CS388: Natural Language Processing

Lecture 15: Semantics II / Seq2seq I

Greg Durrett
Project 2 out today

Mini 2 graded by tomorrow

Final project feedback soon
Recall: Parses to Logical Forms

\[ \text{sings}(e470) \land \text{dances}(e470) \]

\[ \text{S} \]

\[ \text{e470} \]

\[ \text{NP} \]

\[ \text{NNP} \text{ Lady Gaga} \text{ NNP} \]

\[ \text{VP} \]

\[ \lambda y. \text{sings}(y) \land \text{dances}(y) \]

\[ \text{VP:} \lambda y. \text{a}(y) \lor \text{b}(y) \rightarrow \text{VP:} \lambda y. \text{a}(y) \text{ CC VP:} \lambda y. \text{b}(y) \]

\[ \text{S:} f(x) \rightarrow \text{NP:} x \text{ VP:} f \]
Recall: CCG

- Steedman+Szabolcsi 1980s: formalism bridging syntax and semantics
- Syntactic categories (for this lecture): S, NP, “slash” categories
  - S/\NP: “if I combine with an NP on my left side, I form a sentence” — verb
  - (S/\NP)/NP: “I need an NP on my right and then on my left” — verb with a direct object
This Lecture

- Seq2seq models
- Seq2seq models for semantic parsing
- Intro to attention
Encoder-Decoder Models
Motivation

- Parsers have been pretty hard to build...
  - Constituency/graph-based: complex dynamic programs
  - Transition-based: complex transition systems
  - CCG/semantic parsers: complex syntax/semantics interface, challenging inference, challenging learning
- For semantic parsing in particular: bridging the syntax-semantics divide results in structural weirdnesses in parsers, hard to learn the right semantic grammar
- Encoder-decoder models can (in principle) predict any linearized sequence of tokens
Encoder-Decoder

- Semantic parsing:

  \[ \text{What states border Texas} \rightarrow \lambda x \; \text{state}(x) \land \text{borders}(x, e89) \]

- Syntactic parsing

  \[ \text{The dog ran} \rightarrow (S \; (NP \; (DT \; the) \; (NN \; dog)) \; (VP \; (VBD \; ran)) \; ) \; ) \]

  (but what if we produce an invalid tree or one with different words?)

- Machine translation, summarization, dialogue can all be viewed in this framework as well
Encoder-Decoder

- Encode a sequence into a fixed-sized vector

- Now use that vector to produce a series of tokens as output from a separate LSTM decoder

Sutskever et al. (2014)
It’s not an ACL tutorial on vector representations of meaning if there’s at least one Ray Mooney quote.

In the words of Ray Mooney...

“You can’t cram the meaning of a whole %&&!$ing sentence into a single $&&!*ing vector!”

Yes, the censored-out swearing is copied verbatim.

- Is this true? Sort of... we’ll come back to this later
Model

- Generate next word conditioned on previous word as well as hidden state

- W size is $|\text{vocab}| \times |\text{hidden state}|$, softmax over entire vocabulary

$$P(y_i | x, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h})$$

$$P(y | x) = \prod_{i=1}^{n} P(y_i | x, y_1, \ldots, y_{i-1})$$

Decoder has separate parameters from encoder, so this can learn to be a language model (produce a plausible next word given current one)
Inference

- Generate next word conditioned on previous word as well as hidden state

  - Movie was great

- During inference: need to compute the argmax over the word predictions and then feed that to the next RNN state

- Need to actually evaluate computation graph up to this point to form input for the next state

- Decoder is advanced one state at a time until [STOP] is reached
Implementing seq2seq Models

Encoder: consumes sequence of tokens, produces a vector. Analogous to encoders for classification/tagging tasks.

