

Reinforcement learning for grammar induction

Edgar DUC

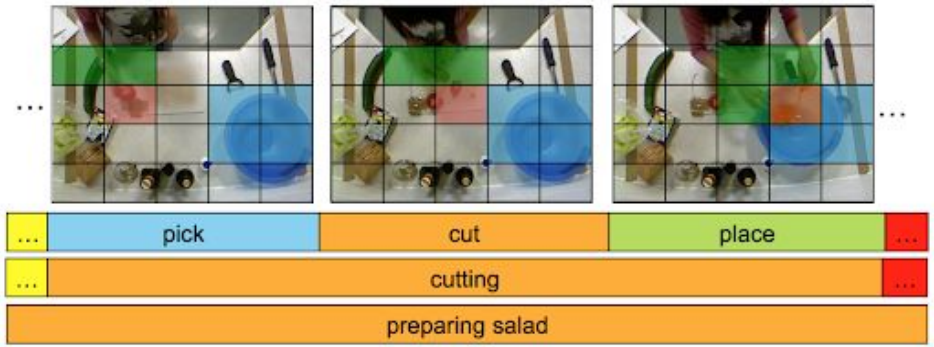
7th July 2025

Table of contents

- Grammar induction
  - Objectives
  - Tree structures, CFG formalism
  - Main tasks : constituency/dependency parsing
  - Datasets and annotation
  - Evaluation methods
- Overview of recent (and less recent) works
  - Early stages : PCFGs, Inside-Outside Algorithm and EM
  - Markovization, neural extension of PCFGs
  - Neural parsing models, LLMs
- Our model
  - Motivation
  - Architecture
  - Results and continuation

Objectives

Sequential data (natural language, DNA, human actions) often exhibit patterns and structure in order to carry information



Extract from 50 salads dataset

Help understanding and predicting sequences by identifying valid patterns

Main applications : NLP (text translation/generation), reverse engineering for code, human action sequence prediction from videos

Tree structures, CFG formalism

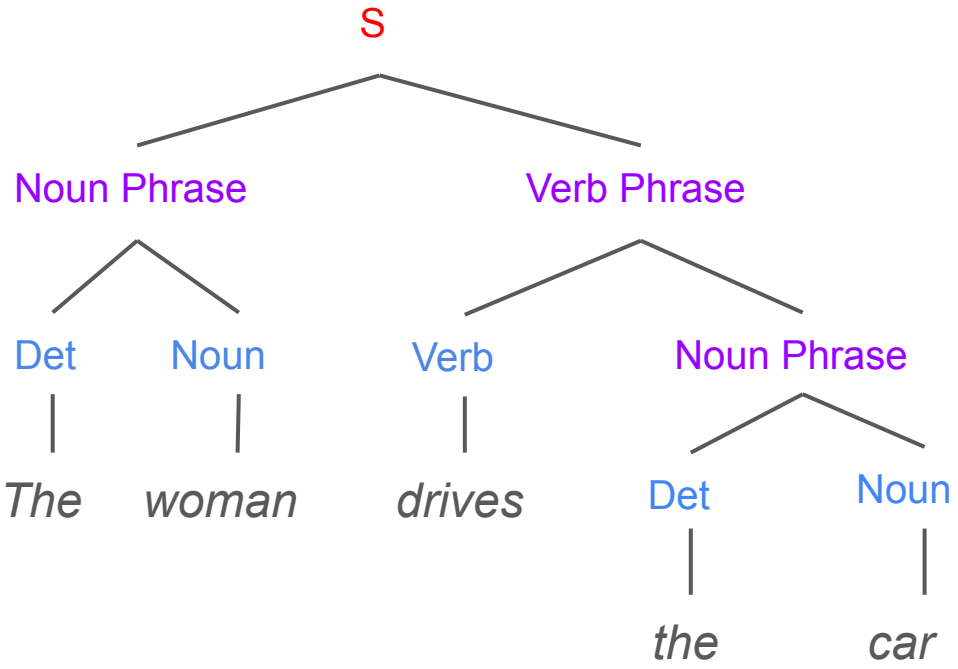
Sequences show independent meaningful constituents



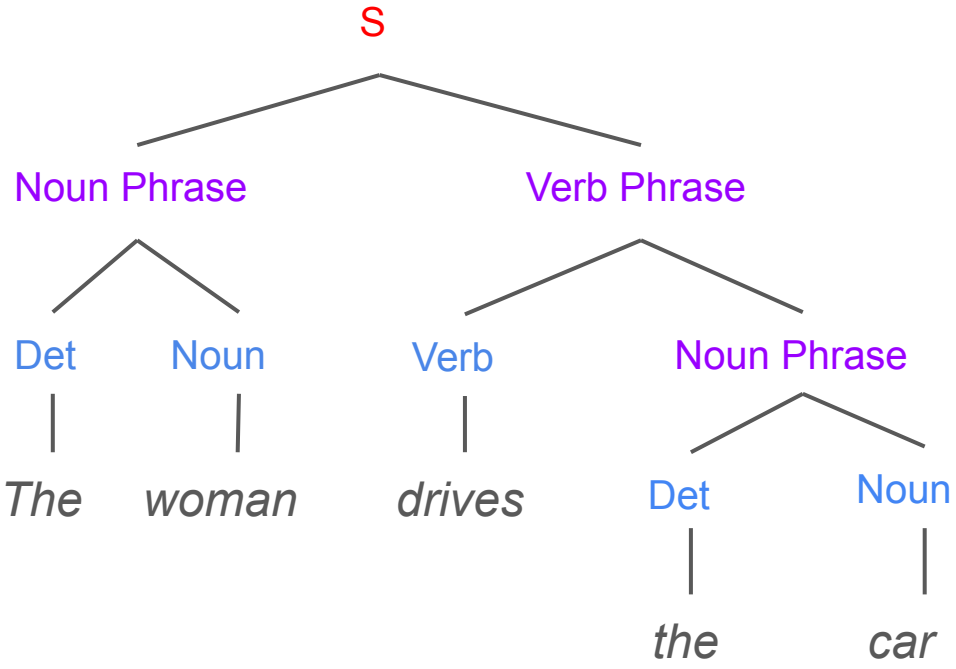
These constituents are nested

Tree structures, CFG formalism

Structure = hierarchically organised constituents
Constituent = independent sub-sequence



