CSE 5525: Foundations of Speech and Language Processing

Sequence to sequence (seq2seq)

Huan Sun (CSE@OSU)

Many thanks to Prof. Greg Durrett @ UT Austin for sharing his slides.
This Lecture

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

- Semantic parsing:

  What states border Texas → \( \lambda x \text{ state}(x) \land \text{borders}(x, e89) \)

- Syntactic parsing

  The dog ran → (S (NP (DT the) (NN dog)) (VP (VBD ran)))

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

- Machine translation (e.g., English sentence as input and French as output), 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)
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.
Training

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”)

the movie was great

le film était bon
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: Check out HW3 encoder

- 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

Offline reading & practice in HW3
Implementa.on Details (cont’d)

‣ 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:

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

- Maintain decoder state, token history in beam

- Keep both *film* states! Hidden state vectors are different

The movie was great
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: Convolutional Neural Networks (CNN) encoders, decoders can be any autoregressive model including certain types of CNNs

- Transformer: another widely used model recently developed: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

  Offline reading & practice in HW3
Seq2seq Semantic Parsing
**Semantic Parsing as Translation**

“What states border Texas”

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

- 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 (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

- 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.
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"
```
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
Takeaways

‣ How encoder-decoder models work

‣ Seq2seq models are a very flexible framework for various problems. Some weaknesses can potentially be patched with more data

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