Lecture 8: Attention and Transformers
Administrative

- Assignment 2 due **05/06**
- Discussion section **tomorrow**
  - Covering PyTorch, the main deep learning framework used by AI researchers + what we recommend for your projects!
Last Time: Recurrent Neural Networks
Last Time: Variable length computation graph with shared weights

\[ y_1 \rightarrow L_1 \]
\[ h_0 \rightarrow f_W \rightarrow h_1 \]
\[ W \rightarrow x_1 \]
Last Time: Variable length computation graph with shared weights

$h_0 \xrightarrow{f_W} h_1 \xrightarrow{f_W} h_2$

$L_1 \xrightarrow{f_W} y_1 \xrightarrow{L_1} h_1$

$L_2 \xrightarrow{f_W} y_2 \xrightarrow{L_2} h_2$

$W$ is reused (recurrently)!
Last Time: Variable length computation graph with shared weights

Calculate total loss across all timesteps to find $dW/dL$ (backpropagation through time)!
**Sequence to Sequence with RNNs: Encoder - Decoder**

**Input:** Sequence \( x_1, \ldots, x_T \)

**Output:** Sequence \( y_1, \ldots, y_T \)

A motivating example for today’s discussion – machine translation! English \( \rightarrow \) Italian

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

Sutskever et al., “Sequence to sequence learning with neural networks”, NeurIPS 2014

Fei-Fei Li, Ehsan Adeli, Zane Durante  

Lecture 9 - 7  

April 25, 2024
Sequence to Sequence with RNNs

**Input:** Sequence $x_1, \ldots, x_T$

**Output:** Sequence $y_1, \ldots, y_T$

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

From final hidden state predict:

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

---

Sutskever et al., "Sequence to sequence learning with neural networks", NeurIPS 2014

---

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 9 - 8

April 25, 2024
Sequence to Sequence with RNNs

**Input:** Sequence $x_1, \ldots, x_T$

**Output:** Sequence $y_1, \ldots, y_T$

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

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

From final hidden state predict:

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

Sutskever et al., “Sequence to sequence learning with neural networks”, NeurIPS 2014
Sequence to Sequence with RNNs

**Input:** Sequence \(x_1, \ldots, x_T\)

**Output:** Sequence \(y_1, \ldots, y_T\)

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

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

From final hidden state predict:

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

Sutskever et al., “Sequence to sequence learning with neural networks”, NeurIPS 2014

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 9 - 10

April 25, 2024
Input: Sequence $x_1, \ldots, x_T$
Output: Sequence $y_1, \ldots, y_T$

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

From final hidden state predict:

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

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

Sutskever et al., "Sequence to sequence learning with neural networks", NeurIPS 2014
Sequence to Sequence with RNNs

Input: Sequence $x_1, \ldots, x_T$
Output: Sequence $y_1, \ldots, y_T$

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

Decoder: $s_t = g(U(y_{t-1}, s_{t-1}, c)$

Remember:

**During Training:**
Often, we use the “correct” token even if the model is wrong. Called **teacher forcing**

**During Test-time:**
We sample from the model’s outputs until we sample [STOP]

---

During Training:

Input: Sequence $x_1, \ldots, x_T$
Output: Sequence $y_1, \ldots, y_T$

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

Decoder: $s_t = g(U(y_{t-1}, s_{t-1}, c)$

Remember:

**During Training:**
Often, we use the “correct” token even if the model is wrong. Called **teacher forcing**

**During Test-time:**
We sample from the model’s outputs until we sample [STOP]
Sequence to Sequence with RNNs

**Input:** Sequence \(x_1, \ldots, x_T\)

**Output:** Sequence \(y_1, \ldots, y_T\)

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

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

From final hidden state predict:

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

Q: Are there any problems with using C like this??

Sutskever et al., “Sequence to sequence learning with neural networks”, NeurIPS 2014
Sequence to Sequence with RNNs

**Input:** Sequence $x_1, \ldots, x_T$

**Output:** Sequence $y_1, \ldots, y_T$

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

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

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

Answer: Input sequence bottlenecked through fixed-sized vector. What if $T=1000$?