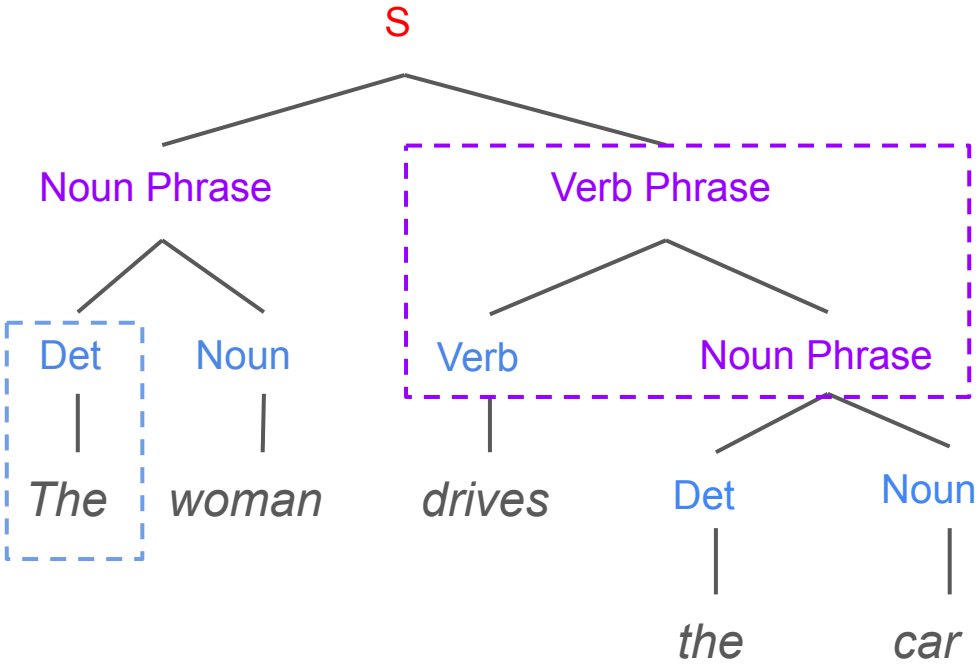
Tree structures, CFG formalism



- Start symbol : whole sentence
- Non-terminals : category of abstract constituent
- Pre-terminals = part-of-speech tag : category of a single word
- Word = terminal

pre-terminals $\subset$ non-terminals
start symbol $\in$ non-terminals

Tree structures, CFG formalism



- Production rule : non-terminal → N pre-terminals/non-terminals
- Lexical rule = unary rule : pre-terminal → word

Tree structures, CFG formalism

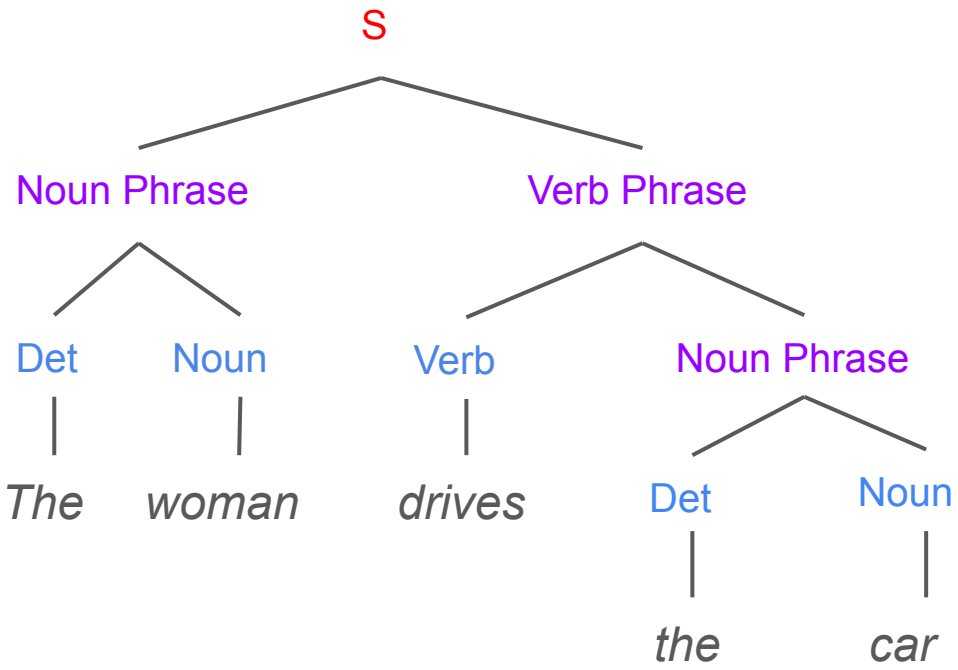
Definition :

A Context-Free Grammar (CFG) is a tuple (N, R, Σ, S) where :

- N is the set of non-terminals, from which we distinguish the pre-terminals $P \subset N$
- $R \subset N \times (N \cup \Sigma)^*$ the set of production rules $(n \rightarrow a_1 \dots a_k)$ that we distinguish in two types : $(n \in N \backslash P \rightarrow a_1 \dots a_k \in N)$ and unary/lexical rules $(p \in P \rightarrow w \in \Sigma)$
- Σ the vocabulary = the set of words
- $S \in N$ the start symbol

Tree structures, CFG formalism

We say that a sentence is “recognized” by a CFG if it exists a tree structure composed of non-terminals and production rule from the CFG

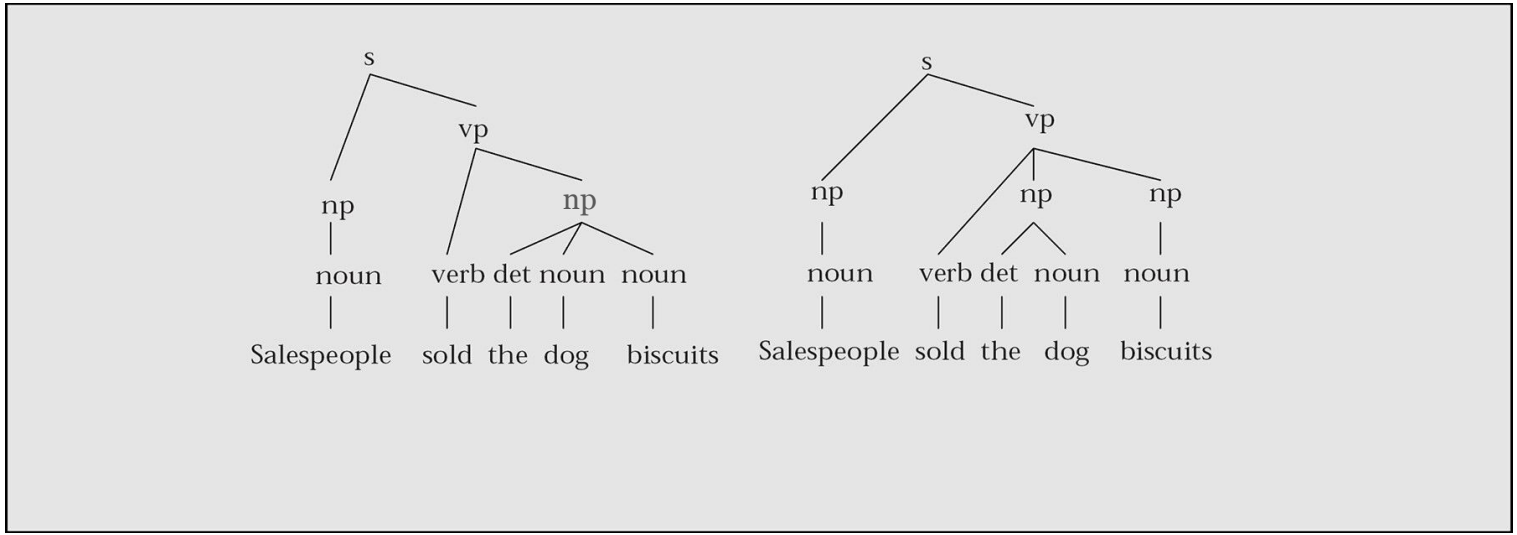


GFG rules :

