Lecture 11: Seq2Seq + Attention

Alan Ritter

(many slides from Greg Durrett)
Administrivia

Course Project

Next homework assignment
Final Project

• Groups of 3-4
• Sign up here:
  • https://forms.gle/3u5vC78uBP6eK8Xn7
• Highly Recommended: stop by the office hours to chat and get feedback on your project.
• **Scope:** on the order of one of the programming assignments
  • But, need to define the problem, come up with right feature representation, write up results in a formal report.
Selecting a Topic

• Part of your thesis? Great!
• Find a problem you are interested in where you think NLP can help.
• Experiment with one of the algorithms we discussed about in class.
• **First question:** what is the dataset?
Datasets

- Various Semeval Tasks:
- Fake News Challenge:
  - http://www.fakenewschallenge.org/
- Machine Translation:
  - http://www.statmt.org/wmt19/robustness.html
- Dialogue:
  - https://github.com/mgalley/DSTC7-End-to-End-Conversation-Modeling
- Many more…
Requirements

- 4 Page Report
- Include empirical analysis of your approach
  - Report performance on dev / test set
  - Compare against some reasonable baseline method.
- In class presentation during scheduled Final Exam time
Advice

• **First question:** is the data available?
• Try to get a simple baseline working as early as possible to determine whether your project idea is feasible.
• Start with a manageable-sized dataset
• Then scale up…
Recall: CNNs vs. LSTMs

\[ n \times k \]

the movie was good
Recall: CNNs vs. LSTMs

c filters, m x k each

the movie was good
Recall: CNNs vs. LSTMs

O(n) x c

c filters,
m x k each

n x k

the movie was good
<table>
<thead>
<tr>
<th>O(n) x c</th>
<th>n x k</th>
<th>n x k</th>
</tr>
</thead>
<tbody>
<tr>
<td>c filters, m x k each</td>
<td>the movie was good</td>
<td>the movie was good</td>
</tr>
</tbody>
</table>
Recall: CNNs vs. LSTMs

O(n) x c

c filters,
m x k each

n x k

the movie was good

BiLSTM with hidden size c

n x 2c

n x k

the movie was good
Recall: CNNs vs. LSTMs

- Both LSTMs and convolutional layers transform the input using context.
- LSTM: “globally” looks at the entire sentence (but local for many problems).
- CNN: local depending on filter width + number of layers.
Encoder-Decoder

- Encode a sequence into a fixed-sized vector

Sutskever et al. (2014)
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 not at 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

```
the movie was great
```
Model

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

the movie was great <s>
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

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

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

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

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

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

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

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

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

The movie was great
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.
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
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
Inference

- Generate next word conditioned on previous word as well as hidden state
  
  ![Diagram of RNN sequence]

  - the  movie  was  great
  - le  film  était

- 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
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
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

Decoder

Decoder

the movie was great

<\text{s}>

le

film

le

...
Implementing seq2seq Models

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

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

Bengio et al. (2015)
Training: Scheduled Sampling

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

the movie was great

la film était bon [STOP]
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

Bengio et al. (2015)
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:
Implementation Details

- Sentence lengths vary for both encoder and decoder:
  - Typically pad everything to the right length
Implementation Details

- Sentence lengths vary for both encoder and decoder:
  - Typically pad everything to the right length

- Encoder: Can be a CNN/LSTM/...
Sentence lengths vary for both encoder and decoder:
  - Typically pad everything to the right length

Encoder: Can be a CNN/LSTM/...

Decoder: also flexible in terms of architecture (more later). Execute one step of computation at a time, so computation graph is formulated as taking one input + hidden state
Implementation Details

- Sentence lengths vary for both encoder and decoder:
  - Typically pad everything to the right length

- Encoder: Can be a CNN/LSTM/...