---

Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Fei-Fei Li, Ehsan Adeli, Zane Durante  Lecture 9 - 14  April 25, 2024
Sequence to Sequence with RNNs

Input: Sequence $x_1, \ldots, x_T$
Output: Sequence $y_1, \ldots, y_T$

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

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

Ideally we can reference the inputs as we decode...

Sequence to Sequence with RNNs and Attention

**Input:** Sequence \( x_1, \ldots, x_T \)

**Output:** Sequence \( y_1, \ldots, y_T' \)

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

From final hidden state:

**Initial decoder state** \( s_0 \)

---

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Sequence to Sequence with RNNs and **Attention**

Compute (scalar) **alignment scores**
\[ e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \]
(f_{\text{att}} is a Linear Layer)

![Diagram of RNNs with attention](image)

**From final hidden state:**
**Initial decoder state** \( s_0 \)

we see the sky

**Bahdanau et al., “Neural machine translation by jointly learning to align and translate”, ICLR 2015**
Sequence to Sequence with RNNs and Attention

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

(f_{\text{att}} is a Linear Layer)

Normalize alignment scores to get attention weights
\[ 0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1 \]

From final hidden state:
Initial decoder state \( s_0 \)

Bahdanau et al., "Neural machine translation by jointly learning to align and translate", ICLR 2015
Sequence to Sequence with RNNs and Attention

Compute (scalar) alignment scores
\[ e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \quad (f_{\text{att}} \text{ is a Linear Layer}) \]

Normalize alignment scores to get attention weights
\[ 0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1 \]

Compute context vector as weighted sum of hidden states
\[ c_t = \sum_i a_{t,i} h_i \]

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
Sequence to Sequence with RNNs and Attention

Intuition: Context vector attends to the relevant part of the input sequence “vediamo” = “we see” so maybe $a_{11} = a_{12} = 0.45, a_{13} = a_{14} = 0.05$

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

Normalize alignment scores to get attention weights $0 < a_{t,i} < 1 \sum a_{t,i} = 1$

Compute context vector as weighted sum of hidden states $c_t = \sum a_{t,i} h_i$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Sequence to Sequence with RNNs and **Attention**

**Intuition:** Context vector attends to the relevant part of the input sequence “vediamo” = “we see” so maybe $a_{11} = a_{12} = 0.45$, $a_{13} = a_{14} = 0.05$

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

$$c_t = \sum_i a_{t,i} h_i$$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

This is all differentiable! No supervision on attention weights – backprop through everything

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Sequence to Sequence with RNNs and Attention

Repeat: Use $s_1$ to compute new context vector $c_2$

Compute (scalar) alignment scores

$$e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$$

($f_{\text{att}}$ is a Linear Layer)

Bahtdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
Sequence to Sequence with RNNs and Attention

Repeat: Use $s_1$ to compute new context vector $c_2$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Fei-Fei Li, Ehsan Adeli, Zane Durante
Lecture 9 - 23 April 25, 2024
Sequence to Sequence with RNNs and **Attention**

Repeat: Use $s_1$ to compute new context vector $c_2$

Intuition: Context vector attends to the relevant part of the input sequence “*il*” = “*the*”

so maybe $a_{21}=a_{22}=0.05$, $a_{24}=0.1$, $a_{23}=0.8$

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Sequence to Sequence with RNNs and Attention

Use a different context vector in each timestep of decoder

- Input sequence not bottlenecked through single vector
- At each timestep of decoder, context vector “looks at” different parts of the input sequence

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Sequence to Sequence with RNNs and Attention

**Example:** English to French translation

\[
\begin{array}{cccc}
\text{softmax} \\
\begin{array}{cccc}
\text{e}_{t1} & \text{e}_{t2} & \text{e}_{t3} & \text{e}_{t4} \\
\text{h}_1 & \text{h}_2 & \text{h}_3 & \text{h}_4 \\
\text{x}_1 & \text{x}_2 & \text{x}_3 & \text{x}_4 \\
\text{we} & \text{see} & \text{the} & \text{sky} \\
\end{array}
\end{array}
\]

Visualize attention weights \( a_{t,i} \)

Bahdanau et al., "Neural machine translation by jointly learning to align and translate“, ICLR 2015
Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Sequence to Sequence with RNNs and Attention

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”


Bahdanau et al., “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Sequence to Sequence with RNNs and Attention

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”


Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Sequence to Sequence with RNNs and Attention

