Parsing as language modelling

Séminaire Cental

Benoît Crabbé

Université de Paris

Laboratoire de linguistique formelle

IUF

12 décembre 2019

Overall question/problem

- Design of a computational model of human sentence processing that can predict human behaviour
- Interested in testing the structural hypothesis:
  - Sequential models without explicit structural biases
  - Hierarchical (tree-like) models with tree structure and recursivity

Deep learning context

The discussion was somehow settled by Chomsky in the 1950's but there is a resurgence caused by deep learning models such as LSTM and attention based models.

Plan

Language modelling

- Let a string $\mathbf{x} = x_1 \ldots x_n$ be a sequence of n words.
- Let $\mathcal{X}$ be a **formal language**, that is a (possibly infinite) set of strings. A language model assigns a probability to each string $\mathbf{x} \in \mathcal{X}$ such that :

$$\sum_{\mathbf{x} \in \mathcal{X}} P(\mathbf{x}) = 1$$

- By applying the chain rule of conditional probabilities, we get :

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i | x_1 \ldots x_{i-1})$$

Example

$$P(a, b, c, d, e) = P(a)P(b|a)P(c|a, b)P(d|a, b, c)P(e|a, b, c, d)$$

n-gram language models

- Factors involved in the chain rule grow up in size as the sequence gets longer (see $P(e|a, b, c, d)$ above). It is common place to approximate these longer factors by shorter ones, ex. $P(e|c, d)$.
- Capping the size of those factors to some size k is called a **markov assumption** of order k. Under this assumption, the probability of a sentence is computed as:

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i|x_{i-k} \ldots x_{i-1})$$

  when $k = 1$ we say that the model is **bigram** and when $k = 2$ we say that the model is **trigram**.

Example

$P(c|a, b)$ is a trigram, $P(b|a)$ is a bigram.

Estimation of an n-gram model

- An n-gram model is usually estimated from data where :

$$P(x_i|x_{i-k}\ldots x_{i-1}) = \frac{\text{count}(x_{i-k}\ldots x_i)}{\text{count}(x_{i-k}\ldots x_{i-1})}$$

- In practice, this involves counting long sequences of words if the order k of the model is large.
- This is quickly getting largely problematic when $k > 3$ (and already problematic with $k \in \{1,2\}$ because some sequences remain unseen in data, as a consequence of Zipf law (Smoothing methods).

Example

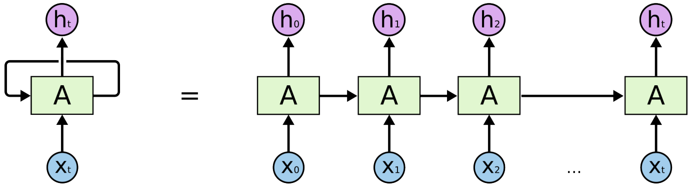
$P(c|a,b) = \frac{\text{count}(a,b,c)}{\text{count}(a,b)}$, $P(b|a) = \frac{\text{count}(a,b)}{\text{count}(a)}$

Neural language models

- Neural language models do not make the markov assumption.
- Rather a neural language model uses a parametric function, a neural network, to compute the probabilities of the conditionals $P(x_i | x_1 \ldots x_{i-1})$
- An example of such network is the recurrent neural network (RNN), that is a function of the form :

$$\begin{aligned} \mathbf{h}_{i-1} &= g(\mathbf{W}_h \mathbf{h}_{i-2} + \mathbf{W}_e \mathbf{e}(x_{i-1}) + \mathbf{b}_r) && \text{(recurrence)} \\ P(x_i | x_1 \ldots x_{i-1}) &= \text{softmax}(\mathbf{W}_o \mathbf{h}_{i-1} + \mathbf{b}_o) && \text{(output)} \end{aligned}$$

Example RNN



Practical considerations

In practice, vanilla RNNs suffer the gradient vanishing problem during parameter estimation. We work with LSTMs or GRU networks instead. But this level of granularity is irrelevant in this talk.