- S ⟶ Noun Phrase + Verb Phrase
- Noun Phrase ⟶ Det + Noun
- Verb Phrase ⟶ Verb + Noun Phrase
- Det ⟶ “The”
- Noun ⟶ “woman”
- Verb ⟶ “drives”
- Det ⟶ “the”
- Noun ⟶ “car”

Tree structures, CFG formalism

Ambiguity : sometimes the sentence has several tree structures compatible with a same CFG



Tree structures, CFG formalism

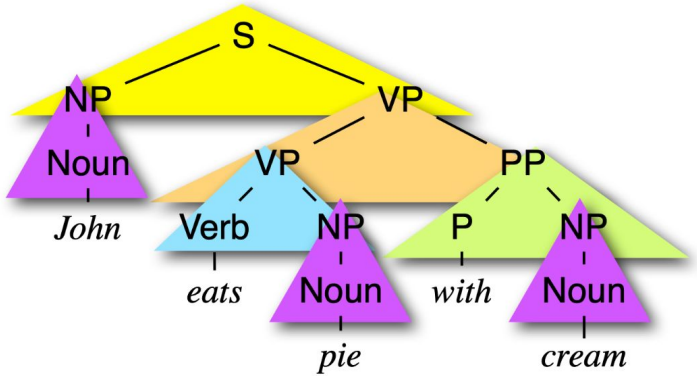
If you take a CFG and you associate a probability π(r) to each production rule r, you obtain a **PCFG**

You must make sure that for a given non-terminal v ∈ N, the sum of the probability of each rule of the form (v → …) is 1.

$$\forall v \in N, \quad \sum_{r \in R, \mathrm{LHS}(r)=v} \pi(r) = 1$$

Tree structures, CFG formalism

The probability of a tree τ is the product of the probabilities of all its rules:



S	→	NP VP	0.8
S	→	S conj S	0.2
NP	→	Noun	0.2
NP	→	Det Noun	0.4
NP	→	NP PP	0.2
NP	→	NP conj NP	0.2
VP	→	Verb	0.4
VP	→	Verb NP	0.3
VP	→	Verb NP NP	0.1
VP	→	VP PP	0.2
PP	→	P NP	1.0

$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3$$

$$= \mathbf{0.00384}$$

Tree structures, CFG formalism

PCFG allows to define the most likely tree structure for each sentence !

PCFG also allows to define the probability of a sentence as sum of the probabilities of all tree structures for that sentence.

Main tasks

- **Grammar inference :** given a corpus of sentences, find a compact representation (often CFG) from which you can recognize valid sentences and reject invalid ones
- **Constituency parsing :** find a model that can give a meaningful tree structure to the sentence.

Main tasks

Grammar inference :

- Given a training corpus (generally the training corpus only contains valid sentences), we must find a CFG that can recognize valid sentences without over-generalizing or over-fitting : unseen valid sentences must be recognized and unseen invalid sentences must be rejected
- You may also use a PCFG and try to make valid sentences likely and invalid sentences unlikely or not recognized

Evaluation :

- **Discrimination (test corpus with valid and invalid sentences)** : F1 score on true positives, false positives, false negatives
- Compression ratio (test corpus with only valid sentences) : compromise between coverage (= proportion of test corpus that is recognized by CFG) and description length (=number of non-terminals + pre-terminals)
- For PCFGs : average likelihood of the corpus

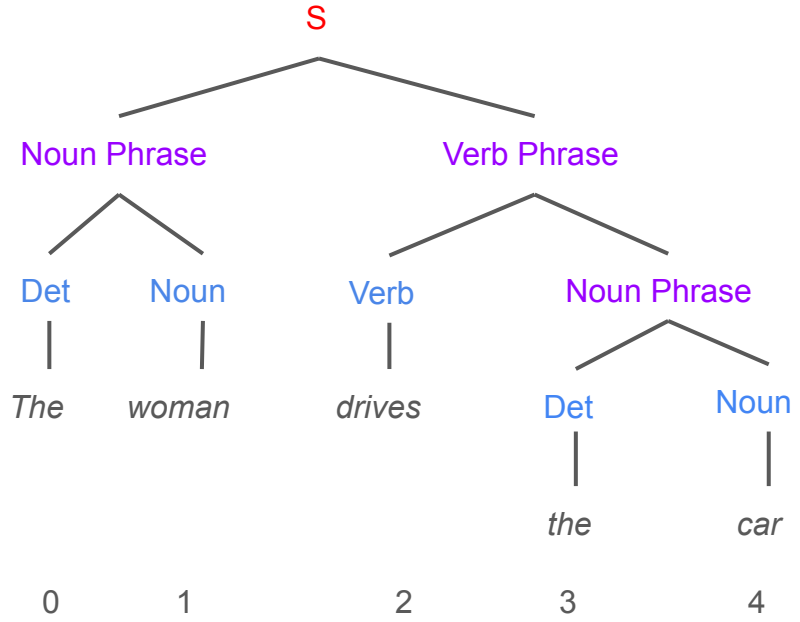
It is know that turtles like salad — Valid ⟶ True positive

You should not play with fire — Invalid ⟶ False Negative

Schools are good places learn — Valid ⟶ False Positive

Main tasks

An other way to represent the tree structure as a set of **spans**, where spans are a tuple (left, right, label) that define a constituent



Spans :

- (0, 4, S)
- (0, 1, Noun Phrase)
- (2, 4, Verb Phrase)
- (0, 0, Det)
- (1, 1, Noun)
- (2, 2, Verb)
- (3, 4, Noun Phrase)
- (3, 3, Det)
- (4, 4, Noun)

Main tasks

F1 score (unlabeled)

- Sentence : “The woman loves the man”
- Ground truth spans : (The woman) (loves (the man))

  C* = {(0,1), (2,4), (3,4)}

- Predicted spans : ((The woman) loves) ((the) man)

  C_{pred} = {(0,1), (0,2), (3,3),(3,4)}

Precision = 2/4, Recall = 2/3, F1 = 2x0.5x0.666 / (0.5+0.666)

For a given sentence $s_1, s_2, ..., s_n$, a set of ground truth spans C^* and a set of predicted spans C^{pred} we compute precision and recall as :

$$precision = \frac{|C^* \cap C^{pred}|}{|C^{pred}|}, \; recall = \frac{|C^* \cap C^{pred}|}{|C^*|}$$

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

Main tasks

Constituency parsing :

We build a model that takes as input a sentence and outputs its tree structure.