- Decoder: also flexible in terms of architecture (more later). Execute one step of computation at a time, so computation graph is formulated as taking one input + hidden state

- 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

the movie was great

la: 0.4
le: 0.3
les: 0.1
Beam Search

- Maintain decoder state, token history in beam

the movie was great

la: 0.4
le: 0.3
les: 0.1

<s>
Beam Search

- Maintain decoder state, token history in beam

the movie was great
Beam Search

- Maintain decoder state, token history in beam

```
<s>
la: 0.4
le: 0.3
les: 0.1
...
log(0.4)
log(0.3)
log(0.1)
```

the movie was great
Beam Search

- Maintain decoder state, token history in beam

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

```
la: 0.4
le: 0.3
les: 0.1

log(0.4)
log(0.3)
log(0.1)
```
Beam Search

- Maintain decoder state, token history in beam

the movie was great

la: 0.4
le: 0.3
les: 0.1

log(0.4)
log(0.3)
log(0.1)

film: 0.4

...
Maintain decoder state, token history in beam

the movie was great

film: 0.4

la: 0.4
le: 0.3
les: 0.1

log(0.4)
log(0.3)
log(0.1)

<s>

the movie was great

Beam Search
Beam Search

- Maintain decoder state, token history in beam

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

```
l: 0.4
le: 0.3
les: 0.1
```

```
log(0.4) log(0.3) log(0.1)
```

```
la
```

```
film: 0.4
```
Beam Search

- Maintain decoder state, token history in beam

```
<s>
la: 0.4
le: 0.3
les: 0.1

log(0.4)
log(0.3)
log(0.1)

the movie was great

film: 0.4

log(0.4) + log(0.4)

la
le
les

log(0.4)
log(0.3)

log(0.1)

la

film

log(0.4) + log(0.4)
```
Beam Search

- Maintain decoder state, token history in beam

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

```
la: 0.4
le: 0.3
les: 0.1
```

```
log(0.4)
log(0.3)
log(0.1)
```

```
film: 0.4
```

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

```
la
le
les
```

```
film: 0.8
```

```
log(0.4)
```

```
la
```

```
le
```

```
les
```

```
the movie was great
```
Beam Search

- Maintain decoder state, token history in beam

```
<s>
the movie was great

la: 0.4
le: 0.3
les: 0.1

film: 0.4
```

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

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

```
le
```

```
film
```

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

```
l a
```

```
film
```

```
log(0.1)
```

```
le
```

```
film
```

```
log(0.4)
```

```
le
```

```
film
```

```
log(0.3)
```

```
le
```

```
film
```
Beam Search

- Maintain decoder state, token history in beam

- Do not max over the two *film* states! Hidden state vectors are different
“what states border Texas”

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

Jia and Liang (2015)
Semantic Parsing as Translation

“what states border Texas”

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

- 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

Jia and Liang (2015)
Semantic Parsing as Translation

“what states border Texas”

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

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

Locascio et al. (2016)
Regex Prediction

- Can use for other semantic parsing-like tasks

Locascio et al. (2016)
Regex Prediction

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

Locascio et al. (2016)
Regex Prediction

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

Locascio et al. (2016)
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

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

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

  Zhong et al. (2017)
SQL Generation

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

Question:
How many CFL teams are from York College?

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

Zhong et al. (2017)
SQL Generation

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

- How to ensure that well-formed SQL is generated?

Question:
How many CFL teams are from York College?

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

Zhong et al. (2017)
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

Question:
How many CFL teams are from York College?

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

Question:
How many CFL teams are from York College?

SQL:
```
SELECT COUNT CFL Team FROM CFLDraft WHERE College = "York"
```
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
Problems with Seq2seq Models

- Encoder-decoder models like to repeat themselves:
Encoder-decoder models like to repeat themselves:

Un garçon joue dans la neige → A boy plays in the snow boy plays boy plays
Problems with Seq2seq Models

- Encoder-decoder models like to repeat themselves:

  Un garçon joue dans la neige $\rightarrow$ A boy plays in the snow **boy plays boy plays**

- Often a byproduct of training these models poorly
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 *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

- 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)
- Suppose we knew the source and target would be word-by-word translated
Suppose we knew the source and target would be word-by-word translated

the movie was great

le film était bon
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!

