Sequence-to-sequence models with attention

Many slides adapted from J. Johnson
Outline

• Vanilla seq2seq with RNNs
• Seq2seq with RNNs and attention
• Image captioning with attention
• Transformers
Sequence-to-sequence modeling: Machine translation

“We are eating bread”  →  “Estamos comiendo pan”
Sequence-to-sequence modeling with RNNs

I. Sutskever, O. Vinyals, Q. Le, *Sequence to Sequence Learning with Neural Networks*, NeurIPS 2014

Sequence-to-sequence modeling with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_{t-1}, s_{t-1})$

From final hidden state predict: Initial decoder state $s_0$

Problem: It is too hard for the decoder state to keep track both of the input context and the decoder context
Sequence-to-sequence modeling with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

- Initial decoder state $s_0$
- Context vector $c$ (often $c = h_T$)

Solution: Introduce a separate context vector to “remember” the input sequence
Sequence-to-sequence modeling with RNNs

Encoder: \( h_t = f_W(x_t, h_{t-1}) \)

From final hidden state predict:
- Initial decoder state \( s_0 \)
- Context vector \( c \) (often \( c = h_T \))

Decoder: \( s_t = g_V(y_{t-1}, s_{t-1}, c) \)

\[ \text{we are eating bread} \rightarrow \text{estamos comiendo pan [STOP]} \]
Sequence-to-sequence modeling with RNNs

Encoder: \( h_t = f_W(x_t, h_{t-1}) \)

From final hidden state predict:
- Initial decoder state \( s_0 \)
- Context vector \( c \) (often \( c = h_T \))

Decoder: \( s_t = g_U(y_{t-1}, s_{t-1}, c) \)

Problem: Input sequence bottlenecked through fixed-sized vector

Idea: use new context vector at each step of decoder!

\[ \begin{align*}
\text{we} & \quad \text{are} \quad \text{eating} \quad \text{bread} \\
\text{estamos} & \quad \text{comiendo} \quad \text{pan} \\
\text{[START]} & \quad \text{[STOP]} \\
\end{align*} \]
Sequence-to-sequence with RNNs and attention

• Intuition: translation requires alignment
Sequence-to-sequence with RNNs and attention

- At each timestep of decoder, context vector “looks at” different parts of the input sequence

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

\[ h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4 \rightarrow s_0 \]

we are eating bread
Sequence-to-sequence with RNNs and attention

\[ a_{11}, a_{12}, a_{13}, a_{14} \]

Normalize to get attention weights \( a_{t,i} \)

\[ e_{11}, e_{12}, e_{13}, e_{14} \]

Compute scalar alignment scores \( e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \)

\[ h_1, h_2, h_3, h_4 \]

Compute context vector as \( c_t = \sum_i a_{t,i} h_i \)

\[ s_0 \rightarrow s_1 \]

Use context vector in decoder:

\[ s_t = g_U(y_{t-1}, s_{t-1}, c_t) \]

Intuition: Context vector “attends” to the relevant part of the input sequence “estamos” = “we are” so maybe \( a_{11} = a_{12} = 0.45, a_{13} = a_{14} = 0.05 \)

Sequence-to-sequence with RNNs and attention

We are eating bread.

Repeat: Use \( s_1 \) to compute new context vector \( c_2 \)

Use \( c_2 \) to compute \( s_2, y_2 \)
Sequence-to-sequence with RNNs and attention
Sequence-to-sequence with RNNs and attention

we are eating bread

softmax

estamos comiendo pan

[START] estamos comiendo pan

[STOP]
Sequence-to-sequence with RNNs and attention

- Visualizing attention weights (English source, French target):

Quantitative evaluation

Google Neural Machine Translation (GNMT)

Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi
yonghui,schuster,zhifengc,qvl,mnorouzi@google.com

Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideko Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean

Y. Wu et al., Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, arXiv 2016

https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html
Google Neural Machine Translation (GNMT)

Y. Wu et al., Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, arXiv 2016
Google Neural Machine Translation (GNMT)

- **Standard training objective:** maximize log-likelihood of ground truth output given input:
  \[ \sum_{i} \log P_{W}(Y_{i}^{*}|X_{i}) \]
  
  - Only encourages the system to reproduce the reference sentences, does not induce a very good ranking on outputs that don’t match reference sentences
  - Not related to task-specific evaluation metric (e.g., BLEU score)

- **Refinement objective:** expectation of rewards over possible predicted sentences \( Y \):
  \[ \sum_{i} \sum_{Y} P_{W}(Y|X_{i}) R(Y,Y_{i}^{*}) \]
  
  - Use variant of BLEU score to compute reward
  - Reward is not differentiable -- need RL to train (initialize with ML-trained model)
Google Neural Machine Translation (GNMT)