Context vectors don’t use the fact that $h_i$ form an ordered sequence – it just treats them as an unordered set \{h_i\}

General architecture + strategy given any set of input hidden vectors \{h_i\}! (calculate attention weights + sum)

Bahdanau et al., “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Fei-Fei Li, Ehsan Adeli, Zane Durante
Image Captioning using spatial features

**Input**: Image $I$

**Output**: Sequence $y = y_1, y_2, ..., y_T$

An example network for image captioning without attention

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

Image Captioning using spatial features

**Input:** Image I
**Output:** Sequence \( y = y_1, y_2, ..., y_T \)

**Encoder:** \( h_0 = f_W(z) \)
where \( z \) is spatial CNN features
\( f_W(\cdot) \) is an MLP

---


Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 9 - 32
April 25, 2024
Image Captioning using spatial features

Input: Image I
Output: Sequence $y = y_1, y_2, ..., y_T$

Encoder: $h_0 = f_W(z)$
where $z$ is spatial CNN features
$f_W(.)$ is an MLP

Decoder: $h_t = g_V(y_{t-1}, h_{t-1}, c)$
where context vector $c$ is often $c = h_0$
and output $y_t = T(h_t)$

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

Image Captioning using spatial features

**Input:** Image I
**Output:** Sequence \( y = y_1, y_2, ..., y_T \)

**Encoder:** \( h_0 = f_w(z) \)
where \( z \) is spatial CNN features
\( f_w(.) \) is an MLP

**Decoder:** \( h_t = g_v(y_{t-1}, h_{t-1}, c) \)
where context vector \( c \) is often \( c = h_0 \)
and output \( y_t = T(h_t) \)

Extract spatial features from a pretrained CNN

Image Captioning using spatial features

**Input:** Image I  
**Output:** Sequence $y = y_1, y_2, \ldots, y_T$

**Encoder:** $h_0 = f_W(z)$  
where $z$ is spatial CNN features  
$f_W(\cdot)$ is an MLP

**Decoder:** $h_t = g_V(y_{t-1}, h_{t-1}, c)$  
where context vector $c$ is often $c = h_0$  
and output $y_t = T(h_t)$

Image Captioning using spatial features

**Input:** Image $I$

**Output:** Sequence $y = y_1, y_2, ..., y_T$

**Encoder:** $h_0 = f_w(z)$
where $z$ is spatial CNN features
$f_w(\cdot)$ is an MLP

**Decoder:** $h_t = g_V(y_{t-1}, h_{t-1}, c)$
where context vector $c$ is often $c = h_0$
and output $y_t = T(h_t)$

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

Image Captioning using spatial features

**Input**: Image \( I \)

**Output**: Sequence \( y = y_1, y_2, \ldots, y_T \)

**Encoder**: \( h_0 = f_W(z) \)

where \( z \) is spatial CNN features

\( f_W(\cdot) \) is an MLP

**Decoder**: \( h_t = g_V(y_{t-1}, h_{t-1}, c) \)

where context vector \( c \) is often \( c = h_0 \)

and output \( y_t = T(h_t) \)

---

**Extract spatial features from a pretrained CNN**

**Features**: \( H \times W \times D \)

---

Image Captioning using spatial features

**Answer:** Input is "bottlenecked" through c
- Model needs to encode everything it wants to say within c

This is a problem if we want to generate really long descriptions? 100s of words long

---

Extract spatial features from a pretrained CNN

Features: H x W x D

Image Captioning with RNNs and **Attention**

**Attention idea:** New context vector at every time step.

Each context vector will attend to different image regions

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

\( f_{\text{att}}(\cdot) \) is an MLP

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):
\[ e_{t, i, j} = f_{\text{att}}(h_{t-1}, z_{i, j}) \]
f_{\text{att}(.)} is an MLP

Extract spatial features from a pretrained CNN

Features:
H x W x D

Alignment scores:
H x W

Attention:
H x W

Normalize to get attention weights:
\[ a_{t, :, :} = \text{softmax}(e_{t, :, :}) \]
0 < a_{t, i, j} < 1, attention values sum to 1

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):
\[ e_{t,i,j} = f_{\text{att}} (h_{t-1}, z_{i,j}) \]
f_{\text{att}}(\cdot) is an MLP

Extract spatial features from a pretrained CNN

Extract spatial features from a pretrained CNN

Alignment scores: \( H \times W \)

Attention: \( H \times W \)

Normalize to get attention weights:
\[ a_{t,:,i} = \text{softmax} ( e_{t,:,i} ) \]
\[ 0 < a_{t,i,j} < 1, \text{ attention values sum to 1} \]

Compute context vector:
\[ c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j} \]

Q: How many context vectors are computed?