Decoder: separate module, single cell. Takes two inputs: hidden state (vector $h$ or tuple $(h, c)$) and previous token. Outputs token + new state.
Objective: maximize
$$\sum_{(x,y)} \sum_{i=1}^{n} \log P(y_i^*|x, y_1^*, \ldots, y_{i-1}^*)$$

One loss term for each target-sentence word, feed the correct word regardless of model’s prediction (called “teacher forcing”)
Training: Scheduled Sampling

- Model needs to do the right thing even with its own predictions

- Scheduled sampling: with probability $p$, take the gold as input, else take the model’s prediction
- Starting with $p = 1$ (teacher forcing) and decaying it works best
- “Right” thing: train with reinforcement learning

Bengio et al. (2015)
Implementation Details

- Sentence lengths vary for both encoder and decoder:
  - Typically pad everything to the right length and use a mask or indexing to access a subset of terms

- Encoder: looks like what you did in Mini 2

- Decoder: execute one step of computation at a time, so computation graph is formulated as taking one input + hidden state
  - Test time: do this until you generate the stop token
  - Training: do this until you reach the gold stopping point
Batching is pretty tricky: decoder is across time steps, so you probably want your label vectors to look like [num timesteps x batch size x num labels], iterate upwards by time steps.

Beam search: can help with lookahead. Finds the (approximate) highest scoring sequence:

$$\argmax_y \prod_{i=1}^{n} P(y_i | x, y_1, \ldots, y_{i-1})$$
Beam Search

- Maintain decoder state, token history in beam
- Keep both *film* states! Hidden state vectors are different

```
<s>
the movie was great
```

```
film: 0.4
log(0.4)
```

```
le: 0.3

les: 0.1

log(0.1)

log(0.3)+log(0.8)
```

```
film: 0.8
log(0.4)
```

```
la: 0.4

le: 0.3

les: 0.1

log(0.4)+log(0.4)
```

```
le

film

log(0.3)+log(0.8)
```

```
la

film

log(0.4)
```

```
film

log(0.4)
```
Other Architectures

- What’s the basic abstraction here?
  - Encoder: sentence -> vector
  - Decoder: hidden state, output prefix -> new hidden state, new output
    - OR: sentence, output prefix -> new output (more general)
- Wide variety of models can apply here: CNN encoders, decoders can be any autoregressive model including certain types of CNNs
- Transformer: another model discussed next lecture
Seq2seq Semantic Parsing
Semantic Parsing as Translation

“what states border Texas”

\[
\lambda x \ ( \ \text{state}(x) \ \text{and} \ \text{border}(x, e89))
\]

- Write down a linearized form of the semantic parse, train seq2seq models to directly translate into this representation.

- What are some benefits of this approach compared to grammar-based?

- What might be some concerns about this approach? How do we mitigate them?

Jia and Liang (2016)
Handling Invariances

“what states border Texas”  “what states border Ohio”

- Parsing-based approaches handle these the same way
  - Possible divergences: features, different weights in the lexicon
- Can we get seq2seq semantic parsers to handle these the same way?
- Key idea: don’t change the model, change the data
- “Data augmentation”: encode invariances by automatically generating new training examples
Data Augmentation

Examples
(“what states border texas ?”,
answer(NV, (state(V0), next_to(V0, NV), const(V0, stateid(texas)))))

Rules created by ABSENTITIES
ROOT → {“what states border STATEId ?”,
   answer(NV, (state(V0), next_to(V0, NV), const(V0, stateid(STATEId))})
STATEId → {“texas”,texas}
STATEId → {“ohio”,ohio}

- Lets us synthesize a “what states border ohio ?” example
- Abstract out entities: now we can “remix” examples and encode invariance to entity ID. More complicated remixes too

Jia and Liang (2016)
Semantic Parsing as Translation

Geo
$x$: “what is the population of iowa?”
y: answer ( NV , ( _population ( NV , V1 ) , _const ( V0 , _stateid ( iowa ) ) ) )

ATIS
$x$: “can you list all flights from chicago to milwaukee”
y: ( _lambda $0 e ( _and ( _flight $0 ) ( _from $0 chicago : _ci ) ( _to $0 milwaukee : _ci ) ) )

Overnight
$x$: “when is the weekly standup”
y: ( call listValue ( call getPropertyValue meeting.weekly_standup ( string start_time ) ) )

- Prolog
- Lambda calculus
- Other DSLs

Handle all of these with uniform machinery!

