A forum for Brown folks to showcase the cool things they’ve built using large language models, generative AI, and related technologies, and a research center to advance these technologies.

Launch meeting on Monday, March 11, 5-6 pm, Macmillan Hall 115

**Agenda**

Organizational meeting and talk by Isaac Wecht (Masters, DSI) on *Using LangChain and OpenAI to create a question answering system.*
Review: seq2seq Models

• Last time, we saw an encoder-decoder architecture for sequence-to-sequence learning
Encoder LSTM

Hansards

Révisé

Numéro

1

STOP

Final LSTM state as sentence embedding

Decoder LSTM

Revised

Hansards

Number

1

STOP

Dense layer

Hansards

Number

1

STOP
Review: seq2seq Models

• Last time, we saw an encoder-decoder architecture for sequence-to-sequence learning
• We also saw how, instead of initializing the decoder with the final state of the encoder, we could use the sum of all encoder states
Old:

\[ E_s = \text{final LSTM state} \]
New: 
\[ E_s = \text{sum of LSTM states} \] 

What are the information bottlenecks?
What if the decoder LSTM forgets the sentence embedding?
What if we passed the sum of our encoder states to every cell in the decoder?

Just summing might not be sufficient! Different words in the source have different importance for the target.
What if we passed the sum of our encoder states to every cell in the decoder?

What if the sum was a weighted sum instead?
- Idea: different words in the input carry different importance
What if each decoder cell received a different weighted sum?

- **Idea:** different words in the input carry different importance for each word in the output.

What if we passed the sum of our encoder states to **every cell** in the decoder?

What if the sum was a **weighted sum** instead?

**How do we achieve this?**

What if each decoder cell received a **different** weighted sum?

- **Idea:** different words in the input carry different importance for each word in the output.
This idea of passing each cell of the decoder a weighted sum of the encoder states is called **attention**.

- Different words in the output “pay attention” to different words in the input
“Attention” - intuition

“How about we let the model learn what is relevant for a particular output?”
Attention - implementation

Decoder

Encoder
Attention - implementation

\[ x_t \]

\[ y_t \]

Decoder

Encoder

\[ s_{t-1} \rightarrow s_t \]

\[ c_t \]

\[ x_t \]

\[ y_t \]

\[ s_t \]

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5 \]

\[ \text{hansards} \]

\[ \text{révisé} \]

\[ \text{numéro} \]

\[ 1 \]

\[ \text{STOP} \]
Attention - implementation

Decoder

Encoder

Attention layer

\[
\begin{align*}
\alpha_{t,1} & \quad \alpha_{t,2} & \quad \alpha_{t,3} & \quad \alpha_{t,4} & \quad \alpha_{t,5} \\
\times+ & & & & \\
\end{align*}
\]

\[
\begin{align*}
h_1 & \quad \rightarrow \quad h_2 & \quad \rightarrow \quad h_3 & \quad \rightarrow \quad h_4 & \quad \rightarrow \quad h_5 \\
\text{hansards} & \quad \text{révisé} & \quad \text{numéro} & \quad 1 & \quad \text{STOP} \\
\end{align*}
\]
\[
\begin{align*}
\mathbf{x} &= [x_1, x_2, \ldots, x_n] \\
\mathbf{y} &= [y_1, y_2, \ldots, y_m] \\
c_t &= \sum_{i=1}^{n} \alpha_{t,i} \mathbf{h}_i \quad \text{Context vector for output } y_t
\end{align*}
\]

\[
\alpha_{t,i} = \text{align}(y_t, x_i)
\]

How well two words \(y_t\) and \(x_i\) are aligned.

\[
\frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))}
\]

Softmax of some predefined alignment score.

Any questions?
Attention alignment score functions

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i' = 1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))}$$

Softmax of some predefined alignment score.

How well two words $y_t$ and $x_i$ are aligned.

How to measure this?
Any ideas?
Attention alignment score functions

\[ \alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))} \]

Softmax of some predefined alignment score.

How well two words \( y_t \) and \( x_i \) are aligned.

General attention:

\[ \text{score}(s_{t-1}, h_i) = s_{t-1}^T W_a h_i \]
Attention alignment score functions

\[ \alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))} \]

Softmax of some predefined alignment score.

How well two words \( y_t \) and \( x_i \) are aligned.

