Recall: Seq2seq Model

- Generate next word conditioned on previous word as well as hidden state.
- W size is $|voc| \times |\text{hidden state}|$, 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).

Recall: Seq2seq Training

- Objective: maximize $\sum_{(x, y)} \sum_{i=1}^{n} \log P(y_i^t|x, y_1^t, \ldots, y_{i-1}^t)$

- Teacher forcing: feed the correct word regardless of model’s prediction (most typical way to train).
Recall: 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
- Might not produce well-formed logical forms, might require lots of data

Jia and Liang (2015)

This Lecture

- Attention for sequence-to-sequence models
- Copy mechanisms for copying words to the output
- Transformer architecture

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
  - LSTM state is not behaving as expected 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:
  - 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)
  - 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)

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

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

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

\[ P(y_i|x, y_1, \ldots, y_{i-1}) = \text{softmax}(Wc_1\bar{h}_i) \]

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

\[ \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_j \exp(e_{ij})} \]

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

Some function \( f(TBD) \)

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

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
- Sentence outputs: batch size x hidden size
- Attention scores = batch size x sentence length
- Make sure tensors are the right size!

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)
Copying Input/Pointers

> 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 RNN hidden state
- from attention

> Problems: target word has to be in the vocabulary, attention + RNN need to generate good embedding to pick it

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

Pointer Networks

- Standard decoder \( P_{\text{vocab}} \): softmax over vocabulary, all words get >0 prob
- Pointer network: predict from source words instead of target vocab

\[ P_{\text{pointer}}(y_i|x, y_1, \ldots, y_{i-1}) \propto \begin{cases} h_j^T V \bar{h}_i & \text{if } y_i = w_j \\ 0 & \text{otherwise} \end{cases} \]

Pointer Generator Mixture Models

- Define the decoder model as a mixture model of the \( P_{\text{vocab}} \) and \( P_{\text{pointer}} \) models (previous slide)

\[ P(y_i|x, y_1, \ldots, y_{i-1}) = P(\text{copy})P_{\text{pointer}} + (1 - P(\text{copy}))P_{\text{vocab}} \]

- Predict \( P(\text{copy}) \) based on decoder state, input, etc.
- Marginalize over copy variable during training and inference
- Model will be able to both generate and copy, flexibly adapt between the two
### Copying

- **en:** The *écotax* portico in *Pont-de-Buis*, … [truncated] …
- **fr:** *Le portique écotax de Pont-de-Buis*, … [truncated]
- **nn:** *Le unk de unk à unk*, … [truncated] …, a été pris

Some words we may want to copy may not be in the fixed output vocab (*Pont-de-Buis*)

Solution: expand the vocabulary dynamically. New words can only be predicted by copying (always 0 probability under \(P_{\text{vocab}}\))

### 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

Copying typically helps a bit, but attention captures most of the benefit. However, vocabulary expansion is critical for some tasks (machine translation)

Jia and Liang (2016)

### Sentence Encoders

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

- CNNs do something similar with filters

- Attention can give us a third way to do this

Vaswani et al. (2017)
**Self-Attention**

- Assume we’re using GloVe — what do we want our neural network to do?
  - *The ballerina is very excited that* she will dance in the *show.*

- What words need to be contextualized here?
  - Pronouns need to look at antecedents
  - Ambiguous words should look at context
  - Words should look at syntactic parents/children

- Problem: LSTMs and CNNs don’t do this

---

**Self-Attention**

- Want:
  - *The ballerina is very excited that she will dance in the show.*

- LSTMs/CNNs: tend to look at local context

- To appropriately contextualize embeddings, we need to pass information over long distances dynamically for each word

---

**Self-Attention**

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

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

---

**What can self-attention do?**

- *The ballerina is very excited that she will dance in the show.*

\[
\begin{bmatrix}
0 & 0.5 & 0 & 0 & 0.1 & 0.1 & 0.2 & 0 & 0 & 0 \\
0 & 0.1 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0.4 & 0
\end{bmatrix}
\]

- Attend nearby + to semantically related terms

- This is a demonstration, we will revisit what these models actually learn when we discuss BERT

- Why multiple heads? Softmaxes end up being peaked, single distribution cannot easily put weight on multiple things
Transformer Uses

- Supervised: transformer can replace LSTM as encoder, decoder, or both; will revisit this when we discuss MT
- Unsupervised: transformers work better than LSTM for unsupervised pre-training of embeddings: predict word given context words
- BERT (Bidirectional Encoder Representations from Transformers): pretraining transformer language models similar to ELMo
- Stronger than similar methods, SOTA on ~11 tasks (including NER — 92.8 F1)

Takeaways

- Attention is very helpful for seq2seq models
- Explicitly copying input can be beneficial as well
- Transformers are strong models we’ll come back to later
- We’ve now talked about most of the important core tools for NLP
- Rest of the class is more focused on applications: translation, information extraction, QA, and more, then other applications