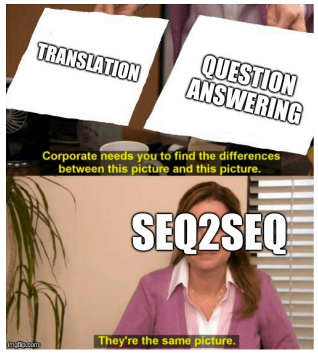
CS388: Natural Language Processing

Lecture 14: Seq2seq II, Attention

Greg Durrett

TEXAS
The University of Texas at Austin



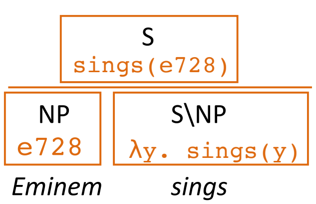
credit: NawaphonIsarathanachaikul on imgflip

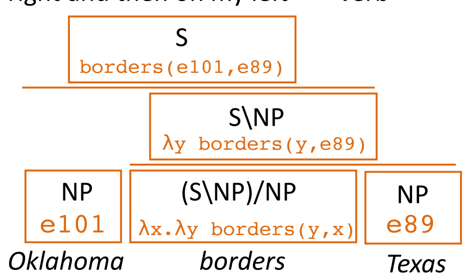
Administrivia

- Project 2 out today
- Mini 2 graded soon
- Final project proposals due tonight (or Friday if you want)

Recall: CCG

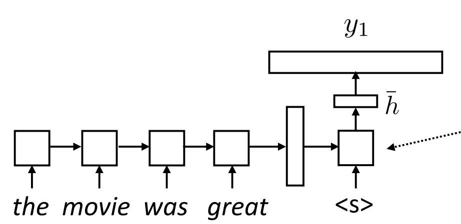
- Steedman+Szabolcsi 1980s: formalism bridging syntax and semantics
- Syntactic categories: S, NP, "slash" categories
  - S\NP: "if I combine with an NP on my left side, I form a sentence" — verb
  - (S\NP)/NP: "I need an NP on my right and then on my left" — verb with a direct object





Recall: Seq2seq Models

- Generate next word conditioned on previous word as well as hidden state
- W size is |vocab| x |hidden state|, softmax over entire vocabulary

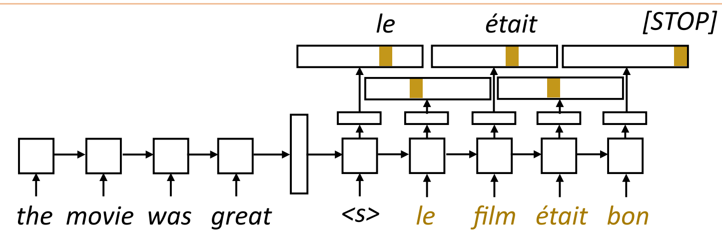


$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \text{softmax}(W\bar{h})$$

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

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

Recall: Training Seq2seq Models



- Objective: maximize $\sum_{(\mathbf{x},\mathbf{y})} \sum_{i=1}^{n} \log P(y_i^*|\mathbf{x}, y_1^*, \ldots, y_{i-1}^*)$
- One loss term for each target-sentence word, feed the correct word regardless of model's prediction (called "teacher forcing")

This Lecture

- Seq2seq implementation (continued)
- Seq2seq models for semantic parsing
- Attention motivation
- Attention definitions, math, mechanics

Implementing Encoder-Decoder Models

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:
  - Test time: execute one step of computation at a time, so computation graph is formulated as taking one input + hidden state
  - Training: you can execute all timesteps as part of one computation graph

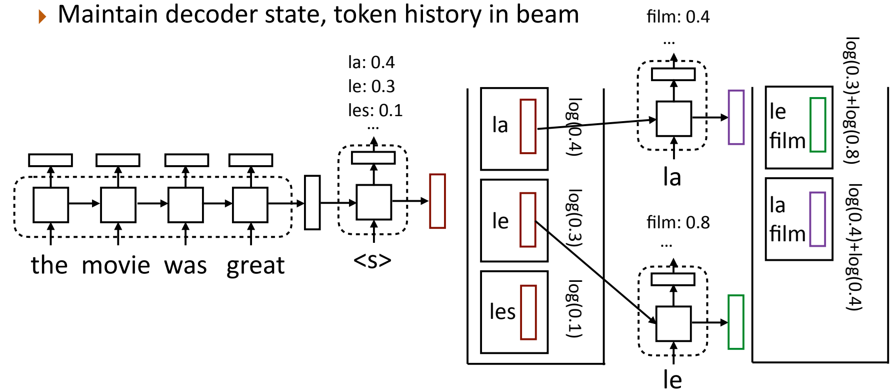
Implementation Details (cont'd)

- Batching is a bit tricky: encoder should use pack_padded_sequence to handle different lengths. The decoder should pad everything to the same length and use a mask to only accumulate "valid" loss terms
- Beam search: can help with lookahead. Finds the (approximate) highest scoring sequence:

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

Beam Search

- Maintain decoder state, token history in beam



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

Seq2seq Semantic Parsing

Semantic Parsing as Translation

"what states border Texas"

↓

```
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

Jia and Liang (2016)

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

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
    getProperty meeting.weekly_standup
    ( string start_time ) ) )
```

- Prolog
- Lambda calculus
- Other DSLs
- Handle all of these with uniform machinery!