Evaluating language models

- It is often of interest to measure the amount of uncertainty underlying the prediction of the next word for a language model. This can be measured with the quantity of information :

$$-log_2 P_\theta(x_i | x_1 \dots x_{i-1})$$

and can be made global for a whole corpus $\mathbf{x} = x_1 \dots x_n$ as

$$-\frac{1}{N} \sum_{i=1}^{N} log_2 P_\theta(x_i | x_1 \dots x_{i-1})$$

- The metric used is called **perplexity**, and computed as :

$$PPL(\mathbf{x}) = 2^{-\frac{1}{N} \sum_{i=1}^{N} log_2 P_\theta(x_i | x_1 \dots x_{i-1})}$$

... the lower, the better ($\sim$ the model is less perplex)

Plan

Parsing as language modelling

- Recall that a language model computes the probability of a sentence with :

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i | x_1 \ldots x_{i-1})$$

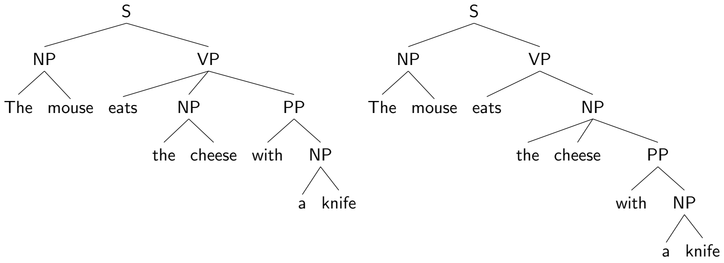
- Instead of computing the factors $P(x_i | x_1 \ldots x_{i-1})$ like if words were just raw sequences, we introduce an additional assumption :

Sentences have underlying hidden tree structures !

- Language modelling with this additional treeness assumption will essentially provide a new way to compute the factors $P(x_i | x_1 \ldots x_{i-1})$

Trees and ambiguity

- A single sentence can be generated by several trees:



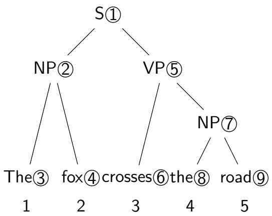
  where structural difference may also involve a semantic difference

- **No grammaticality**. Current parsers do not assume a grammar. Every tree covering the full sentence is potentially a good candidate. ($\Rightarrow$ combinatorial explosion)

Encoding trees as sequences

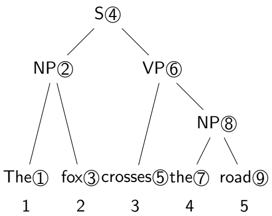
Tree traversals

- All generative parsing algorithms for language modelling encode trees as sequences of actions, called **derivations**. A particular parsing algorithm can be understood as a way to generate trees and their words according to a particular tree traversal:



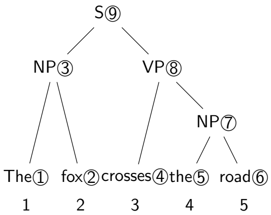
Preorder traversal

Top down parser



In-order traversal

Left corner parser



Postorder traversal

bottom up (shift reduce) parser

Tree traversals as transition systems

- The parsing state is a couple $(\mathbf{S}, \mathbf{B})$ where $\mathbf{S}$ is a Stack and $\mathbf{B}$ a Buffer.
- The parser moves from state to state by performing actions. It starts on an initial state and its goal is to reach a final state
- Here is a possible transition system for top down tree traversal:

$$\begin{array}{rc}
\mathit{Pred}(X) & \dfrac{(\mathbf{S}, \mathbf{B})}{(\mathbf{S}|(X, \mathbf{B})} \\
\mathit{Shift}(w) & \dfrac{(\mathbf{S}, w|\mathbf{B})}{(\mathbf{S}|w, \mathbf{B})} \\
\mathit{Reduce} & \dfrac{(\mathbf{S}|(X \ldots Y|\mathbf{B})}{(\mathbf{S}|X, \mathbf{B})} \\
\mathit{Init} & (\emptyset, w_1 \ldots w_n) \\
\mathit{End} & (X, \emptyset)
\end{array}$$