A PCFG can work as parsing model.

A lot of modern models are based on neural networks and unexplainable.

Evaluation :

- **Unlabeled F1 score**
- Labeled F1 score

Main tasks

Binarization :

To limit the search space of CFGs and tree structures, almost everyone only considers **binary** production rules

A binary CFG is a CFG where the production rules are either unary ($p \in P \rightarrow$ word) or binary ($n \in N \backslash P \rightarrow n1\ n2 \in N$)

This is possible because any ordinary CFG admits an equivalent binary CFG

The classical algorithms over CFG and PCFGs (to determine the possible/most-likely tree structures of a sentence) are much more efficient if the grammar is binary !

ex : CYK algorithm, Viterbi

Dataset and annotation

The main dataset for grammar induction tasks is **Penn TreeBank**.

It consists in sentences from the Wall Street Journal that were annotated by human experts (each sentence has an associated tree structure).

example : (S (CC *But*) (NP-SBJ (PRP *they*)) (VP (VBP *are*) (ADJP-PRD (JJ *worried*))))

Sentences are between 2 and 60 words long.

Approximately 49000 sentences

Around 20000 distinct words

Dataset and annotation

PTB dataset allows production rules with more than 2 children.

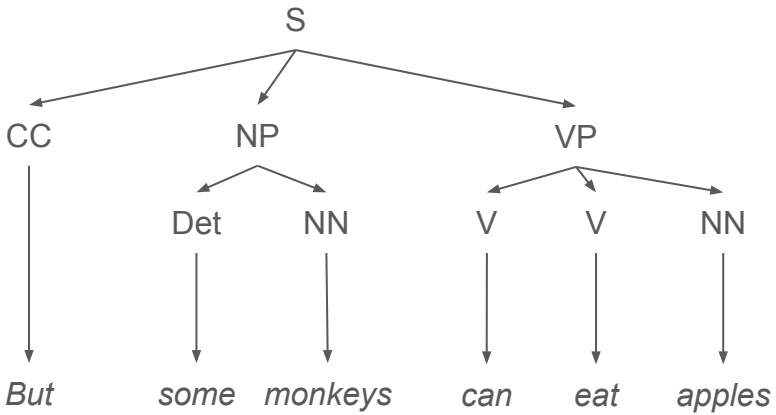
To stay aligned with the limitation to binary production rules, we need to binarize the tree structures !

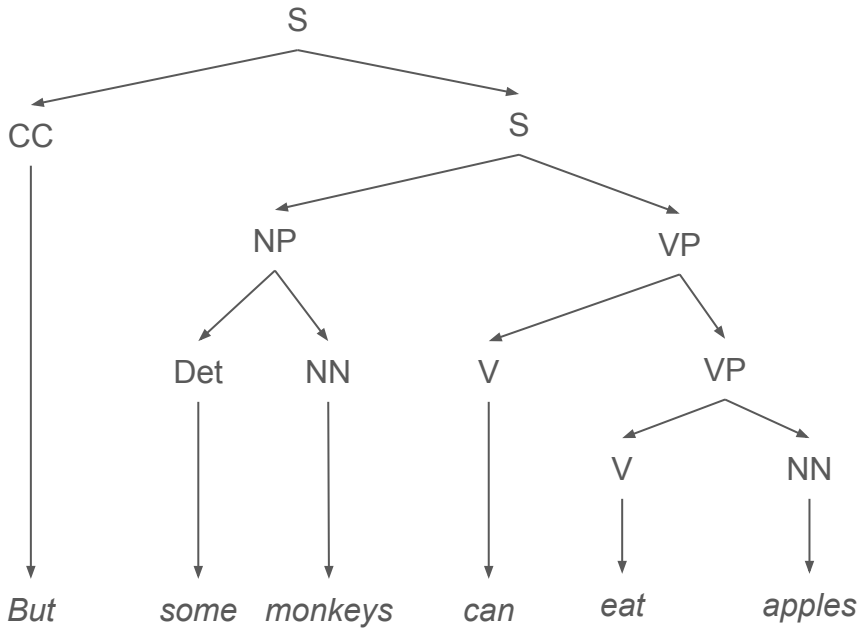
Different techniques to binarize a tree structure :

- right branching with parent label : we decompose (NP → DT JJ NN) into (NP → DT NP) and (NP → JJ NN)
- right branching with artificial labels :
  - basic : (NP → DT JJ NN) becomes (NP → DT X) and (X → JJ NN)
  - LHS-specific : (NP → DT JJ NN) becomes (NP → DT NP') and (NP' → JJ NN)
  - Markov order 1 (horizontal): (NP → DT JJ NN) becomes (NP → DT NP^DT) and (NP^DT → JJ NN)
  - Markov order 1 (vertical)...
- head-driven branching…

Dataset and annotation

Example of right-branching with parent annotation :





Overview of grammar induction works

PCFG induction through EM algorithm

For each sentence from the corpus, we compute the likelihood of the sentence ***x*** under the parameters π as the sum of its parse tree probabilities

This is done with dynamic programming (Inside algorithm) : we recursively compute for each span (i, j, A) the probability that non-terminal A produces the substring from word i to word j. The desired probability is the one of (0, n, S).

$$p(i,j,A) = \sum_{B,C \in N} \left[\pi(A \rightarrow BC) \sum_{i \leq k < j} p(i,k,B) p(k+1,j,C) \right]$$

We then try to maximize through EM the following objective :

$$J(\pi) = \sum_{x_i \in X} log(P(x_i|\pi)) = \sum_{x_i \in X} log(\sum_{t \rightarrow x_i} P(t|\pi))$$

Overview of grammar induction works

Markovization, prior distribution over PCFGs

Here are common extension of the PCFG model

- Refinement of non-terminals : automatically splitting one non-terminal category into several more specific (NounPhrase into NounPhrase-Object and NounPhrase-Subject), preprocessing the data with horizontal or vertical markovization of tree nodes (S → NP VP becomes S → NP^S VP^S)
- Bayesian inference : assume a Dirichlet prior over the rule distribution of a non-terminal $\pi_A = (\pi(A \to BC), \pi(A \to DF), ...) \sim Dirichlet(\alpha)$

Overview of grammar induction works

Neural networks arrive…

- Neural parametrization of PCFG : the rule distribution is controlled by a prior and the output of a sentence embedding from a neural network. Rule distribution is conditioned over the sentence
- RNNs, Transformers, LLMs : LLMs give embeddings to the sentences, and then a NN is used to estimate the span scores s(i, j, A) with similar dynamic programming as PCFG.

