CS388: Natural Language Processing
Lecture 15: Attention

Greg Durrett

This Lecture

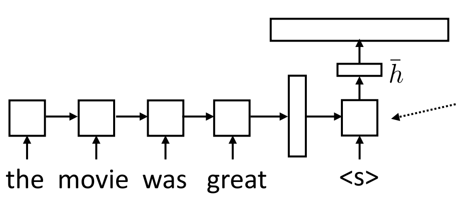
- Graham Neubig (CMU) talk this Friday at 11am in 6.302.
  “Towards Open-domain Generation of Programs from Natural Language”
- Project 2 out by the end of today; due *Friday* November 2
- Mini 2 graded by this weekend

Recall: Seq2seq Model

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

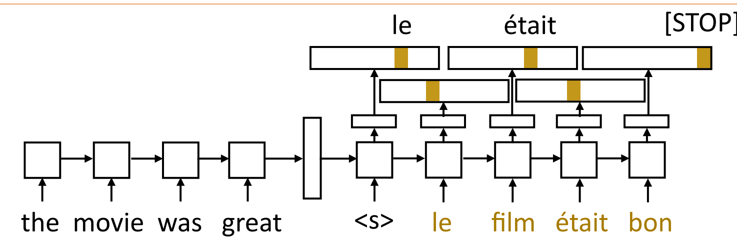
$$P(y_i|\mathbf{x}, y_1, \ldots, y_{i-1}) = \mathrm{softmax}(W\bar{h})$$

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



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

Recall: Seq2seq Training



- 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

Recall: Semantic Parsing as Translation

"what states border Texas"

↓

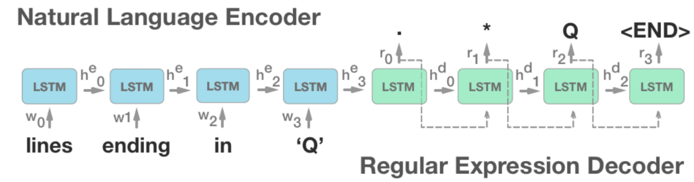
`lambda x ( state ( x ) and border ( x , e89 ) ) )`

- Write down a linearized form of the semantic parse, train seq2seq models to directly translate into this representation
- No need to have an explicit grammar, simplifies algorithms
- Might not produce well-formed logical forms, might require lots of data

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

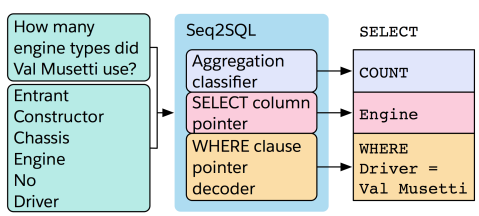
- Convert natural language description into a SQL query against some DB
- How to ensure that well-formed SQL is generated?
  - Three seq2seq models
- How to capture column names + constants?
  - Pointer mechanisms

Question:

How many CFL teams are from York College?

SQL:

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



Zhong et al. (2017)

This Lecture

- Attention
- Copying
- Transformers

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**

- Often a byproduct of training these models poorly
- Need some notion of input coverage or what input words we've translated

Problems with Seq2seq Models

- Unknown words:

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

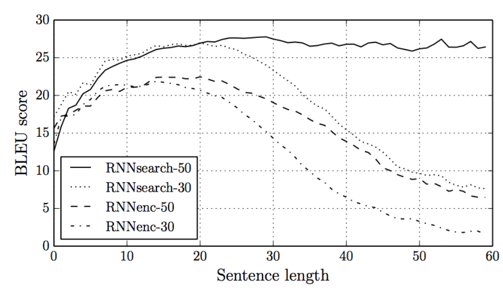
fr: Le *portique* *écotaxe* de *Pont-de-Buis* , . . . [truncated] . . . , a été *démonté* jeudi matin

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

- No matter how much data you have, you'll need some mechanism to copy a word like Pont-de-Buis from the source to target

Problems with Seq2seq Models

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

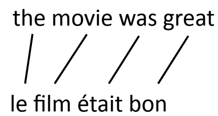


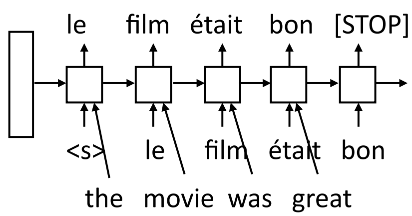
RNNsearch: introduces attention mechanism to give "variable-sized" representation

Bahdanau et al. (2014)