Example

Stack	Buffer	Action
∅	The , fox, crosses, the, road	Pred(S)
(S	The , fox, crosses, the, road	Pred(NP)
(S (NP	The , fox, crosses, the, road	Shift(The)
(S (NP The	fox, crosses, the, road	Shift(fox)
(S (NP The fox	crosses, the, road	Reduce
(S NP	crosses, the, road	Pred(VP)
(S NP (VP	crosses, the, road	Shift(crosses)
(S NP (VP crosses	the, road	Pred(NP)
(S NP (VP crosses (NP	the, road	Shift(the)
(S NP (VP crosses (NP the	road	Shift(road)
(S NP (VP crosses (NP the road	∅	Reduce
(S NP (VP crosses NP	∅	Reduce
(S NP VP	∅	Reduce
S	∅	goal !

The action column is the derivation (**parse history**)

We can retrieve the actual **parse tree** from a derivation.

Non determinism and combinatorial explosion

- At each time step during inference, the parser has to choose an action to perform in the set A of actions. This set is the union of :
  - All Pred actions (one Pred action/Non terminal)
  - All Shift action (one Shift action/terminal).
  - One reduce action

  Thus the search space naturally increases exponentially in size with the length of the derivation even if some constraints apply. For instance input constraints on shift are obvious.
- Most parsers use a scoring method as support for the decision procedure. Here we provide some deep learning based methods.

History based models

- Since the derivation (history) is a sequence $\mathbf{a} = a_1 \ldots a_m$, just use an RNN to predict the next action :

$$\begin{aligned} \mathbf{h}_{i-1} &= g(\mathbf{W}_h \mathbf{h}_{i-2} + \mathbf{W}_e \mathbf{e}(a_{i-1}) + \mathbf{b}_r) && \text{(recurrence)} \\ P(a_i | a_1 \ldots a_{i-1}) &= \text{softmax}(\mathbf{W}_o \mathbf{h}_{i-1} + \mathbf{b}_o) && \text{(output)} \end{aligned}$$

- Then the probability of a derivation $P(a_1 \ldots a_m)$ is computed with the chain rule :

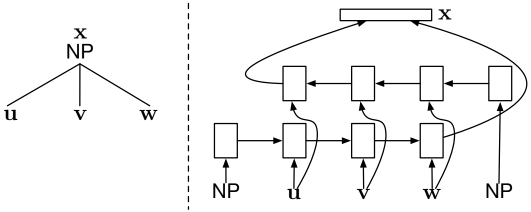
$$P(a_1 \ldots a_m) = \prod_{i=1}^{m} P(a_i | a_1 \ldots a_{i-1})$$

- That's the method used by (Choe and Charniak 2016; Stern et al 2017)

Configuration based models

Input vectorisation

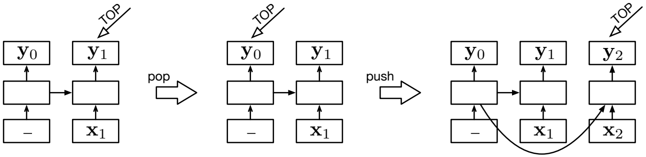
- Instead of using the history, one could use the information found in a given configuration $(\mathbf{S}, \mathbf{B})$ to inform the decision.
- To do that we have to vectorize the configuration.
  - On the buffer, we cannot look there if we want to be incremental
  - On the stack there are words, open constituents, closed constituents.
  - Words and open constituents can be vectorized by **lexical embeddings**
  - More complex is the representation of closed constituents where we want to use **tree embeddings**. Here is one way to do that:



Configuration based models

Stack-RNN

- Basic idea : use an RNN to encode the stack with inputs **x** the vectorized contents of the stack
- Contrary to standard RNN that keeps pushing on top of the sequence, a full parser also has to pop stack elements. Thus we need to implement a pop function for RNN
- Here is how this is done (Dyer et al. 2015) :



This is called a **stack-lstm**. This can be implemented only with libraries using a dynamic computation graph such as DyNet and Pytorch.