The models are able to capture context, they have greater performance in terms of F1 score on PTB dataset, but they are less explainable…

Our model

Motivation

- Although supervised (using tree-structure annotation) constituency parsing is now well handled by neural architectures and LLMs, unsupervised constituency parsing is still a hard task
- A lot of models rely on dynamic programming during training and evaluation which makes them slow !
- Models are not explainable anymore, they sometimes don’t use non-terminal labelling, and they don’t give us a global representation that captures the patterns of the corpus

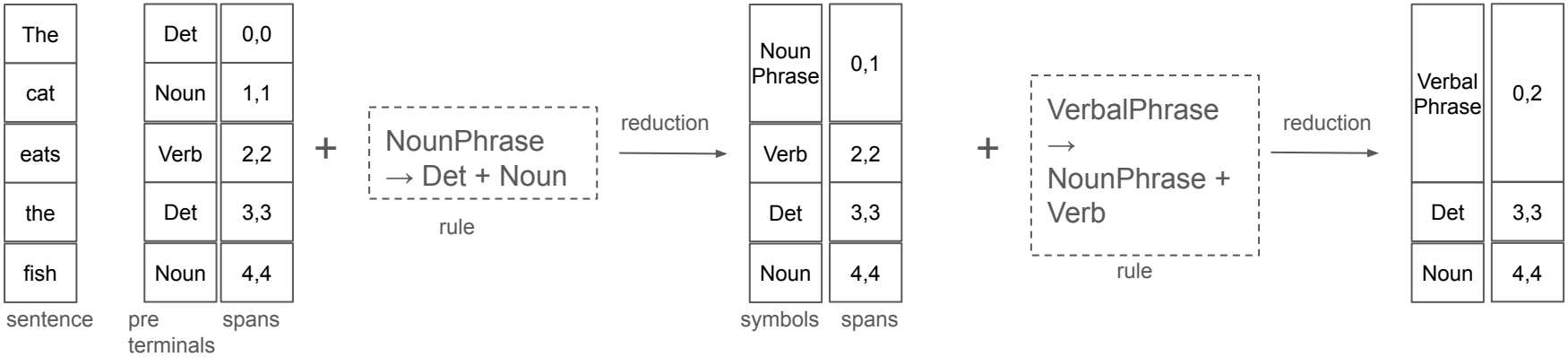
Solution : use reinforcement learning to build an explainable model where exploration replaces dynamic programming and from which we can build a CFG !

Parsing method

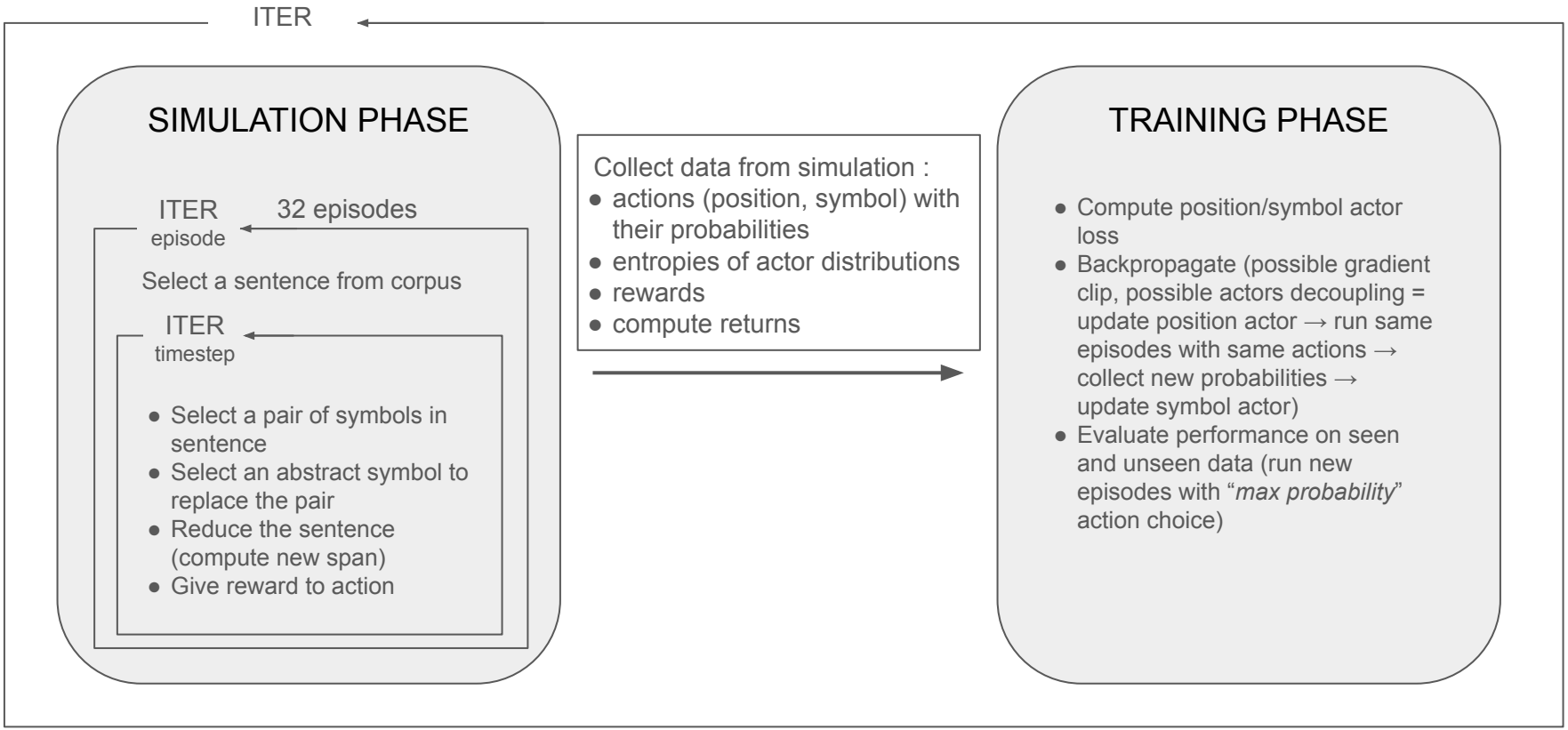
Z. Solan, D. Horn, E. Ruppin, & S. Edelman, Unsupervised learning of natural languages, Proc. Natl. Acad. Sci. U.S.A. (2005).

