Graham Neubig (CMU) talk this Friday at 11am in 6.302. “Towards Open-domain Generation of Programs from Natural Language”

Mini 2 graded by the end of the week

Project 2 out by Thursday
Recall: Pareses to Logical Forms

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

\[ S \]

\[ e470 \quad \text{NP} \quad \lambda y. \text{sings}(y) \land \text{dances}(y) \]

\[ \text{NNP} \quad \text{NNP} \]

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

\[ \text{VP} \quad \text{CC} \quad \text{VP} \]

\[ \text{VBP} \quad \text{VBP} \]

\[ \text{sings} \quad \text{dances} \]

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

- General rules:
  - VP: \( \lambda y. a(y) \land b(y) \) -> VP: \( \lambda y. a(y) \) CC VP: \( \lambda y. b(y) \)
  - S: \( f(x) \) -> NP: \( x \) 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

```
S
sings(e728)

NP
e728

S\NP
λy. sings(y)

NP
λy. borders(y, e98)

(S\NP)/NP
λx. λy. borders(y, x)

NP
e98

S
borders(e98, e101)
```

- Eminem sings e728
- Oklahoma borders e98 e101
- Texas
This Lecture

- Lambda-DCS: more lightweight than CCG
- Seq2seq models
- Seq2seq models for semantic parsing
Lambda-DCS

- Dependency-based compositional semantics — original version was less powerful than lambda calculus, lambda-DCS is as powerful.

- Designed in the context of building a QA system from Freebase.

- Freebase: set of entities and relations.

- `[PlaceOfBirth]` = set of pairs of (person, location).

Liang et al. (2011), Liang (2013)
Lambda-DCS

Lambda-DCS
Seattle
PlaceOfBirth
PlaceOfBirth.Seattle

Lambda calculus

\( \lambda x. x = \text{Seattle} \)
\( \lambda x. \lambda y. \text{PlaceOfBirth}(x,y) \)
\( \lambda x. \text{PlaceOfBirth}(x,\text{Seattle}) \)

- Looks like a tree fragment over Freebase, denotes the set of people born in Seattle, no explicit variables

\( \land \) PlaceOfBirth - Seattle

\( \land \) Profession.Scientist
\( \land \) PlaceOfBirth.Seattle

\( \lambda x. \text{Profession}(x,\text{Scientist}) \)
\( \lambda x. \text{PlaceOfBirth}(x,\text{Seattle}) \)

Liang et al. (2011), Liang (2013)
Lambda-DCS

March 15, 1961

Alice Smith

DateOfBirth

PlaceOfBirth

Scientist

Profession

Bob Cooper

PlaceOfBirth

PlaceOfBirth

Seattle

PlaceOfBirth

PlaceOfBirth

Washington

PlaceOfBirth

CapitalOf

“list of scientists born in Seattle”

Profession.Scientist ^

PlaceOfBirth.Seattle

Execute this fragment against Freebase, returns Alice Smith (and others)

Liang et al. (2011), Liang (2013)
Parsing into Lambda-DCS

- Derivation $d$ on sentence $x$:
  
  
- No more explicit syntax in these derivations like we had in CCG

- Everything is a set, sets combine in a few ways

- Building the lexicon: more sophisticated process than GENLEX, but can handle thousands of predicates

- Log-linear model with features on rules: $P(d|x) \propto \exp w^\top \left( \sum_{r \in d} f(r, x) \right)$

  Berant et al. (2013)
Learn from derivations: standard supervised learning, maximize probability of correct derivation

\[ \mathcal{L}(\theta) = \sum_{i=1}^{n} \log P(d_i^* | x_i) \]

Problem: supervision looks like “Where was Barack Obama born” — “Hawaii” without a derivation
Learn just from question-answer pairs: maximize the likelihood of the right denotation $y^*$ with the derivation $d$ marginalized out

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \log \sum_{d: [[d]]_K = y^*_i} P(d|x_i)$$

sum over derivations $d$ such that the denotation of $d$ on knowledge base $K$ is $y_i$

Approx procedure: for each example:

1. Run beam search to get a set of derivations
2. Let $d =$ highest-scoring derivation in the beam
3. Let $d^* =$ highest-scoring derivation in the beam with correct denotation
4. Do a structured perceptron update towards $d^*$ away from $d$
Learning

- Each vertical slice is the beam for one example. Green = correct denotation

- Only a small number of questions are even reachable by beam search initially (but some questions are very easy so even a totally untrained model can answer them)

- During training, more and more “good” derivations surface and will result in model updates

Berant et al. (2013)
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
  - Encoder-decoder models can be a lot more uniform — we’ll come back to this later in the lecture
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 the least one Ray Mooney quote.

- 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.

- 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
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 $p = 1$ and decaying it works best

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. Can be a CNN/LSTM/...

- Decoder: also flexible in terms of architecture (more next lecture). 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
Implementa9on Details (cont’d)

- Batching is pretty tricky
  - Decoder is across time steps, so you probably want your label vectors to look like \([\text{num timesteps} \times \text{batch size} \times \text{num labels}]\), iterate upwards by time steps
- Beam search: can help with lookahead. Finds the (approximate) highest scoring sequence:

\[
\arg\max_y \prod_{i=1}^{n} P(y_i | x, y_1, \ldots, y_{i-1})
\]
Beam Search

- Maintain decoder state, token history in beam

- Do not max over the two film states! Hidden state vectors are different
Seq2seq Semantic Parsing
Semantic Parsing as Translation

“*what states border Texas*”

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

- 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 (2015)
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

Jia and Liang (2015)

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

(“what is the highest mountain in ohio ?”,
answer(NV, highest(V0, (mountain(V0), loc(V0, NV), const(V0, stateid(ohio)))))

Rules created by ABSENTITIES
ROOT → {“what states border STATEID ?”,
   answer(NV, (state(V0), next_to(V0, NV), const(V0, stateid(STATEID ))))}
STATEID → {“texas”, “texas”}
ROOT → {“what is the highest mountain in STATEID ?”,
   answer(NV, highest(V0, (mountain(V0), loc(V0, NV),
   const(V0, stateid(STATEID )))))}
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
Semantic Parsing as Translation

- Prolog
- Lambda calculus
- Other DSLs

Handle all of these with uniform machinery!

Jia and Liang (2015)
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.

Jia and Liang (2015)
Regex Prediction

- Can use for other semantic parsing-like tasks
- Predict regex from text

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

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

Question:
How many CFL teams are from York College?

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

“what states border Texas” → lambda x ( state ( x ) and border ( x , e89 ) )

- 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. This is a common question
- LSTM has to remember the value of Texas for 13 steps!
- Next lecture: attention mechanisms that let us “look back” at the input to avoid having to remember everything
Takeaways

- Lambda-DCS is a more lightweight formalism than lambda calculus.

- 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.