Parsing as language modelling
Séminaire Cental

Benoît Crabbé

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

1. Language modelling
2. Generative neural parsing
3. Variable beam search
4. Unsupervised perspectives
Language modelling

Let a string \( x = x_1 \ldots x_n \) 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 \( x \in \mathcal{X} \) such that:

\[
\sum_{x \in \mathcal{X}} P(x) = 1
\]

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

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

**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:

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

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

\[ P(c|a, b) = \frac{\text{count}(a, b, c)}{\text{count}(a, b)}, \quad P(b|a) = \frac{\text{count}(a, b)}{\text{count}(a)} \]
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 ($\text{RNN}$), that is a function of the form:

$$h_{i-1} = g(W_h h_{i-2} + W_e e(x_{i-1}) + b_r) \quad \text{(recurrence)}$$

$$P(x_i|x_1 \ldots x_{i-1}) = \text{softmax}(W_o h_{i-1} + b_o) \quad \text{(output)}$$
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:

\[- \log_2 P_\theta(x_i|x_1 \ldots x_{i-1})\]

and can be made global for a whole corpus \(\mathbf{x} = x_1 \ldots x_n\) as

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

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

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

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

1. Language modelling
2. Generative neural parsing
3. Variable beam search
4. Unsupervised perspectives
Recall that a language model computes the probability of a sentence with:

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

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 $P(x_i|x_1 \ldots x_{i-1})$
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)
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 \((S, B)\) where \(S\) is a Stack and \(B\) 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:

\[
\begin{align*}
\text{Pred}(X) & \quad \frac{(S, B)}{(S|X, B)} \\
\text{Shift}(w) & \quad \frac{(S, w|B)}{(S|w, B)} \\
\text{Reduce} & \quad \frac{(S|(X \ldots Y|B)}{(S|X, B)} \\
\text{Init} & \quad (\emptyset, w_1 \ldots w_n) \\
\text{End} & \quad (X, \emptyset)
\end{align*}
\]
## Example

<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>The, fox, crosses, the, road</td>
<td>Pred(S)</td>
</tr>
<tr>
<td>(S)</td>
<td>The, fox, crosses, the, road</td>
<td>Pred(NP)</td>
</tr>
<tr>
<td>(S (NP)</td>
<td>The, fox, crosses, the, road</td>
<td>Shift(The)</td>
</tr>
<tr>
<td>(S (NP The)</td>
<td>fox, crosses, the, road</td>
<td>Shift(fox)</td>
</tr>
<tr>
<td>(S (NP The fox)</td>
<td>crosses, the, road</td>
<td>Reduce</td>
</tr>
<tr>
<td>(S NP)</td>
<td>crosses, the, road</td>
<td>Pred(VP)</td>
</tr>
<tr>
<td>(S NP (VP)</td>
<td>crosses, the, road</td>
<td>Shift(crosses)</td>
</tr>
<tr>
<td>(S NP (VP crosses)</td>
<td>the, road</td>
<td>Pred(NP)</td>
</tr>
<tr>
<td>(S NP (VP crosses (NP</td>
<td>the, road</td>
<td>Shift(the)</td>
</tr>
<tr>
<td>(S NP (VP crosses (NP the</td>
<td>road</td>
<td>Shift(road)</td>
</tr>
<tr>
<td>(S NP (VP crosses (NP the road</td>
<td>$\emptyset$</td>
<td>Reduce</td>
</tr>
<tr>
<td>(S NP (VP crosses NP</td>
<td>$\emptyset$</td>
<td>Reduce</td>
</tr>
<tr>
<td>(S NP VP</td>
<td>$\emptyset$</td>
<td>Reduce</td>
</tr>
<tr>
<td>S</td>
<td>$\emptyset$</td>
<td>goal !</td>
</tr>
</tbody>
</table>

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 $a = a_1 \ldots a_m$, just use an $\text{RNN}$ to predict the next action:

  $$h_{i-1} = g(W_h h_{i-2} + W_e e(a_{i-1}) + b_r) \quad \text{(recurrence)}$$
  $$P(a_i|a_1 \ldots a_{i-1}) = \text{softmax}(W_o h_{i-1} + b_o) \quad \text{(output)}$$