Configuration based models

Adding the softmax

- Let's call $\mathbf{h}_i$ the hidden state of the stack-lstm at the $i-$th inference step, then the probability decision on the next action can be computed as :

$$P(a_i|a_1 \ldots a_{i-1}) = \text{softmax}(\mathbf{W}\,\mathbf{h}_i + \mathbf{b})$$

The RNNG model

It plugs everything together

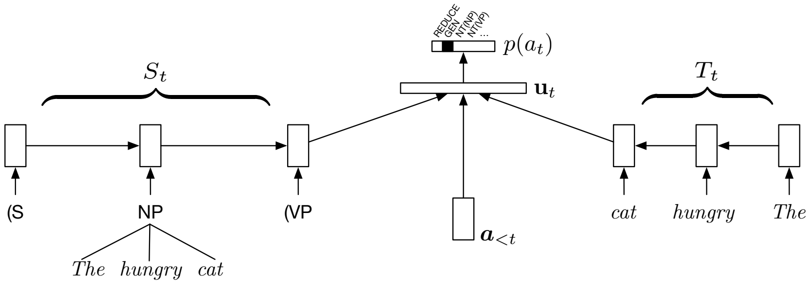
- RNNG combines 3 representations : an history based model encoded with an RNN, a configuration based model encoded with a STACK-RNN and a linear model encoding linearily the input with a RNN. Thus the overall model is defined as :

$$
\begin{aligned}
\mathbf{h}_i &= \text{rnn}_H(a_{i-1}) \\
\mathbf{s}_i &= \text{stack-rnn}_H(\mathbf{x}_{top}) \\
\mathbf{L}_i &= \text{rnn}_L(\mathbf{e}_{last}) \\
\mathbf{u}_i &= g\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_i \\ \mathbf{s}_i \\ \mathbf{L}_i\end{bmatrix} + \mathbf{b}\right) \\
P(a_i|a_1 \ldots a_{i-1}) &= \text{softmax}(\mathbf{W}_o\mathbf{u}_i + \mathbf{b}_o)
\end{aligned}
$$

where w_i is the last word shifted at inference step i

The RNNG model

Illustration



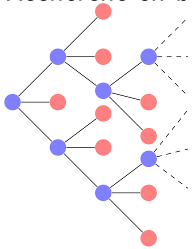
Probability of a derivation

Note that the probability of a derivation $P(\mathbf{x}, \mathbf{y})$ still decomposes as :

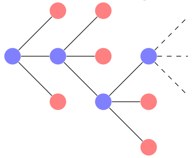
$$P(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{m} P(a_i | a_1 \dots a_{i-1})$$

Parsing and search

- Parsing amounts to explore the search the tree. Traditional methods are beam search (and sometimes even greedy search).
- Recherche en beam :



- Recherche gloutonne :



Plan

The fundamental problem of generative parsing

The fundamental problem

- Beam search does not work ! (as is)
- Cause ? **lexical biases**, lexical transition probabilities (shift/generate actions) are much lower than structural transition probabilities (reduce, predict)
- Consequences : derivations that generate lexical items early are pruned out of the beam. (complex structure at the beginning of the sentence)

Possible solutions

- Reranking architecture (Dyer et al 2016; Choe and Charniak 2016).
- Word synchronous Beam Search (Stern et al 2017; Hale et al. 2018)
- Particle filtering inspired Beam Search (Crabbé, Fabre, Pallier 2019)

Reranking architecture
(related work)

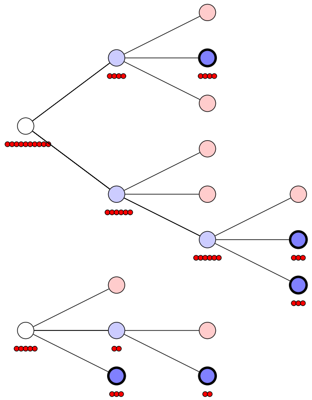
- Initially (Dyer et al 2016; Charniak and Choe 2016) framed generative neural parsing as a two stage process :
  1. Parse with a discriminative parser and get K derivations
  2. Rescore those K derivations with the generative model (reranking)
- A discriminative parser does not have the lexical bias problem. Indeed words are given and, as a result, there is just one shift action (with predict and reduce actions)