Image Captioning with RNNs and **Attention**

Each timestep of decoder uses a different context vector that looks at different parts of the input image:

\[
e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})
\]
\[
a_{t,:,:) = \text{softmax}(e_{t,:,:})
\]
\[
c_t = \sum_{ij} a_{t,i,j} z_{t,i,j}
\]

**Decoder**: \( y_t = g_V(y_{t-1}, h_{t-1}, c_t) \)

New context vector at every time step.


---

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 9 - 43

April 25, 2024
Image Captioning with RNNs and Attention

Extract spatial features from a pretrained CNN

Alignment scores: $H \times W$

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{t,i,j})$$
$$a_{t,:,i} = \text{softmax}(e_{t,:,i})$$

$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$

Attention: $H \times W$

Features: $H \times W \times D$

Decoder: $y_t = g_{y}(y_{t-1}, h_{t-1}, c_t)$

New context vector at every time step

Image Captioning with RNNs and Attention

Each timestep of decoder uses a different context vector that looks at different parts of the input image:

\[ e_{t,i,j} = f_{\text{att}}(h_{t-1}, z_{i,j}) \]
\[ a_{t,:} = \text{softmax}(e_{t,:}) \]
\[ c_t = \sum_{ij} a_{t,i,j} z_{t,i,j} \]

Extract spatial features from a pretrained CNN:

Features: \( H \times W \times D \)

Decoder:
\[ y_t = g_V(y_{t-1}, h_{t-1}, c_t) \]
New context vector at every time step

Image Captioning with RNNs and Attention

Each timestep of decoder uses a different context vector that looks at different parts of the input image:

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

\[ a_{t,:} = \text{softmax}(e_{t,:}) \]

\[ c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j} \]

Decoder: \( y_t = g_V(y_{t-1}, h_{t-1}, c_t) \)
New context vector at every time step

Extract spatial features from a pretrained CNN

Features: \( H \times W \times D \)

Image Captioning with RNNs and Attention

Each timestep of decoder uses a different context vector that looks at different parts of the input image:

\[ e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j}) \]
\[ a_{t,:} = \text{softmax}(e_{t,:}) \]
\[ c_t = \sum_{ij} a_{t,i,j} z_{t,i,j} \]

Extract spatial features from a pretrained CNN:

Features: \( H \times W \times D \)


Decoder: \( y_t = g_{\mathbf{v}}(y_{t-1}, h_{t-1}, c_t) \)

New context vector at every time step
Image Captioning with RNNs and Attention

Extract spatial features from a pretrained CNN

This entire process is differentiable.
- model chooses its own attention weights. No attention supervision is required


Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 9 - 48
April 25, 2024
Image Captioning with Attention

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.

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.
Image Captioning with RNNs and Attention

Attention we just saw in image captioning

<table>
<thead>
<tr>
<th>Features</th>
<th>$z_{0,0}$</th>
<th>$z_{0,1}$</th>
<th>$z_{0,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$z_{1,0}$</td>
<td>$z_{1,1}$</td>
<td>$z_{1,2}$</td>
</tr>
<tr>
<td></td>
<td>$z_{2,0}$</td>
<td>$z_{2,1}$</td>
<td>$z_{2,2}$</td>
</tr>
</tbody>
</table>

**Inputs:**
- Features: $z$ (shape: $H \times W \times D$)
- Query: $h$ (shape: $D$) $\leftarrow$ “query” refers to a vector used to calculate a corresponding context vector.
Attention we just saw in image captioning

Operations:
Alignment: $e_{i,j} = f_{\text{att}}(h, z_{i,j})$

Inputs:
Features: $z$ (shape: $H \times W \times D$)
Query: $h$ (shape: $D$)
Attention we just saw in image captioning