Jia and Liang (2016)

Semantic Parsing as Translation

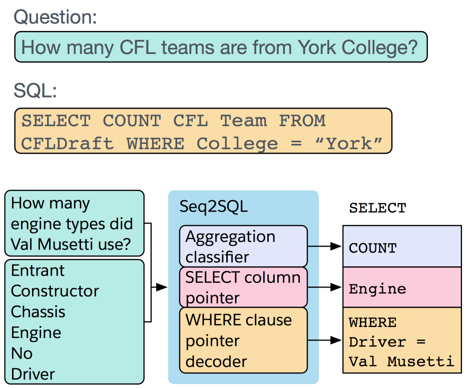
	Geo	ATIS
Previous Work		
Zettlemoyer and Collins (2007)		**84.6**
Kwiatkowski et al. (2010)	88.9	
Liang et al. (2011)[2]	91.1	
Kwiatkowski et al. (2011)	88.6	82.8
Poon (2013)		83.5
Zhao and Huang (2015)	88.9	84.2
Our Model		
No Recombination	85.0	76.3
AbsEntities	85.4	79.9
AbsWholePhrases	87.5	
Concat-2	84.6	79.0
Concat-3		77.5
AWP + AE	88.9	
AE + C2		78.8
AWP + AE + C2	**89.3**	
AE + C3		83.3

- 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 (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



Zhong et al. (2017)

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.
- 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

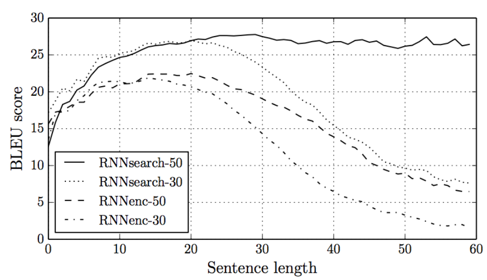
- Encoder-decoder models like to repeat themselves:

Un garçon joue dans la neige → A boy plays in the snow **boy plays boy plays**

- Why does this happen?
  - Models trained poorly (undertrained / haven't converged)
  - 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 *ecotax* portico in *Pont-de-Buis* , . . . [truncated] . . . , was taken down on Thursday morning

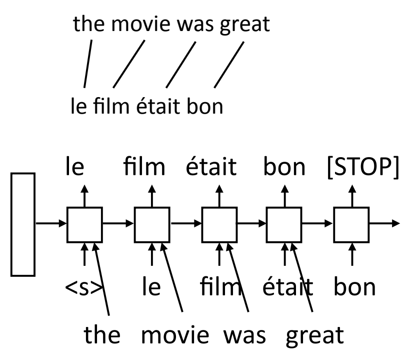
fr: Le *portique* *écotaxe* de *Pont-de-Buis* , . . . [truncated] . . . , a été *démonté* jeudi matin

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

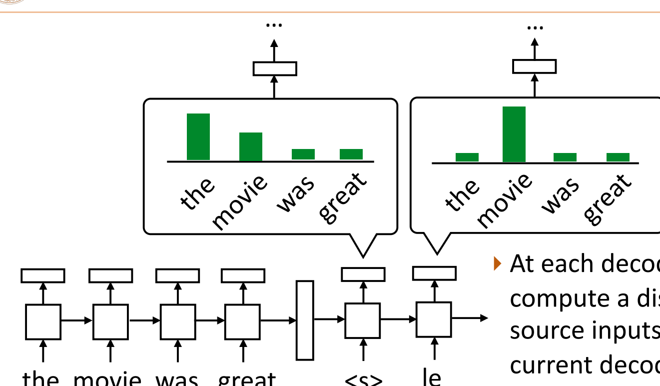
- Encoding these rare words into a vector space is really hard
- In fact, we don't want to encode them, we want a way of directly looking back at the input and copying them (Pont-de-Buis)

Aligned Inputs

- 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?