Variable beam search

the idea

- The previous solution (reranking) breaks incrementality.
- Crabbé, Fabre and Pallier 2019 proposed to use an alternative search method that progresses iteratively from word x_i to word x_{i+1}. The move from w_i to w_{i+1} proceeds as follows :
    1. All derivations in the beam $(\mathbf{x}, \mathbf{y}) \in \mathcal{Y}(\mathbf{x}_i)$ that successfully generated the sentence up to word x_i are weighted with a finite budget K of particles that is spread proportionally to their probabilities. Derivations without particles are pruned from the search.
    2. The derivations are expanded until they generate x_{i+1} the next word or are pruned from the search when the number of particles associated to the derivation is 0.

Illustration



Example of a sampling step from derivations generating x_i (white nodes) to derivations generating x_{i+1} (circled blue nodes) with a budget of $K = 15$ particles. Inference may stop because of a lack of budget as illustrated by red nodes. Derivations are never compared to each other during the sampling step, hence avoiding lexical biases that hamper the process of beam search.

Sequential importance sampling

Sampling step

- The probability distribution to sample from cannot be $P(a|\mathbf{x}_{<i}, \mathbf{y}_{<i})$
- The reason comes from the fact that we can generate **only a single word** at the next time step (by construction of the parsing task)
- We rather use an importance distribution:

$$P^*(a|\mathbf{x}_{<i}, \mathbf{y}_{<i}) = \frac{P(a|\mathbf{x}_{<i}, \mathbf{y}_{<i})}{\sum_{a' \in A^*} P(a'|\mathbf{x}_{<i}, \mathbf{y}_{<i})}$$

- Search tree growth (particle sampling):

$$\pi(\mathbf{x}, \mathbf{y}a) = \lfloor \pi(\mathbf{x}, \mathbf{y}) P^*(a|\mathbf{x}_{<i}, \mathbf{y}_{<i}) \rceil$$

Reweighting

- Weighting partial derivations:

$$w(\mathbf{x}_i, \mathbf{y}_i) = \frac{P(\mathbf{x}_i, \mathbf{y}_i)}{\sum_{\mathbf{y}' \in \tilde{\mathcal{Y}}(\mathbf{x}_i)} P(\mathbf{x}_i, \mathbf{y}')}$$

- Given a constant budget K we reallocate each derivation a number of particles such that:

$$\pi(\mathbf{x}_i, \mathbf{y}_i) = \lfloor K w(\mathbf{x}_i, \mathbf{y}_i) \rceil$$

Parsing task

This amounts to predict the most probable derivation $(\mathbf{x}, \mathbf{y})$ (encoded tree) with max probability :

$$(\mathbf{x}, \hat{\mathbf{y}}) = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\operatorname{argmax}} P(\mathbf{x}, \mathbf{y})$$

Parsing for language modelling

In our current notation the **prefix probability** of a string prefix up to word x_i is computed by marginalizing out the parses :

$$P(\mathbf{x}_i) = P(x_1 \ldots x_i) = \sum_{\mathbf{y} \in \mathcal{Y}(\mathbf{x}_i)} P(\mathbf{x}_i, \mathbf{y}_i)$$

where the probability of a derivation $P(\mathbf{x}_i, \mathbf{y}_i)$ is the probability of the sequence of actions generating jointly the words **x** and structure **y**.

$$P(\mathbf{x}, \mathbf{y}) = P(a_1 \ldots a_m) = \prod_{i=1}^{m} P(a_i | a_1 \ldots a_{i-1})$$

Frow where one can get conditionals of the form $P(x_i | x_1 \ldots x_{i-1}) = \frac{P(x_1 \ldots x_i)}{P(x_1 \ldots x_{i-1})}$ by applying basic definitions of conditional probabilities.