Jia and Liang (2016)
Three forms of data augmentation all help

Results on these tasks are still not as strong as hand-tuned systems from 10 years ago, but the same simple model can do well at all problems

<table>
<thead>
<tr>
<th>Previous Work</th>
<th>GEO</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zettlemoyer and Collins (2007)</td>
<td></td>
<td>84.6</td>
</tr>
<tr>
<td>Kwiatkowski et al. (2010)</td>
<td>88.9</td>
<td></td>
</tr>
<tr>
<td>Liang et al. (2011)</td>
<td>91.1</td>
<td></td>
</tr>
<tr>
<td>Kwiatkowski et al. (2011)</td>
<td>88.6</td>
<td>82.8</td>
</tr>
<tr>
<td>Poon (2013)</td>
<td>83.5</td>
<td></td>
</tr>
<tr>
<td>Zhao and Huang (2015)</td>
<td>88.9</td>
<td>84.2</td>
</tr>
<tr>
<td><strong>Our Model</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No Recombination</td>
<td>85.0</td>
<td>76.3</td>
</tr>
<tr>
<td><strong>ABS</strong></td>
<td>85.4</td>
<td>79.9</td>
</tr>
<tr>
<td><strong>ABS</strong>WholePhrases</td>
<td>87.5</td>
<td></td>
</tr>
<tr>
<td>CONCAT-2</td>
<td>84.6</td>
<td>79.0</td>
</tr>
<tr>
<td>CONCAT-3</td>
<td></td>
<td>77.5</td>
</tr>
<tr>
<td>AWP + AE</td>
<td>88.9</td>
<td></td>
</tr>
<tr>
<td>AE + C2</td>
<td></td>
<td>78.8</td>
</tr>
<tr>
<td>AWP + AE + C2</td>
<td><strong>89.3</strong></td>
<td>83.3</td>
</tr>
<tr>
<td>AE + C3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Jia and Liang (2016)
Regex Prediction

- Predict regex from text

Problem: requires a lot of data: 10,000 examples needed to get ~60% accuracy on pretty simple regexes

Does not scale when regex specifications are more abstract (*I want to recognize a decimal number less than 20*)

Locascio et al. (2016)
SQL Generation

- Convert natural language description into a SQL query against some DB

- How to ensure that well-formed SQL is generated?
  - Three seq2seq models

- How to capture column names + constants?
  - Pointer mechanisms, to be discussed later

**Question:** How many CFL teams are from York College?

**SQL:**

```
SELECT COUNT CFL Team FROM CFLDraft WHERE College = "York"
```

Zhong et al. (2017)
Orange pieces are probably reused across many problems.

Not too hard to learn to generate: start with lambda, always follow with x, follow that with paren, etc.

LSTM has to remember the value of Texas for 13 steps!

Next: attention mechanisms that let us “look back” at the input to avoid having to remember everything.
Attention
Problems with Seq2seq Models

- Encoder-decoder models like to repeat themselves:

  Un garçon joue dans la neige → A boy plays in the snow \textit{boy plays boy plays}

- Why does this happen?
  - Models trained poorly
    - Input is forgotten by the LSTM so it gets stuck in a “loop” of generating the same output tokens again and again
  - Need some notion of input coverage or what input words we’ve translated
Problems with Seq2seq Models

- Bad at long sentences: 1) a fixed-size hidden representation doesn’t scale; 2) LSTMs still have a hard time remembering for really long periods of time.

RNNenc: the model we’ve discussed so far.
RNNsearch: uses attention.

Bahdanau et al. (2014)
Problems with Seq2seq Models

- Unknown words:

  *en*: The *écotax* portico in *Pont-de-Buis*, ... [truncated] ..., was taken down on Thursday morning

  *fr*: Le *portique écotaxe* de *Pont-de-Buis*, ... [truncated] ..., a été *démonté* jeudi matin

  *nn*: Le *unk* de *unk* à *unk*, ... [truncated] ..., a été pris le jeudi matin

- Encoding these rare words into a vector space is really hard

- In fact, we don’t want to encode them, we want a way of directly looking back at the input and copying them (Pont-de-Buis)
Suppose we knew the source and target would be word-by-word translated.

In that case, we could look at the corresponding input word when translating — might improve handling of long sentences!

How can we achieve this without hardcoding it?
At each decoder state, compute a distribution over source inputs based on current decoder state.

Use the weighted sum of input tokens to predict output.
Takeaways

- Rather than combining syntax and semantics like in CCG, we can either parse to semantic representations directly or generate them with seq2seq models.

- Seq2seq models are a very flexible framework, some weaknesses can potentially be patched with more data.

- How to fix their shortcomings? Next time: attention, copying, and transformers.