Bottom-up parsing :

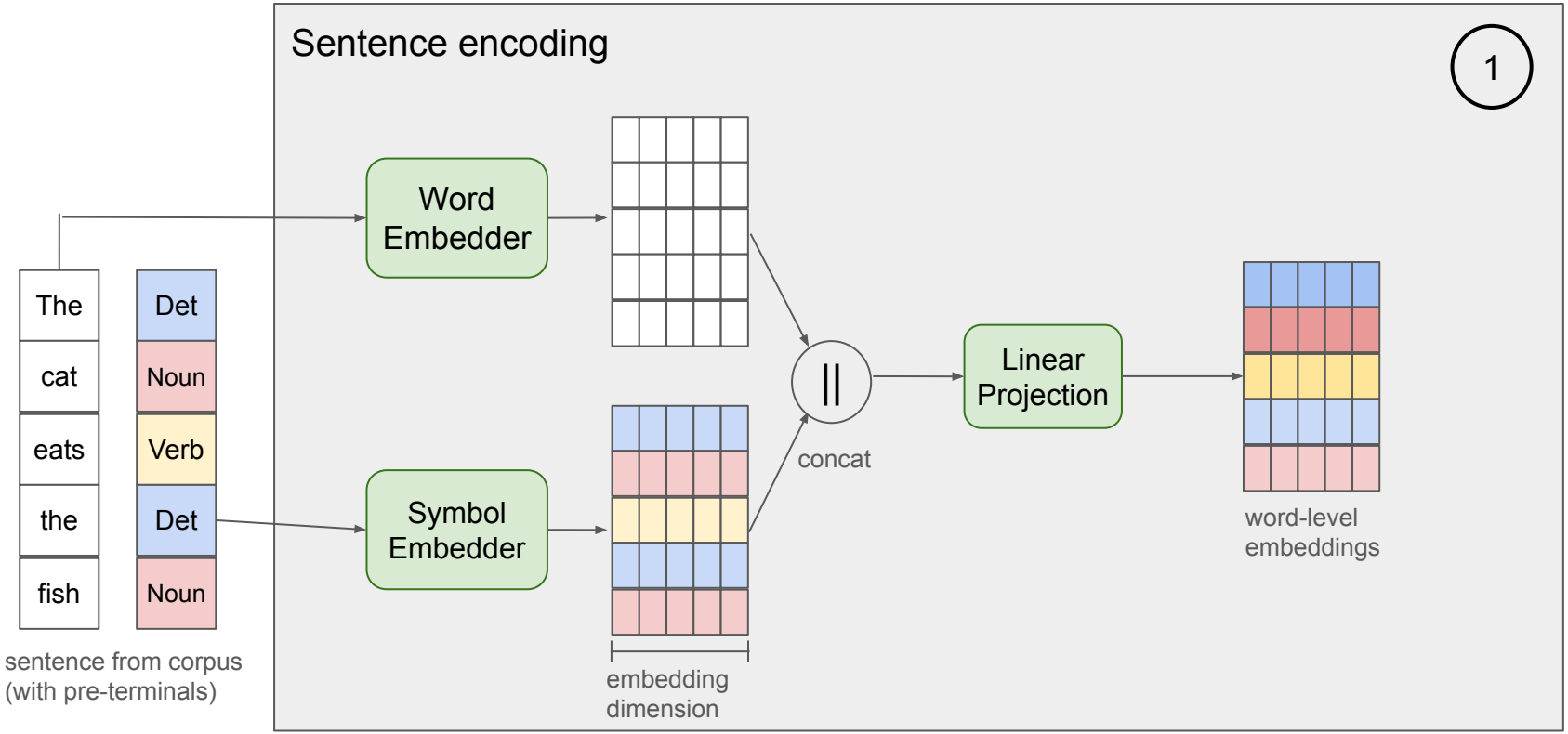
- start from sequence of pre-terminals
- end with only start symbol
- iteratively apply a rule to a pair of succeeding symbols in the sequence (RHS) and replace the pair with the LHS of the rule



REINFORCE algorithm



The embedding of the sentence



Word embedder (2 options)

Pre-trained model (BERT): each word's string is tokenized and embedded

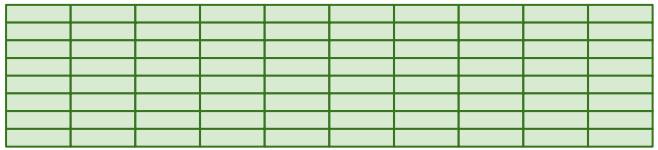
Embedding layer :

- word-to-index dictionary for N most frequent words + 1 special index for unknown words
- table of N+1 trainable embeddings randomly initialized

MEMORY

Integer/Word dictionnary

UNK	the	to	a	this	one	it	can	…	small
0	1	2	3	4	5	6	7	…	N



Learnable embeddings

Symbol embedder

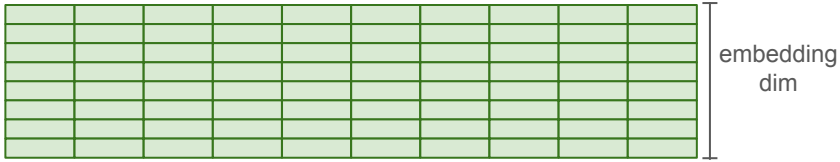
Embedding layer :

- symbol-to-index dictionary with 14 Part-of-Speech tags + K placeholders for non-terminals
- table of 14+K trainable embeddings randomly initialized