the movie was great

le film était bon
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!
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!
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!
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!
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
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?
the movie was great <s> le
the movie was great
The movie was great.
the movie was great...
the movie was great
At each decoder state, compute a distribution over source inputs based on current decoder state.
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
Attention

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

the movie was great

\(<s>\)
Attention

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

\[ \bar{h}_1 \]

the movie was great

<s>
For each decoder state, compute weighted sum of input states

For each decoder state, compute weighted sum of input states

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

Unnormalized scalar weight
Attention

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

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

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

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

\[ c_i = \sum_j \alpha_{ij} h_j \]

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

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

Weighted sum of input hidden states (vector)

Unnormalized scalar weight

Attention

the movie was great
For each decoder state, compute weighted sum of input states

\[ c_i = \sum_j \alpha_{ij} h_j \]

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

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

Attention

- Weighted sum of input hidden states (vector)
- Unnormalized scalar weight

the movie was great
Attention

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

The movie was great

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

\[ c_i = \sum_j \alpha_{ij} h_j \]

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

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

- Weighted sum of input hidden states (vector)
- Unnormalized scalar weight
Attention

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

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

- Weighted sum of input hidden states (vector)

- Unnormalized scalar weight

\[
\begin{align*}
\alpha_{ij} &= \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij})} \\
e_{ij} &= f(\bar{h}_i, h_j) \\
c_i &= \sum_j \alpha_{ij} h_j \\
P(y_i|x, y_1, \ldots, y_{i-1}) &= \text{softmax}(W[c_i; \bar{h}_i])
\end{align*}
\]
Attention

\[ c_i = \sum_j \alpha_{ij} h_j \]

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

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

Luong et al. (2015)
Attention

\[ c_i = \sum_j \alpha_{ij} h_j \]

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

\[ e_{ij} = f(\tilde{h}_i, h_j) \]

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

- Bahdanau+ (2014): additive

Luong et al. (2015)
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

Luong et al. (2015)
\[
c_i = \sum_j \alpha_{ij} h_j
\]

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

\[
e_{ij} = f(\vec{h}_i, h_j)
\]

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

- Bahdanau+ (2014): additive
  \[
f(\vec{h}_i, h_j) = \vec{h}_i \cdot h_j
\]

- Luong+ (2015): dot product
  \[
f(\vec{h}_i, h_j) = \vec{h}_i^\top Wh_j
\]

- Luong+ (2015): bilinear

Luong et al. (2015)
Attention

- Note that this all uses outputs of hidden layers

\[
c_i = \sum_j \alpha_{ij} h_j
\]

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

\[
e_{ij} = f(\vec{h}_i, h_j)
\]

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

- Bahdanau+ (2014): additive

\[
f(\vec{h}_i, h_j) = \vec{h}_i \cdot h_j
\]

- Luong+ (2015): dot product

\[
f(\vec{h}_i, h_j) = \vec{h}_i^\top Wh_j
\]

- Luong+ (2015): bilinear

Luong et al. (2015)
Copying Input/Pointers
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.

Jean et al. (2015), Luong et al. (2015)
The ecotax portico in Pont-de-Buis, ... [truncated] ..., was taken down on Thursday morning.

Jean et al. (2015), Luong et al. (2015)
Want to be able to copy named entities like Pont-de-Buis
Want to be able to copy named entities like Pont-de-Buis

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

from attention
from RNN hidden state

Jean et al. (2015), Luong et al. (2015)
Want to be able to copy named entities like Pont-de-Buis

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

from attention

from RNN hidden state

Still can only generate from the vocabulary

Jean et al. (2015), Luong et al. (2015)
en: The *ecotax* portico in *Pont-de-Buis*, ... [truncated]...

fr: Le *portique écotaxe* de *Pont-de-Buis*, ... [truncated]

nn: Le *unk* de *unk* à *unk*, ... [truncated]..., a été pris
Copying

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

fr: Le *portique écotaxe* de *Pont-de-Buis*, ... [truncated]

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

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

en: The *ecotax* portico in *Pont-de-Buis*, … [truncated] …

fr: Le *portique écotaxe* de *Pont-de-Buis*, … [truncated]

nn: Le *unk* de *unk à unk*, … [truncated] …, a été pris

- Vocabulary contains “normal” vocab as well as words in input. Normalizes over both of these:
The ecotax portico in Pont-de-Buis, ...