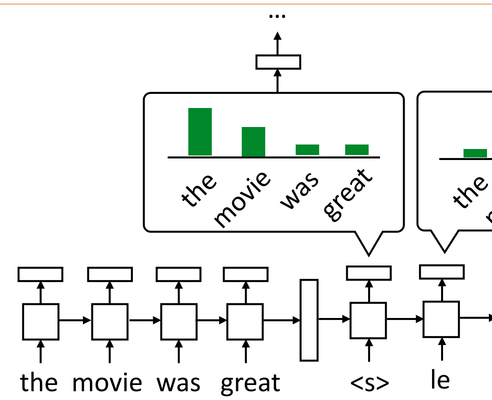
Aligned Inputs

- Suppose we knew the source and target would be word-by-word translated
- Can look at the corresponding input word when translating — this could scale!
- Much less burden on the hidden state
- How can we achieve this without hardcoding it?





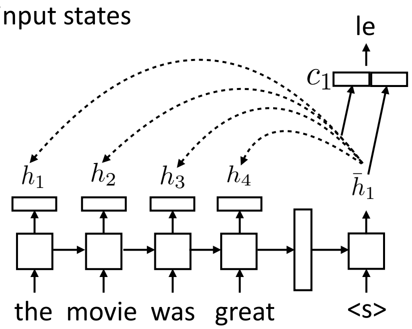
Attention



- At each decoder state, compute a distribution over source inputs based on current decoder state
- Use that in output layer

Attention

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



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

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

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

- Weighted sum of input hidden states (vector)

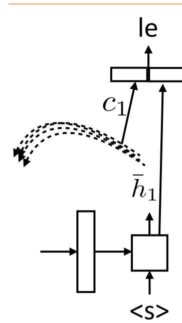
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

- Normalized scalar weight

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

- Unnormalized scalar weight

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

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

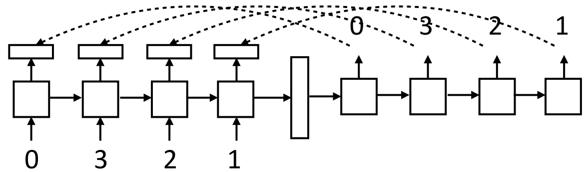
- Luong+ (2015): bilinear

- Note that this all uses outputs of hidden layers

Luong et al. (2015)

What can attention do?

- Learning to copy — how might this work?

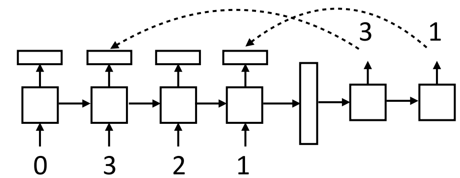


- LSTM can learn to count with the right weight matrix
- This is effectively position-based addressing

Luong et al. (2015)

What can attention do?

- Learning to subsample tokens

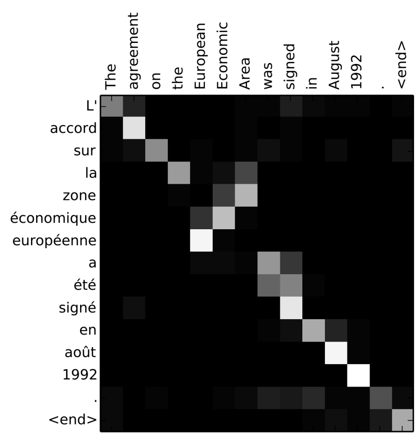


- Need to count (for ordering) and also determine which tokens are in/out
- Content-based addressing

Luong et al. (2015)

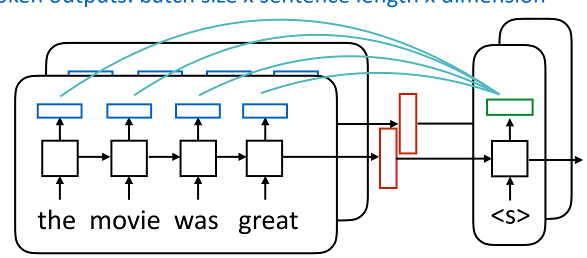
Attention

- 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 dimension



hidden state: batch size x dimension

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

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

sentence outputs: batch size x hidden size

attention scores = batch size x sentence length

c = batch size x hidden size

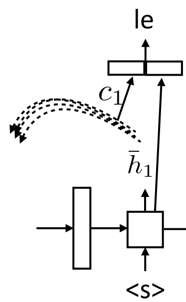
$$c_i = \sum_j \alpha_{ij} h_j$$

- Make sure tensors are the right size!