Tests
Parsing

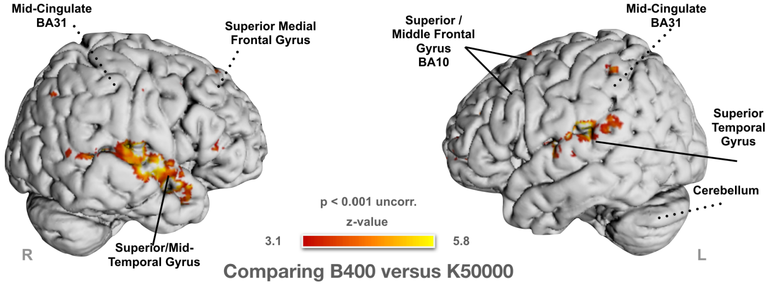
- Tests run on the Penn Treebank ($\approx$ 40000 sentences for train) and 2400 for test.

Model	F-score	Ppl (wsj)	Ppl (prince)
K=50000	91.02	94.35	154.93
IKN5	-	155.02	309.54
LSTM-LM	-	141.28	204.06
(Dyer et al. 2016)	93.3	105.2	unknown
(Kitaev et al. 2018)	**93.55**	-	-

Parsing and human behaviour

- The parser outputs measures of the model behaviour using word-synchronized pattern
- There are two families of measures output by the parser:
  - **Beam size measures**
    (beam successful activity, beam unsuccessful activity, total beam activity)
  - **Language model measures**
    (entropy,surprisal,conditional log probabilities. . .)
- These measures can be used as input (of a GLM) to predict brain activity (or some other forms of human behaviour)

Illustration



Plan

Observation

Data set is small

- On current Penn Treebank standards ($\sim$ 40000 sentences), parsers are pretty good language models (CFP 2019) :

MODEL	F-SCORE	PPL (wsj)	PPL (prince)
K=50000	**91.02**	94.35	154.93
IKN5	-	155.02	309.54
LSTM-LM	-	141.28	204.06

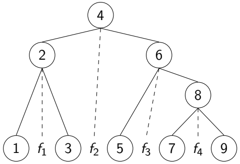
- But are pretty bad in the language modelling world if we consider very large scale language models ala (Jozefowicz et al. 2016) that get a perplexity around 23.2 (although the numbers cannot be truly compared since the data sets are different).

Weakly-supervised parsing

- Motivation similar to unsupervised learning. Scaling up parsers and reducing the dependence to annotated data
- In weakly supervised learning we keep some supervision to make the process less computationnaly expensive.
- Different ways to reduce supervision :
    1. Semi-supervised model with guiding parser assistance (Choe and Charniak 2016)
    2. Full unsupervised learning, such as RNNG (Kim et al. 2019). Inference methods too expensive, will hardly scale up.
    3. Built-in semi-supervised model (my current idea). Amounts to frame a parsing model very similar to an LSTM-LM
    4. STACK-LSTM (Joulin and Mikolov 2015)

Strong left corner encoding of a tree

- The idea is to rely on the following encoding of a binary tree (Rozenkrantz and Lewis 1970,Kitaev et al. 2019):



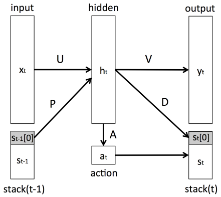
  the tree traversal that has the property to be:

  1. Strongly **incremental**
  2. Strictly alternates the use of lexical and structural nodes
  3. The number of actions to traverse the tree can be predicted from the size of the input $(2n-1)$

- Provided word embeddings, properties (2) and (3) allows to train conveniently a parser almost like an LSTM or GPT tagger on a GPU
- This is easily amenable to weakly supervised learning from raw text, where words are used as partial supervision.

Stack LSTM

- The STACK RNN or STACK LSTM is an RNN (resp. LSTM) augmented with a neural stack and a controller. At each time step the controller will make a soft prediction on actions such as Push, Pop, No-Op. The controller will update the content of the stack accordingly:



Expected advantage

The advantage of those methods comes from the fact that raw text is largely sufficient (?) to train these models.