**Inputs:**
Features: $z$ (shape: $H \times W \times D$)
Query: $h$ (shape: $D$)

**Operations:**
Alignment: $e_{i,j} = f_{\text{att}}(h, z_{i,j})$
Attention: $a = \text{softmax}(e)$
Attention we just saw in image captioning

Inputs:
Features: \(z\) (shape: \(H \times W \times D\))
Query: \(h\) (shape: \(D\))

Outputs:
context vector: \(c\) (shape: \(D\))

Operations:
Alignment: \(e_{i,j} = f_{\text{att}}(h, z_{i,j})\)
Attention: \(a = \text{softmax}(e)\)
Output: \(c = \sum_{i,j} a_{i,j} z_{i,j}\)
Attention we just saw in image captioning

Inputs:
Features: \( z \) (shape: \( H \times W \times D \))
Query: \( h \) (shape: \( D \))

Operations:
Alignment: \( e_{i,j} = f_{\text{att}}(h, z_{i,j}) \)
Attention: \( a = \text{softmax}(e) \)
Output: \( c = \sum_{i,j} a_{i,j} z_{i,j} \)

Outputs:
context vector: \( c \) (shape: \( D \))

How is this different from the attention mechanism in transformers?

We’ll go into that next, any questions?
Attention operation is permutation invariant.
- Doesn't care about ordering of the features
- Stretch into $N = H \times W$ vectors

General attention layer – used in LLMs + beyond

Inputs:
- Input vectors: $x$ (shape: $N \times D$)
- Query: $h$ (shape: $D$)

Outputs:
- context vector: $c$ (shape: $D$)

Operations:
- Alignment: $e_i = f_{att}(h, x_i)$
- Attention: $a = \text{softmax}(e)$
- Output: $c = \sum_i a_i x_i$
Change $f_{\text{att}}(\cdot)$ to a dot product, this actually can work well in practice, but a simple dot product can have some issues...
Change $f_{\text{att}}(\cdot)$ to a scaled simple dot product
- Larger dimensions means more terms in the dot product sum.
- So, the variance of the logits is higher. Large magnitude vectors will produce much higher logits.
- So, the post-softmax distribution has lower-entropy, assuming logits are IID.
- Ultimately, these large magnitude vectors will cause softmax to peak and assign very little weight to all others
- Divide by $\sqrt{D}$ to reduce effect of large magnitude vectors
- Similar to Xavier and Kaiming Initialization!

**Inputs:**
Input vectors: $\mathbf{x}$ (shape: $N \times D$)
Query: $\mathbf{h}$ (shape: $D$)

**Outputs:**
context vector: $\mathbf{c}$ (shape: $D$)

**Operations:**
- Alignment: $e_i = h \cdot x_i / \sqrt{D}$
- Attention: $a = \text{softmax}(e)$
- Output: $\mathbf{c} = \sum_i a_i x_i$
General attention layer

**Inputs:**
Input vectors: \( \mathbf{x} \) (shape: \( N \times D \))

Queries: \( \mathbf{q} \) (shape: \( M \times D \))

**Outputs:**
context vectors: \( \mathbf{y} \) (shape: \( D \))

**Operations:**
Alignment: \( e_{i,j} = q_j \cdot x_i / \sqrt{D} \)
Attention: \( a = \text{softmax}(e) \)
Output: \( y_j = \sum_i a_{i,j} x_i \)

Multiple query vectors
- each query creates a new, corresponding output context vector

Allows us to compute multiple attention context vectors at once
Will go into more details in future slides, but this allows us to compute context vectors for multiple timesteps in parallel
General attention layer

Inputs:
- Input vectors: \( x \) (shape: \( N \times D \))
- Queries: \( q \) (shape: \( M \times D \))

Operations:
- Alignment: \( e_{i,j} = q_j \cdot x_i / \sqrt{D} \)
- Attention: \( a = \text{softmax}(e) \)
- Output: \( y_j = \sum_i a_{i,j} x_i \)

Outputs:
- Context vectors: \( y \) (shape: \( D \))

Notice that the input vectors are used for both the alignment as well as the attention calculations.
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.
Notice that the input vectors are used for both the alignment as well as the attention calculations.
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