- Human evaluation results on production data (500 randomly sampled sentences from Wikipedia and news websites)

<table>
<thead>
<tr>
<th></th>
<th>PBMT</th>
<th>GNMT</th>
<th>Human</th>
<th>Relative Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>English → Spanish</td>
<td>4.885</td>
<td>5.428</td>
<td>5.550</td>
<td>87%</td>
</tr>
<tr>
<td>English → French</td>
<td>4.932</td>
<td>5.295</td>
<td>5.496</td>
<td>64%</td>
</tr>
<tr>
<td>English → Chinese</td>
<td>4.035</td>
<td>4.594</td>
<td>4.987</td>
<td>58%</td>
</tr>
<tr>
<td>Spanish → English</td>
<td>4.872</td>
<td>5.187</td>
<td>5.372</td>
<td>63%</td>
</tr>
<tr>
<td>French → English</td>
<td>5.046</td>
<td>5.343</td>
<td>5.404</td>
<td>83%</td>
</tr>
<tr>
<td>Chinese → English</td>
<td>3.694</td>
<td>4.263</td>
<td>4.636</td>
<td>60%</td>
</tr>
</tbody>
</table>

**Side-by-side scores:** range from 0 (“completely nonsense translation”) to 6 (“perfect translation”), produced by human raters fluent in both languages

**PBMT:** Translation by phrase-based statistical translation system used by Google

**GNMT:** Translation by GNMT system

**Human:** Translation by humans fluent in both languages
Outline

- Vanilla seq2seq with RNNs
- Seq2seq with RNNs and attention
- Image captioning with attention
Generalizing attention

\[ c_t = \sum_i a_{t,i} h_i \]

- The decoder doesn’t use the fact that the \( h_i \) form an ordered sequence – it just treats them as an unordered set
- Can use similar architecture given any set of input hidden vectors \( \{h_i\} \)!
Image captioning with RNNs and attention

• Idea: pay attention to different parts of the image when generating different words

• Automatically learn this *grounding* of words to image regions without direct supervision

K. Xu et al., *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, ICML 2015
Image captioning with RNNs and attention

\[ e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j}) \]

K. Xu et al., *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, ICML 2015
Image captioning with RNNs and attention

\[ e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j}) \]

\[ c_t = \sum_i a_{t,i,j} h_i \]
Image captioning with RNNs and attention

Use a CNN to compute a grid of features for an image

\[ e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j}) \]

\[ c_t = \sum_i a_{t,i,j} h_i \]
Each time step of decoder uses a different context vector that looks at different parts of the input image.
Example results

• Good captions

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

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

A little girl sitting on a bed with a teddy bear.

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

A giraffe standing in a forest with trees in the background.
Example results

- Mistakes

  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.
## Quantitative results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>BLEU-1</th>
<th>BLEU-2</th>
<th>BLEU-3</th>
<th>BLEU-4</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr8k</td>
<td>Google NIC</td>
<td>63</td>
<td>41</td>
<td>27</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Soft-Attention</td>
<td>67</td>
<td>44.8</td>
<td>29.9</td>
<td>19.5</td>
<td>18.93</td>
</tr>
<tr>
<td></td>
<td>Hard-Attention</td>
<td>67</td>
<td>45.7</td>
<td>31.4</td>
<td>21.3</td>
<td>20.30</td>
</tr>
<tr>
<td>Flickr30k</td>
<td>Google NIC</td>
<td>66.3</td>
<td>42.3</td>
<td>27.7</td>
<td>18.3</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Soft-Attention</td>
<td>66.7</td>
<td>43.4</td>
<td>28.8</td>
<td>19.1</td>
<td>18.49</td>
</tr>
<tr>
<td></td>
<td>Hard-Attention</td>
<td><strong>66.9</strong></td>
<td><strong>43.9</strong></td>
<td><strong>29.6</strong></td>
<td><strong>19.9</strong></td>
<td>18.46</td>
</tr>
<tr>
<td>COCO</td>
<td>Google NIC</td>
<td>66.6</td>
<td>46.1</td>
<td>32.9</td>
<td>24.6</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Soft-Attention</td>
<td>70.7</td>
<td>49.2</td>
<td>34.4</td>
<td>24.3</td>
<td><strong>23.90</strong></td>
</tr>
<tr>
<td></td>
<td>Hard-Attention</td>
<td><strong>71.8</strong></td>
<td><strong>50.4</strong></td>
<td><strong>35.7</strong></td>
<td><strong>25.0</strong></td>
<td>23.04</td>
</tr>
</tbody>
</table>
**Sequence modeling beyond RNNs**

**RNNs**

Works on **ordered sequences**
- Pros: Not limited by fixed context size (in principle): After one RNN layer, $h_T$ "sees" the whole sequence
- Con: Hidden states have limited expressive capacity
- Con: Not parallelizable: need to compute hidden states sequentially