MEMORY

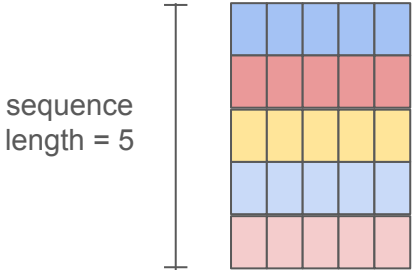
Integer/Symbol dictionary

NT0	NT1	NT2	…	NTK	Adj	Prep	Verb	…	Noun
0	1	2	…	K-1	K	K+1	K+2	…	K+13



Learnable embeddings

Initial state

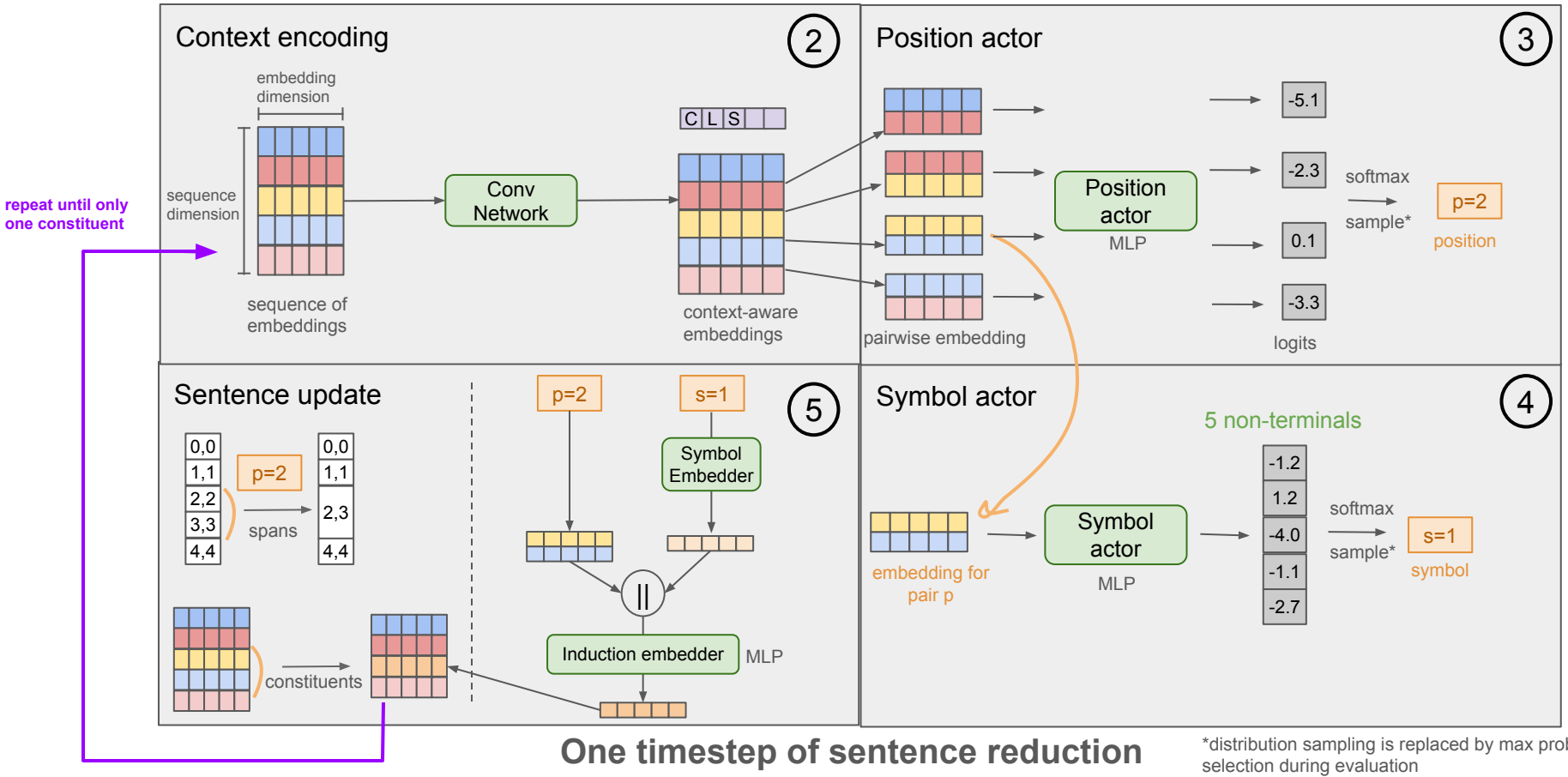


sequence of
word-level
embeddings



sequence
of spans

The overall architecture



One timestep of sentence reduction

*distribution sampling is replaced by max probability selection during evaluation

Reward

During sentence update (5)

Each sentence in the dataset comes with a set of ground-truth spans

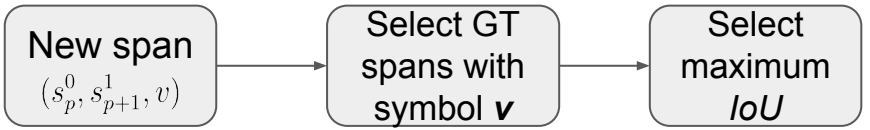
ex : $S^{GT} = \{(Sentence, 0, 4), (NounPhrase, 0, 1), (VerbalPhrase, 2, 4), (NounPhrase, 3, 4)\}$

Two reward frameworks (given position choice ***p***, symbol choice ***v***, and span (s_p^0, s_{p+1}^1)) :

- semi-supervised : only position actor is rewarded. This reward is used for position actor loss and symbol actor loss.
- fully supervised : position reward + symbol reward



$$reward(p) = \mathbf{1}[(s_p^0, s_{p+1}^1) \in S_{GT}]$$

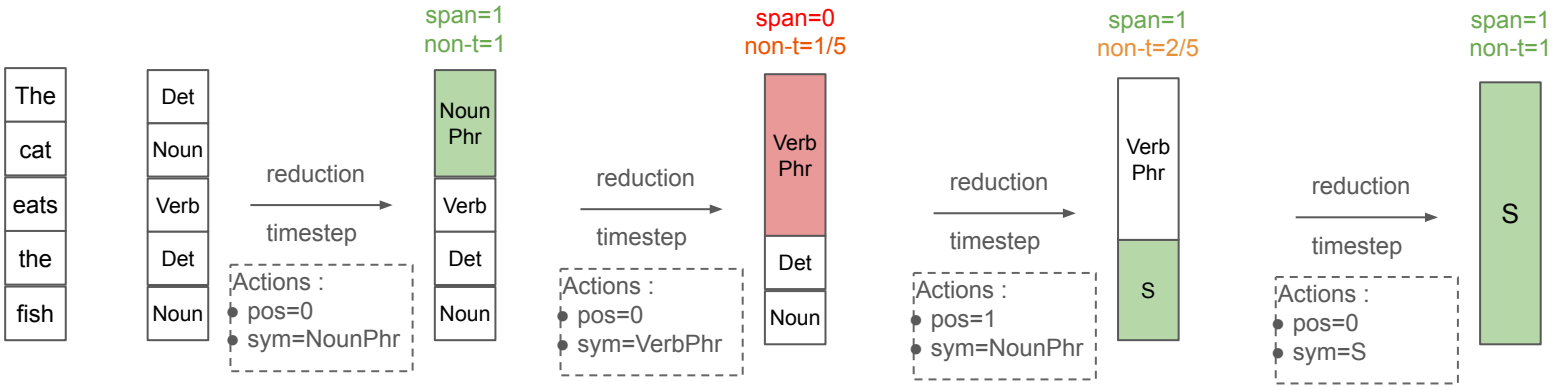


$$reward(v|p) = \max_{\bar{s} \in S_{GT}, NT(\bar{s}) = v} IoU(\bar{s}, (s_p^0, s_{p+1}^1))$$

Simulation phase

One episode consists in reducing a sentence from corpus to one symbol
A simulation phase consists in running a number E=32 of episodes

S_GT = {(NounPhrase,0,1), (VerbalPhrase,2,4), (NounPhrase,2,4), (S,0,4)}



Processing 1 sentence = 1 episode with

Training phase

For each episode we collect the data of all timesteps (position/symbol actions with the probabilities and rewards)

$$a_t^{pos}, \pi^{pos}(a_t^{pos}), r_t^{pos}, a_t^{sym}, \pi^{sym}(a_t^{sym}), r_t^{sym}$$

and we compute returns $R_t = \sum_{l=0}^{T-t} \gamma^l r_{t+l}$

In the case of semi-supervised rewards, we have $R_t^{sym} = R_{t+1}^{pos}$

$$L_{pos}(\theta) = -\mathbb{E}_t[\pi^{pos}(a_t^{pos})R_t + c_{ent}e_t^{pos}]$$

$$L_{sym}(\theta) = -\mathbb{E}_t[\pi^{sym}(a_t^{sym})R_t + c_{ent}e_t^{sym}]$$

$$L(\theta) = L_{pos}(\theta) + L_{sym}(\theta)$$

Results

We evaluate the parsing model with **unlabeled F1 score** on binarized trees

Our first framework is full-supervision (we use the annotation of spans with labels from PTB)

Model can achieve 91% F1 score (best models are 96% on non-binarized trees) which is strong

If we move to semi-supervised setting (spans annotation, no labels), accuracy is the same, meaning non-terminal labelling is useless when there is span annotation (confirmed by experiment where symbol actor can only choose one symbol)

The reinforcement learning framework is not really adapted to supervised learning : setting gamma factor to 0 is the condition to achieve maximum F1 score !

