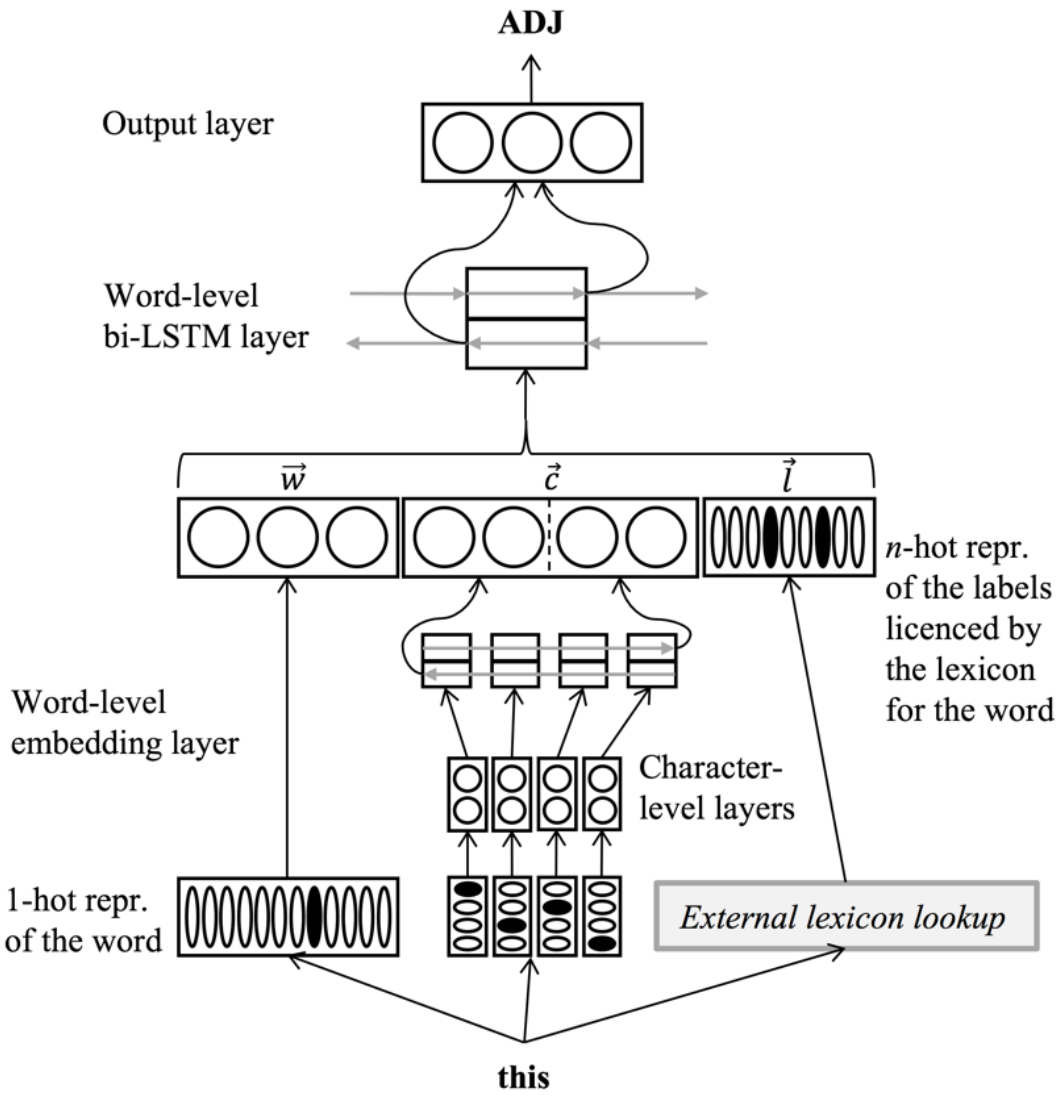
LSTM POS tagging

- (Plank 2016)
- State-of-the-art on the UD corpora (version 1.3) when published



LSTM POS tagging

- (Sagot & Martinez Alonso 2017)
- Extension of the previous architecture to use lexical information from external lexicons (state-of-the-art when published)