Le portique écotaxe de Pont-de-Buis, ...

Le unk de unk à unk, ...

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

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

en: The ecotax portico inPont-de-Buis, ... [truncated]...

fr: Le portique écotaxe de Pont-de-Buis, ... [truncated]

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

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

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

- Bilinear function of input representation + output hidden state
Only point to the input, don’t have any notion of vocabulary
**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

<table>
<thead>
<tr>
<th></th>
<th>GEO</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Copying</td>
<td>74.6</td>
<td>69.9</td>
</tr>
<tr>
<td>With Copying</td>
<td>85.0</td>
<td>76.3</td>
</tr>
</tbody>
</table>

Jia and Liang (2016)
Results

<table>
<thead>
<tr>
<th></th>
<th>GEO</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Copying</td>
<td>74.6</td>
<td>69.9</td>
</tr>
<tr>
<td>With Copying</td>
<td>85.0</td>
<td>76.3</td>
</tr>
</tbody>
</table>

For semantic parsing, copying tokens from the input (texas) can be very useful.

Jia and Liang (2016)
Results

<table>
<thead>
<tr>
<th></th>
<th>GEO</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Copying</td>
<td>74.6</td>
<td>69.9</td>
</tr>
<tr>
<td>With Copying</td>
<td>85.0</td>
<td>76.3</td>
</tr>
</tbody>
</table>

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
LSTM abstraction: maps each vector in a sentence to a new, context-aware vector

Vaswani et al. (2017)
Self-Attention

- LSTM abstraction: maps each vector in a sentence to a new, context-aware vector

- CNNs did something similar with filters

Vaswani et al. (2017)
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

Vaswani et al. (2017)
Each word forms a “query” which then computes attention over each word.

Vaswani et al. (2017)
Self-Attention

- Each word forms a “query” which then computes attention over each word.

Vaswani et al. (2017)
Self-Attention

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

Vaswani et al. (2017)
Each word forms a “query” which then computes attention over each word

$$\alpha_{i,j} = \text{softmax}(x_i^T x_j) \quad \text{scalar}$$
Self-Attention

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

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

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

Vaswani et al. (2017)
Self-Attention

- Each word forms a “query” which then computes attention over each word
  \[ \alpha_{i,j} = \text{softmax}(x_i^\top x_j) \] 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

Vaswani et al. (2017)
Self-Attention

- Each word forms a “query” which then computes attention over each word
  \[
  \alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar}
  \]
  \[
  x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector} = \text{sum of scalar} \times \text{vector}
  \]

- Multiple “heads” analogous to different convolutional filters. Use parameters \(W_k\) and \(V_k\) to get different attention values + transform vectors

  \[
  \alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j)
  \]

Vaswani et al. (2017)
Self-Attention

- Each word forms a “query” which then computes attention over each word
  \[ \alpha_{i,j} = \text{softmax}(x_i^\top x_j) \] 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
  \[ \alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \]
  \[ x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j \]

Vaswani et al. (2017)
Self-Attention

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

\[ \alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar} \]

\[ x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar \times vector} \]

- Multiple “heads” analogous to different convolutional filters. Use parameters \( W_k \) and \( V_k \) to get different attention values + transform vectors

\[ \alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \quad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j \]

Vaswani et al. (2017)
Deep Transformers

- Supervised: transformer can replace LSTM; will revisit this when we discuss MT
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
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”
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
Takeaways

- Attention is very helpful for seq2seq models

- Used for tasks including summarization and sentence ordering
Takeaways

- Attention is very helpful for seq2seq models

- Used for tasks including summarization and sentence ordering

- Explicitly copying input can be beneficial as well
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
Where are we going

- We’ve now talked about most of the important core tools for NLP
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
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