<table>
<thead>
<tr>
<th>Name</th>
<th>Alignment score function</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-base attention</td>
<td>( \text{score}(s_t, h_i) = \cosine[s_t, h_i] )</td>
<td>Graves2014</td>
</tr>
<tr>
<td>Additive(*)</td>
<td>( \text{score}(s_t, h_i) = v_a^T \tanh(W_a[s_t; h_i]) )</td>
<td>Bahdanau2015</td>
</tr>
<tr>
<td>Location-Base</td>
<td>( \alpha_{t,i} = \text{softmax}(W_a s_i) )</td>
<td>Luong2015</td>
</tr>
<tr>
<td></td>
<td>Note: This simplifies the softmax alignment to only depend on the target position.</td>
<td></td>
</tr>
<tr>
<td>General</td>
<td>( \text{score}(s_t, h_i) = s_i^T W_a h_i )</td>
<td>Luong2015</td>
</tr>
<tr>
<td></td>
<td>where ( W_a ) is a trainable weight matrix in the attention layer.</td>
<td></td>
</tr>
<tr>
<td>Dot-Product</td>
<td>( \text{score}(s_t, h_i) = s_i^T h_i )</td>
<td>Luong2015</td>
</tr>
<tr>
<td>Scaled Dot-Product((^\wedge))</td>
<td>( \text{score}(s_t, h_i) = \frac{s_i^T h_i}{\sqrt{n}} )</td>
<td>Vaswani2017</td>
</tr>
<tr>
<td></td>
<td>Note: very similar to the dot-product attention except for a scaling factor; where ( n ) is the dimension of the source hidden state.</td>
<td></td>
</tr>
</tbody>
</table>
Attention types

\[ \alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^n \exp(\text{score}(s_{t-1}, h_{i'}))} \]

Softmax of some predefined alignment score.

How well two words \( y_t \) and \( x_i \) are aligned.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global/Soft</td>
<td>Attending to the entire input state space.</td>
<td>Xu2015</td>
</tr>
</tbody>
</table>

Attention types

\[ \alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))} \]

Softmax of some predefined alignment score.

How well two words \( y_t \) and \( x_i \) are aligned.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global/Soft</td>
<td>Attending to the entire input state space.</td>
<td>( \text{Xu2015} )</td>
</tr>
<tr>
<td>Local/Hard</td>
<td>Attending to the part of input state space; i.e. a patch of the input image.</td>
<td>( \text{Xu2015; Luong2015} )</td>
</tr>
</tbody>
</table>
Attention types

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))}$$

Softmax of some predefined alignment score.

How well two words $y_t$ and $x_i$ are aligned.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Citation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention(&amp;)</td>
<td>Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, but just replace the target sequence with the same input sequence.</td>
<td>Cheng2016</td>
</tr>
<tr>
<td>Global/Soft</td>
<td>Attending to the entire input state space.</td>
<td>Xu2015</td>
</tr>
<tr>
<td>Local/Hard</td>
<td>Attending to the part of input state space; i.e. a patch of the input image.</td>
<td>Xu2015; Luong2015</td>
</tr>
</tbody>
</table>

Any questions?
\[ x = [x_1, x_2, \ldots, x_n] \]
\[ y = [y_1, y_2, \ldots, y_m] \]
\[ c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \]  
Context vector for output \( y_t \)

\[ \alpha_{t,i} = \text{align}(y_t, x_i) \]
How well two words \( y_t \) and \( x_i \) are aligned.

\[
\alpha_{t,1} = \frac{\exp(\text{score}(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(\text{score}(s_{t-1}, h_{i'}))}
\]
Softmax of some predefined alignment score.
Attention Example

We can represent the attention weights as a matrix:

Columns: words in the input

<table>
<thead>
<tr>
<th></th>
<th>hansards</th>
<th>révisé</th>
<th>numéro</th>
<th>1</th>
<th>STOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>revised</td>
<td>1/2</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hansards</td>
<td>1/4</td>
<td>1/2</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>number</td>
<td>0</td>
<td>1/4</td>
<td>1/2</td>
<td>1/4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>1/2</td>
<td>1/4</td>
</tr>
<tr>
<td>STOP</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>1/4</td>
<td>1/2</td>
</tr>
</tbody>
</table>

$\alpha_{j,i}$: how much ‘attention’ output word j pays to input word i

What do the values in this particular matrix imply about the attention relationship between input/output words?
Attention Example