LSTM POS tagging

- Overall improvement over the neural state-of-the-art as averaged over dozens of languages
- But not very significant improvement over state-of-the-art *statistical* POS taggers
  - Logistic regression classifier with manually designed features, including features extracted from external lexicons
  - Especially true on smaller datasets
- Neural networks are usually better, but this is not always the case, especially for tasks where feature engineering is simple (we “know” the “right” features)

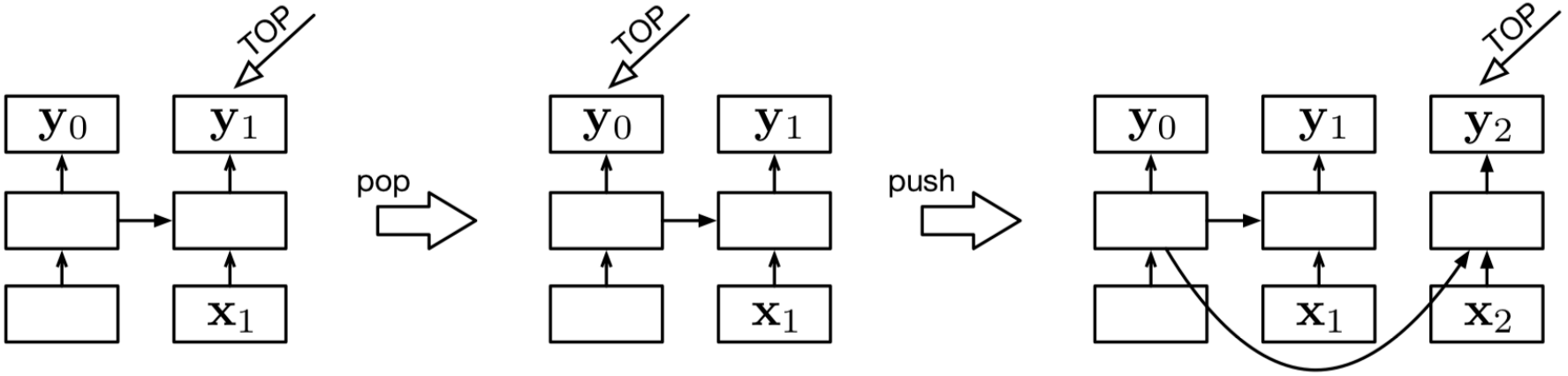
Transition-Based Parsing with Stack LSTMs

Limitations of transition-based parsing with a neural classifier

- Limitations:
  - Elementary feature selection is still done manually
  - Feature combinations are selected by the network
- It would be better to encode all the information about the parse configuration in an RNN or, better, an LSTM
  - **State embeddings**
    - The buffer, the stack, as well as past actions are **sequences** -> we can use RNNs to get representations
- But running an RNN or an LSTM for each state would be too expensive

Parsing with Stack LSTM

- (Dyer et al. 2015)
- We use three LSTMs to encode the stack, the buffer and previous actions respectively
- These LSTMs are not recomputed at each step
- Rather, the notion of LSTM is extended so that the same 3 LSTMs can be updated in the course of parsing