Luong et al. (2015)

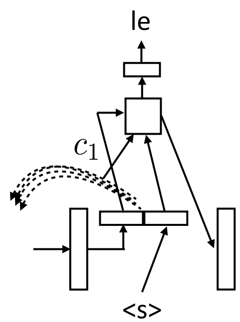
Alternatives

- 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)

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% accuracy -> 70+% accuracy on Geoquery

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

Copying Input/Pointers

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

- Want to be able to copy named entities like Pont-de-Buis

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

from attention

from RNN hidden state

- Still can only generate from the vocabulary

Jean et al. (2015), Luong et al. (2015)

Copying

en: The <u>*ecotax*</u> portico in <u>*Pont-de-Buis*</u> , . . . [truncated] . .

fr: Le <u>*portique*</u> <u>*écotaxe*</u> de <u>*Pont-de-Buis*</u> , . . . [truncated] .

nn: Le <u>unk</u> de <u>unk</u> à <u>unk</u> , . . . [truncated] . . . , a été pris

$\left\{\begin{array}{l}\text{the}\\ \text{a}\\ \dots\\ \text{zebra}\\ \hline \text{Pont-de-Buis}\\ \text{ecotax}\end{array}\right\}$

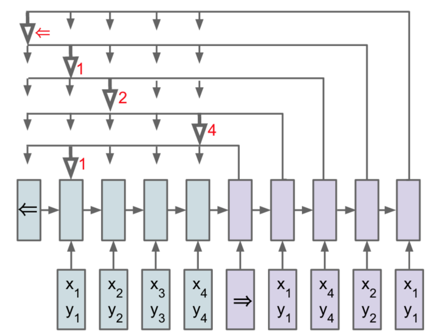
- Vocabulary contains "normal" vocab as well as words in input. Normalizes over both of these:

$$P(y_i = w|\mathbf{x}, y_1, \ldots, y_{i-1}) \propto \begin{cases} \exp W_w[c_i; \bar{h}_i] & \text{if } w \text{ in vocab} \\ h_j^\top V \bar{h}_i & \text{if } w = x_j \end{cases}$$

- Bilinear function of input representation + output hidden state

Pointer Networks

- Only point to the input, don't have any notion of vocabulary
- Used for tasks including summarization and sentence ordering



Vinyals et al. (2015)

Results

	Geo	ATIS
No Copying	74.6	69.9
With Copying	85.0	76.3

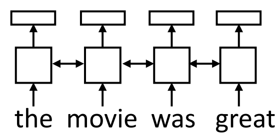
- For semantic parsing, copying tokens from the input (texas) can be very useful
- In many settings, attention can roughly do the same things as copying

Jia and Liang (2016)

Transformers

Self-Attention

- LSTM abstraction: maps each vector in a sentence to a new, context-aware vector
- CNNs did something similar with filters
- Attention can give us a third way to do this



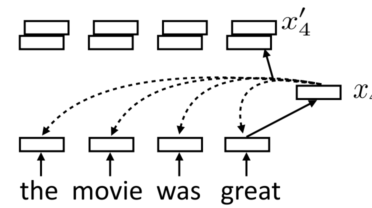
Vaswani et al. (2017)

Self-Attention

- Each word forms a "query" which then computes attention over each word

$$\alpha_{i,j} = \mathrm{softmax}(x_i^\top x_j)$$ scalar

$$x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j$$ vector = sum of scalar * vector



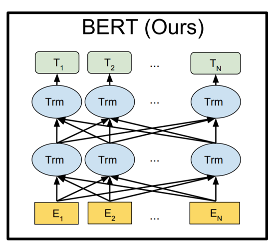
- Multiple "heads" analogous to different convolutional filters. Use parameters W_k and V_k to get different attention values + transform vectors

$$\alpha_{k,i,j} = \mathrm{softmax}(x_i^\top W_k x_j) \qquad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j$$

Vaswani et al. (2017)

Deep Transformers

- Supervised: transformer can replace LSTM; will revisit this when we discuss MT
- Unsupervised: transformers work better than LSTM for unsupervised pre-training of embeddings: predict word given context words
- Devlin et al. October 11, 2018
  "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"
- Stronger than similar methods, SOTA on ~11 tasks (including NER — 92.8 F1)



Takeaways

- Attention is very helpful for seq2seq models
- Used for tasks including summarization and sentence ordering
- Explicitly copying input can be beneficial as well
- Transformers are strong models we'll come back to later

Where are we going

- We've now talked about most of the important core tools for NLP
- Rest of the class: more focused on applications
- Information extraction, then MT, then a grab bag of things