- 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 \((S, B)\) 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-Istmp**. This can be implemented only with libraries using a dynamic computation graph such as DyNet and Pytorch.
Let's call $h_i$ 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:

$$P(a_i|a_1 \ldots a_{i-1}) = \text{softmax}(W h_i + b)$$
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 linearly the input with a **RNN**. Thus the overall model is defined as:

\[
\begin{align*}
    h_i &= \text{rnn}_H(a_{i-1}) \\
    s_i &= \text{stack-rnn}_H(x_{top}) \\
    L_i &= \text{rnn}_L(e_{last}) \\
    u_i &= g \left( W \begin{bmatrix} h_i \\ s_i \\ L_i \end{bmatrix} + b \right) \\
    P(a_i | a_1 \ldots a_{i-1}) &= \text{softmax}(W_o u_i + b_o)
\end{align*}
\]

where \(w_i\) is the last word shifted at inference step \(i\)
The RNNG model

Illustration

The RNNG model

Illustration

Probability of a derivation

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

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

1. Language modelling
2. Generative neural parsing
3. Variable beam search
4. Unsupervised perspectives
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)
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 $(x, y) \in \mathcal{Y}(x_i)$ 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 $P(a|x_{<i}, y_{<i})$
- 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:

$$P^*(a|x_{<i}, y_{<i}) = \frac{P(a|x_{<i}, y_{<i})}{\sum_{a'\in A^*} P(a'|x_{<i}, y_{<i})}$$

- Search tree growth (particle sampling):

$$\pi(x, ya) = \lceil \pi(x, y) P^*(a|x_{<i}, y_{<i}) \rceil$$
Reweighting

- Weighting partial derivations:

\[ w(x_i, y_i) = \frac{P(x_i, y_i)}{\sum_{y' \in \tilde{Y}(x_i)} P(x_i, y')} \]

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

\[ \pi(x_i, y_i) = \lceil Kw(x_i, y_i) \rceil \]
This amounts to predict the most probable derivation \((x, y)\) (encoded tree) with max probability:

\[
(x, \hat{y}) = \arg\max_{y \in \mathcal{Y}(x)} P(x, y)
\]
In our current notation the **prefix probability** of a string prefix up to word $x_i$ is computed by marginalizing out the parses:

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

where the probability of a derivation $P(x_i, y_i)$ is the probability of the sequence of actions generating jointly the words $x$ and structure $y$.

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

From 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

Tests run on the Penn Treebank (≈ 40000 sentences for train) and 2400 for test.

<table>
<thead>
<tr>
<th>Model</th>
<th>F-score</th>
<th>PPL (wsj)</th>
<th>PPL (prince)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=50000</td>
<td>91.02</td>
<td>94.35</td>
<td>154.93</td>
</tr>
<tr>
<td>IKN5</td>
<td>-</td>
<td>155.02</td>
<td>309.54</td>
</tr>
<tr>
<td>LSTM-LM</td>
<td>-</td>
<td>141.28</td>
<td>204.06</td>
</tr>
<tr>
<td>(Dyer et al. 2016)</td>
<td>93.3</td>
<td>105.2</td>
<td>unknown</td>
</tr>
<tr>
<td>(Kitaev et al. 2018)</td>
<td>93.55</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
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

Comparing B400 versus K50000

- Mid-Cingulate BA31
- Superior Medial Frontal Gyrus
- Superior/Mid-Temporal Gyrus
- Superior/Middle Frontal Gyrus BA10
- Mid-Cingulate BA31
- Superior Temporal Gyrus
- Cerebellum

p < 0.001 uncorr.

z-value

3.1

5.8
Plan

1. Language modelling
2. Generative neural parsing
3. Variable beam search
4. Unsupervised perspectives
Observation

Data set is small

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

<table>
<thead>
<tr>
<th>Model</th>
<th>F-score</th>
<th>PPL (wsj)</th>
<th>PPL (prince)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K=50000</td>
<td>91.02</td>
<td>94.35</td>
<td>154.93</td>
</tr>
<tr>
<td>IKN5</td>
<td>-</td>
<td>155.02</td>
<td>309.54</td>
</tr>
<tr>
<td>LSTM-LM</td>
<td>-</td>
<td>141.28</td>
<td>204.06</td>
</tr>
</tbody>
</table>

- But are pretty bad in the language modelling world if we consider very large scale language models (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 computationally 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)
The idea is to rely on the following encoding of a binary tree (Rozenkrantz and Lewis 1970, Kitaev et al. 2019):

```
  4
 / \
2   6
 /   /
1 f1 3 f2 5 f3 7 f4 9
```

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:

  ![Diagram of Stack LSTM](image)

**Expected advantage**

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