Attention

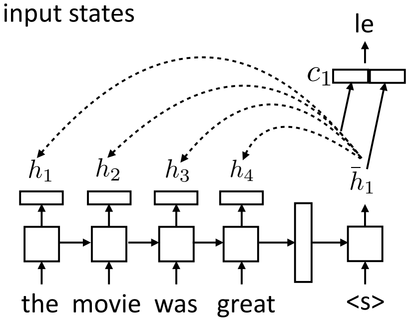


- At each decoder state, compute a distribution over source inputs based on current decoder state
- Use the weighted sum of input tokens to predict output

Attention Mechanism

Attention

- For each decoder state, compute weighted sum of input states



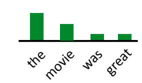
- No attn: $P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W\bar{h}_i)$

$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W[c_i; \bar{h}_i])$$

$$c_i = \sum_j \alpha_{ij} h_j$$

- Weighted sum of input hidden states (vector)

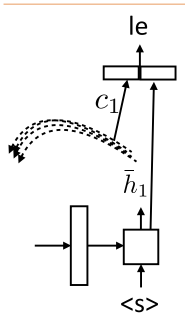
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$



$$e_{ij} = f(\bar{h}_i, h_j)$$

- Some function f (TBD)

Attention



$$c_i = \sum_j \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

$$e_{ij} = f(\bar{h}_i, h_j)$$

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

- Bahdanau+ (2014): additive

$$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$$

- Luong+ (2015): dot product

$$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$$

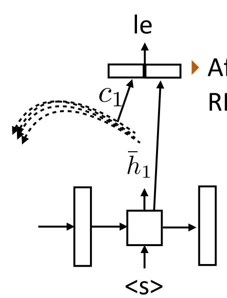
- Luong+ (2015): bilinear

- Note that this all uses outputs of hidden layers

Luong et al. (2015)

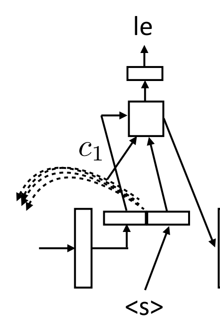
Alternatives

- When do we compute attention? Can compute before or after RNN cell



- After RNN cell

Luong et al. (2015)

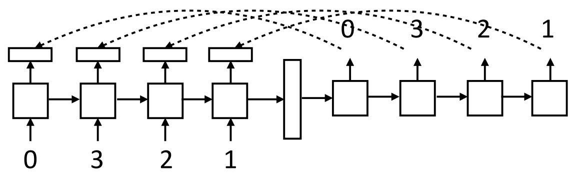


- Before RNN cell; this one is a little more convoluted and less standard

Bahdanau et al. (2015)

What can attention do?

- Learning to copy — how might this work?

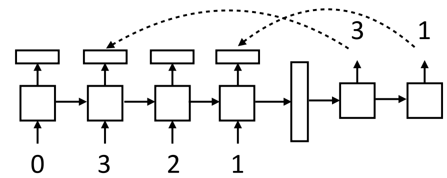


- LSTM can learn to count with the right weight matrix
- This is a kind of position-based addressing

Luong et al. (2015)

What can attention do?

- Learning to subsample tokens

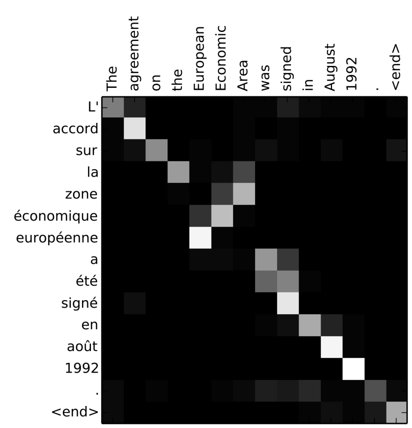


- Need to count (for ordering) and also determine which tokens are in/out
- Content-based addressing

Luong et al. (2015)

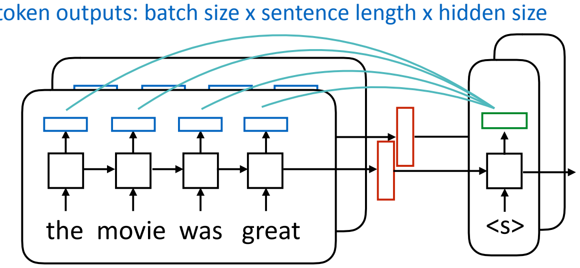
Attention

- Encoder hidden states capture contextual source word identity
- Decoder hidden states are now mostly responsible for selecting what to attend to
- Doesn't take a complex hidden state to walk monotonically through a sentence and spit out word-by-word translations



Batching Attention

token outputs: batch size x sentence length x hidden size



hidden state: batch size x hidden size

$$e_{ij} = f(\bar{h}_i, h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

sentence outputs: batch size x hidden size

attention scores = batch size x sentence length

c = batch size x hidden size $\quad c_i = \sum_j \alpha_{ij} h_j$

- Make sure tensors are the right size!

Luong et al. (2015)

Results

- Machine translation: BLEU score of 14.0 on English-German -> 16.8 with attention, 19.0 with smarter attention (we'll come back to this later)
- Summarization/headline generation: bigram recall from 11% -> 15%
- Semantic parsing: ~30-50% accuracy -> 70+% accuracy on Geoquery

Luong et al. (2015)
Chopra et al. (2016)
Jia and Liang (2016)

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