Parsing with Stack LSTM

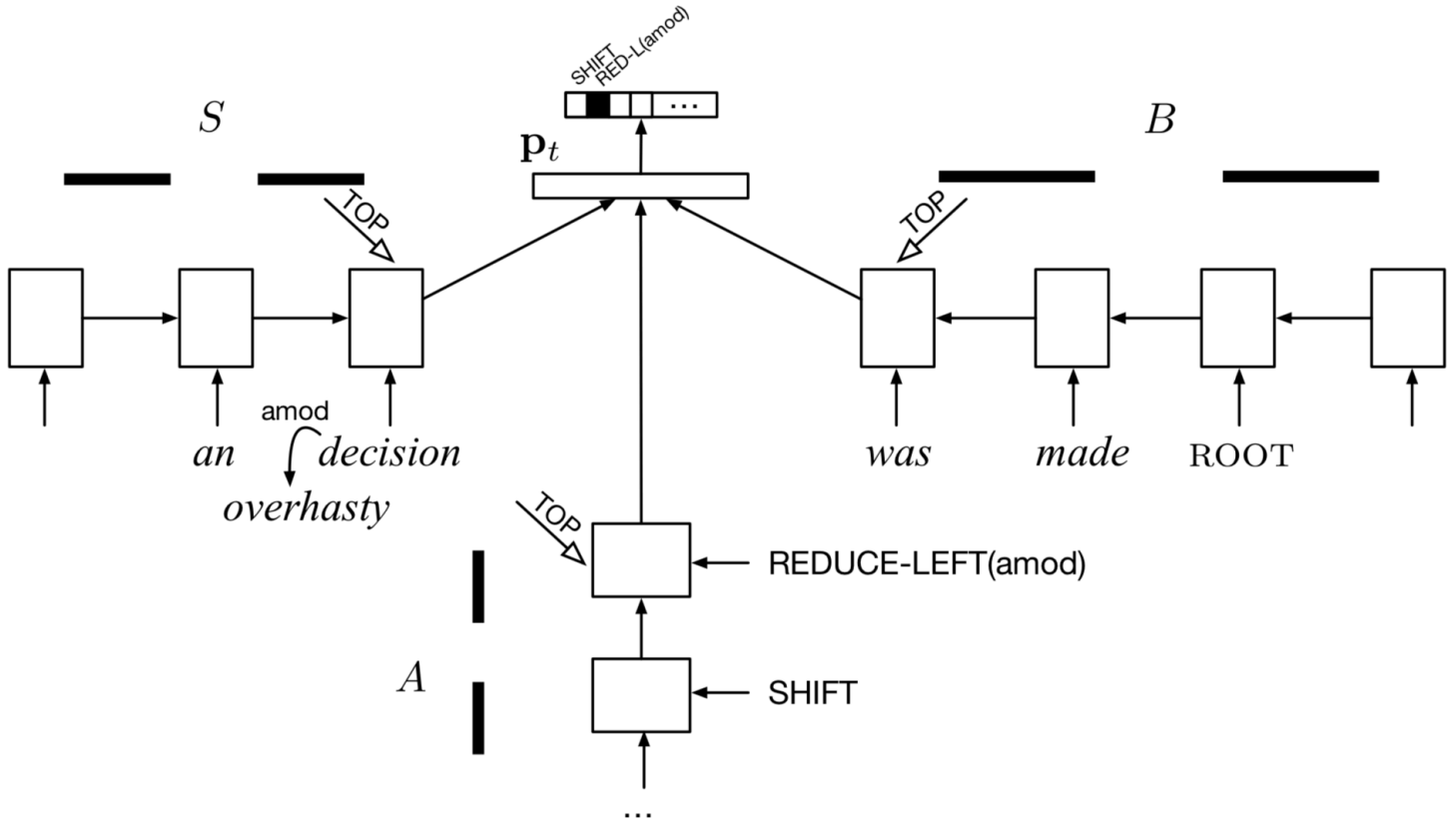


Figure 2: Parser state computation encountered while parsing the sentence "*an overhasty decision was made*." Here S designates the stack of partially constructed dependency subtrees and its LSTM encoding; B is the buffer of words remaining to be processed and its LSTM encoding; and A is the stack representing the history of actions taken by the parser. These are linearly transformed, passed through a ReLU nonlinearity to produce the parser state embedding $\mathbf{p}_t$. An affine transformation of this embedding is passed to a softmax layer to give a distribution over parsing decisions that can be taken.

Parsing with Stack LSTM: results

	Development		Test	
	UAS	LAS	UAS	LAS
S-LSTM	**93.2**	**90.9**	**93.1**	**90.9**
−POS	93.1	90.4	92.7	90.3
−pretraining	92.7	90.4	92.4	90.0
−composition	92.7	89.9	92.2	89.6
S-RNN	92.8	90.4	92.3	90.1
C&M (2014)	92.2	89.7	91.8	89.6

Table 1: English parsing results (SD)

	Dev. set		Test set	
	UAS	LAS	UAS	LAS
S-LSTM	**87.2**	**85.9**	**87.2**	**85.7**
−composition	85.8	84.0	85.3	83.6
−pretraining	86.3	84.7	85.7	84.1
−POS	82.8	79.8	82.2	79.1
S-RNN	86.3	84.7	86.1	84.6
C&M (2014)	84.0	82.4	83.9	82.4

Table 2: Chinese parsing results (CTB5)


Time for a short break!