We can represent the attention weights as a matrix:

Columns: words in the input

<table>
<thead>
<tr>
<th></th>
<th>hansards</th>
<th>révisé</th>
<th>numéro</th>
<th>1</th>
<th>STOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>revised</td>
<td>1/2</td>
<td>1/4</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hansards</td>
<td>1/4</td>
<td>1/2</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>number</td>
<td>0</td>
<td>1/4</td>
<td>1/2</td>
<td>1/4</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>1/2</td>
<td>1/4</td>
</tr>
<tr>
<td>STOP</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>1/4</td>
<td>1/2</td>
</tr>
</tbody>
</table>

$\alpha_{j,i}$: how much ‘attention’ output word j pays to input word i

“Words that are similar between the input and output influence each other the most”
Another Attention Example

Target: “Der Hund bellte mich an.”

Input: “The dog barked at me.”
Attention Example

Target: “Der Hund **bellte** mich an.”

Input: “The **dog** **barked** at me.”

We see that when we apply the attention to our inputs, we will pay attention to relatively important words for translation when predicting “**bellte**”.
Another Attention Example

Target: “Der Hund hatte mich angebellt.”

Input: “The dog had barked at me.”

Here, the verb portion of a past participle in German appears at the end of the sequence (What now?)

Attention weight matrix is another learnable parameter of the model!

Model will re-adjust the weights
Attention in Language Translation

Attention helps solve the alignment problem!
Attention is great!

- Attention significantly **improves MT performance**
  - It’s very useful to allow decoder to focus on certain parts of the source

- Attention **solves the bottleneck problem**
  - Attention allows decoder to look directly at source; bypass bottleneck

- Attention **helps with vanishing gradient problem**
  - Provides shortcut to faraway states

- Attention provides **some interpretability**
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself

*Can you think of any another advantage?*
Attention is a general deep learning technique

More general definition of attention:
Given a set of vector \textit{values}, and a vector \textit{query}, \textbf{attention} is a technique to compute a weighted sum of the values, dependent on the query.

**Intuition:**

- The weighted sum is a \textit{selective summary} of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a \textit{fixed-size representation of an arbitrary set of representations} (the values), dependent on some other representation (the query).
Think-pair-share:

How would you design this architecture with attention?

A **dog** is standing on a hardwood floor.

A **stop** sign is on a road with a mountain in the background.

A group of **people** sitting on a boat in the water.

A giraffe standing in a forest with **trees** in the background.
Image captioning with CNNs, RNNs, and Attention

Image captioning with CNNs, RNNs, and Attention

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.

A large white **bird** standing in a forest.  
A woman holding a **clock** in her hand.  
A man wearing a hat and a hat on a **skateboard**.

A person is standing on a beach with a **surfboard**.  
A woman is sitting at a table with a large **pizza**.  
A man is talking on his cell **phone** while another man watches.

Image captioning (HW5)

Same idea as Machine Translation, just replace $E_s$ with an image-level embedding.
Do we still need the RNNs?

After all, we always compute the weighted sum of all encoder states.
“Attention Is All You Need”

A 2017 paper that introduced the Transformer model for machine translation

- Has no recurrent networks!
- Only uses attention

Motivation:

- RNN training is hard to parallelize since the previous word must be processed before next word
  - Transformers are trivially parallelizable
- Even with LSTMs / GRUs, preserving important linguistic context over very long sequences is difficult
  - Transformers don’t even try to remember things (every step looks at a weighted combination of all words in the input sentence)
Transformer Model Overview

- The Transformer model breaks down into Encoder and Decoder blocks.
- At a high level, similar to the seq2seq architecture we've seen already...
- ...but there are no recurrent nets inside the Encoder and Decoder blocks!

Transformer Model Overview

- The Transformer model breaks down into Encoder and Decoder blocks.
- At a high level, similar to the seq2seq architecture we've seen already...
- ...but there are no recurrent nets inside the Encoder and Decoder blocks!
- For better performance, often stack multiple Encoder and Decoder blocks (deeper network)

To be continued in next class!

Recap

Attention helps remove bottlenecks in simple encoder-decoder model

Attention score functions and types

Attention weights are learnable

Attention for MT

Interpretation

Image captioning (HW5)

Attention is all you need (Transformers)

A dog is standing on a hardwood floor.