Results

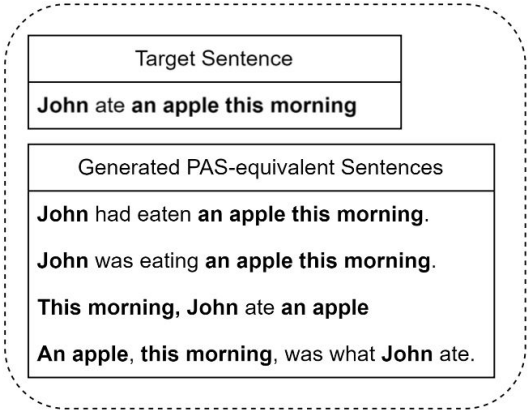
towards unsupervised learning…

We also try a very simple weakly-supervised framework (only POS tag annotation) where position actor is rewarded based on the statistical probability of the POS-tag sequence defined by the span (n-gram model)

This rewarding method is close to modern works : for each sentence ***x*** an LLM is used to create equivalent sentences and see which sub-sequence of words are the most frequents

In this framework, gamma factor becomes important !

With good gamma factor setting we can achieve 59% F1 score, which is a promising first step !

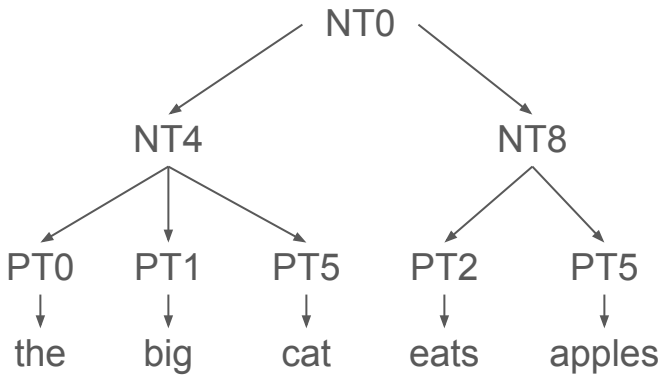


CFG induction from the model

Trivial way of getting a CFG from the model:

⚠ By default we only get binary CFG

- ➢ Parse training corpus with model
- ➢ Collect all reduction rules applied
- ➢ Optionally prune very rare rules (potential noise from model)



[NT0→NT4+NT8,
NT4→PT0+PT1+PT5,
NT8→PT2+PT5,
PT0→'the',
PT1→'big',
PT5→'cat',
PT2→'eats',
PT5→'apples']

Toy grammar setting

Parameters : *n_non_terminals, n_pre_terminals, n_terminals, n_production_rules*

We generate N random “target” CFGs with given parameters. We purposely choose to produce n-ary rules, and potential ambiguity in terminals (several pre-terminals give same terminal)

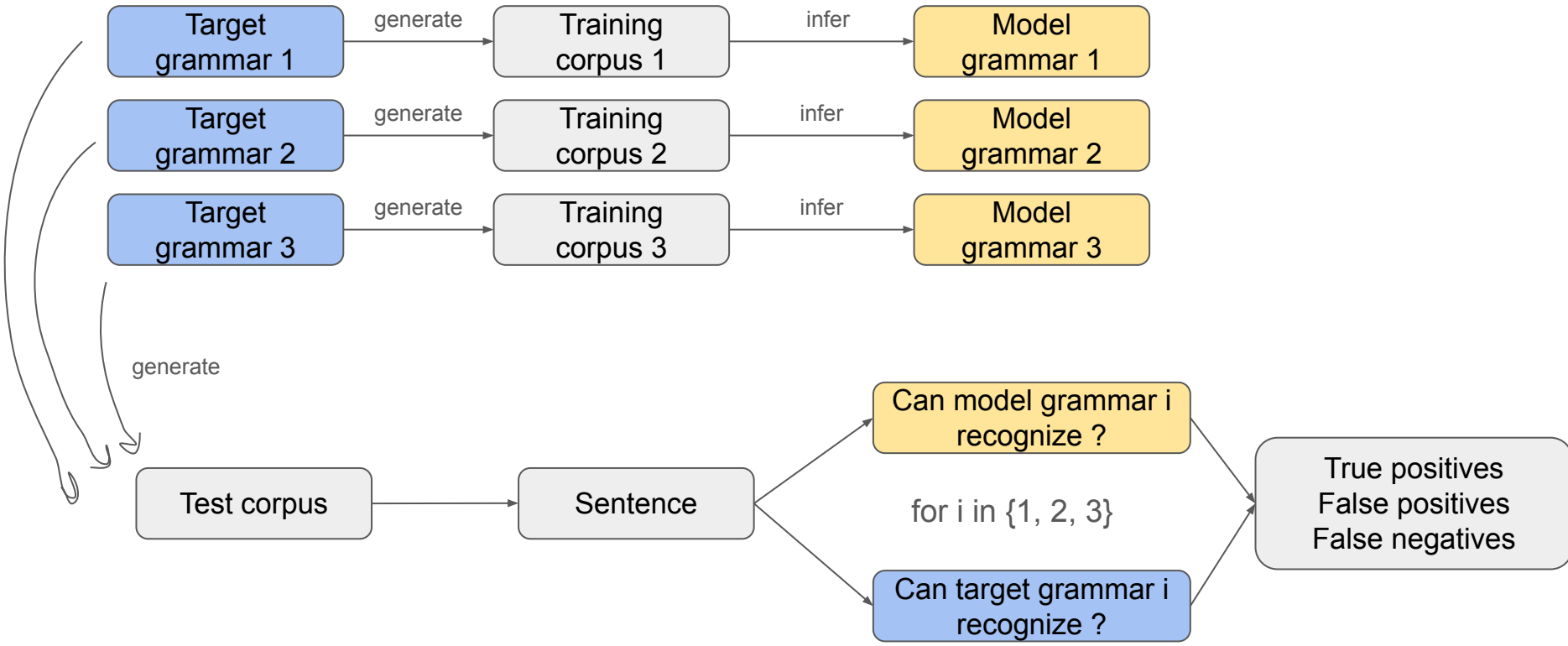
We generate one training corpus and one test corpus for each target CFG

Model is trained on each training corpus, and produces N “model” CFGs

For each target grammar, we read all test corpuses and we collect true positives, false positives and false negatives.

We then compute coverage F1 score

Toy grammar setting



Results

Results are not conclusive for now…

Possible culprits are :

- too simple choice of binarization : current choice is **parent-label right-branching**
- Statistical analysis of rules for merging, pruning…