**1D convolutional networks**

Works on **multidimensional grids**
- Pro: Each output can be computed in parallel (at training time)
- Con: Bad at long sequences: Need to stack many conv layers for outputs to “see” the whole sequence

**Transformers**

- Works on **sets of vectors**
Outline

- Vanilla seq2seq with RNNs
- Seq2seq with RNNs and attention
- Image captioning with attention
- Transformers
Basic transformer model

- Sequence-to-sequence architecture using *only point-wise processing and attention* – no recurrent units or convolutions

**Encoder**: receives entire input sequence and outputs encoded sequence of the same length

**Decoder**: predicts next token conditioned on encoder output and previously predicted tokens

---

Key-Value-Query attention model

The decoder generates a **query** describing what it wants to focus on

- Compute dot products between the query and the **keys** generated by encoder, giving alignment scores between source tokens and the query
- Feed the scores into a softmax to create the attention weights
- Sum the **values** generated by encoder weighted by the attention weights

Decoder

Encoder

$X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4$
Key-Value-Query attention model

- Key vectors: \( K = XW_K \)
- Value Vectors: \( V = XW_V \)
- Query vectors
- Similarities: scaled dot-product attention
  \[
  E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}} \quad \text{or} \quad E = QK^T / \sqrt{D}
  \]
  \( (D \) is the dimensionality of the keys)\)
- Attn. weights: \( A = \text{softmax}(E, \text{dim} = 1) \)
- Output vectors:
  \[
  Y_i = \sum_j A_{i,j}V_j \quad \text{or} \quad Y = AV
  \]

Adapted from J. Johnson
Key-Value-Query attention model

- How does permuting the order of the queries change the output?
- How does changing the order of the keys/values change the output?

Adapted from J. Johnson
Attention mechanisms

- **Encoder self-attention**: queries, keys, and values come from previous layer of encoder
- **Decoder self-attention**: values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention**: queries come from previous decoder layer, keys and values come from output of encoder
Self-attention

- Used to capture context *within the sequence*

As we are encoding “it”, we should focus on “the animal”

As we are encoding “it”, we should focus on “the street”
Self-attention layer

- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*
  \[ E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}} \text{ or } E = QK^T / \sqrt{D} \]
  $(D$ is the dimensionality of the keys$)$
- Attn. weights: $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:
  \[ Y_i = \sum_j A_{i,j} V_j \text{ or } Y = AV \]

Adapted from J. Johnson
Masked self-attention layer

- The decoder should not “look ahead” in the output sequence

Adapted from J. Johnson
Masked self-attention layer

- The decoder should not “look ahead” in the output sequence

Adapted from J. Johnson
Masked self-attention layer

- The decoder should not “look ahead” in the output sequence

Adapted from J. Johnson
Attention mechanisms: Summary

- **Encoder self-attention**: queries, keys, and values come from previous layer of encoder
- **Decoder self-attention**: values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention**: queries come from previous decoder layer, keys and values come from output of encoder
Attention mechanisms: Illustration

Transformer architecture: Details

A. Vaswani et al., *Attention is all you need*, NeurIPS 2017
Positional encoding

- To give transformer information about ordering of tokens, add function of position (based on sines and cosines) to every input
Multi-head attention

- Run $h$ attention models in parallel on top of different linearly projected versions of $Q, K, V$; concatenate and linearly project the results
- Intuition: enables model to attend to different kinds of information at different positions (see visualization tool)
Transformer blocks

- A **Transformer** is a sequence of transformer blocks
  - Vaswani et al.: N=12 blocks, embedding dimension = 512, 6 attention heads
  - **Add & Norm**: residual connection followed by layer normalization
  - **Feedforward**: two linear layers with ReLUs in between, applied independently to each vector
  - Attention is the only interaction between inputs!
Transformer architecture: Zooming back out

A. Vaswani et al., *Attention is all you need*, NeurIPS 2017
Results

English German Translation quality

English French Translation Quality

Transformers: Pros and cons

**RNNs**
- Works on ordered sequences
  - Pros: Not limited by fixed context size (in principle): After one RNN layer, $h_T$ "sees" the whole sequence
  - Con: Not parallelizable: need to compute hidden states sequentially
  - Con: Hidden states have limited expressive capacity

**1D convolutional networks**
- Works on multidimensional grids
  - Pros: Each output can be computed in parallel (at training time)
  - Con: Need to stack many conv layers for outputs to “see” the whole sequence

**Transformers**
- Works on sets of vectors
  - Pros: Good at long sequences: after one self-attention layer, each output "sees" all inputs!
  - Pro: Each output can be computed in parallel (at training time)
  - Con: Memory-intensive: cost of attention operator is quadratic in input size
Making transformers more efficient