**Inputs:**
- Input vectors: \( x \) (shape: \( N \times D \))
- Queries: \( q \) (shape: \( M \times D_k \))

**Operations:**
- Key vectors: \( k = xW_k \)
- Value vectors: \( v = xW_v \)
General attention layer

Inputs:
- Input vectors: \( x \) (shape: \( N \times D \))
- Queries: \( q \) (shape: \( M \times D_k \))

Operations:
- Key vectors: \( k = xW_k \)
- Value vectors: \( v = xW_v \)
- Alignment: \( e_{i,j} = q_j \cdot k_i / \sqrt{D} \)
- Attention: \( a = \text{softmax}(e) \)
- Output: \( y_j = \sum_i a_{i,j} v_i \)

Outputs:
- Context vectors: \( y \) (shape: \( D_v \))

The input and output dimensions can now change depending on the key and value FC layers.

Since the alignment scores are just scalars, the value vectors can be any dimension we want.
Recall that the query vector was a function of the input vectors

General attention layer

Operations:
- Key vectors: \( k = xW_k \)
- Value vectors: \( v = xW_v \)
- Alignment: \( e_{i,j} = q_j \cdot k_i / \sqrt{D} \)
- Attention: \( a = \text{softmax}(e) \)
- Output: \( y_j = \sum_i a_{i,j} v_i \)

Inputs:
- Input vectors: \( x \) (shape: \( N \times D \))
- Queries: \( q \) (shape: \( M \times D_k \))

Outputs:
- context vectors: \( y \) (shape: \( D_v \))

Encoder: \( h_0 = f_W(z) \)
where \( z \) is spatial CNN features
\( f_W(.) \) is an MLP
Lecture 8:
Video Lecture Supplement
Attention and Transformers
Next: The Self-attention Layer

Idea: leverages the strengths of attention layers without the need for separate query vectors.

Inputs:
Input vectors: $x$ (shape: $N \times D$)
Queries: $q$ (shape: $M \times D_k$)

Outputs:
context vectors: $y$ (shape: $D_v$)

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$
Self attention layer

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: $x$ (shape: $N \times D$)
Queries: $q$ (shape: $M \times D_k$)

We can calculate the query vectors from the input vectors, therefore, defining a "self-attention" layer.

Instead, query vectors are calculated using a FC layer.

No input query vectors anymore
Self attention layer

Inputs:
- Input vectors: \( x \) (shape: \( N \times D \))

Operations:
- Key vectors: \( k = xW_k \)
- Value vectors: \( v = xW_v \)
- Query vectors: \( q = xW_q \)
- Alignment: \( e_{i,j} = q_i \cdot k_j / \sqrt{D} \)
- Attention: \( a = \text{softmax}(e) \)
- Output: \( y_j = \sum_i a_{i,j} v_i \)

Outputs:
- Context vectors: \( y \) (shape: \( D_v \))

\[
\begin{align*}
\text{Inputs:} & \quad x_0, x_1, x_2 \\
\text{Key vectors:} & \quad k_0, k_1, k_2 \\
\text{Value vectors:} & \quad v_0, v_1, v_2 \\
\text{Query vectors:} & \quad q_0, q_1, q_2 \\
\text{Alignment:} & \quad e_{0,0}, e_{0,1}, e_{0,2}, e_{1,0}, e_{1,1}, e_{1,2}, e_{2,0}, e_{2,1}, e_{2,2} \\
\text{Attention:} & \quad a_{0,0}, a_{0,1}, a_{0,2}, a_{1,0}, a_{1,1}, a_{1,2}, a_{2,0}, a_{2,1}, a_{2,2} \\
\text{Output:} & \quad y_0, y_1, y_2
\end{align*}
\]
**Self attention layer** - attends over sets of inputs

### Inputs:
- Input vectors: \( x \) (shape: \( N \times D \))

### Operations:
- Key vectors: \( k = xW_k \)
- Value vectors: \( v = xW_v \)
- Query vectors: \( q = xW_q \)
- Alignment: \( e_{i,j} = q_i \cdot k_j / \sqrt{D} \)
- Attention: \( a = \text{softmax}(e) \)
- Output: \( y_j = \sum_i a_{i,j} v_i \)

### Outputs:
- Context vectors: \( y \) (shape: \( D_v \))

### Diagram:
- Self-attention layer diagram showing the flow of input, key, value, and query vectors, along with the alignment and attention operations.
Self attention layer - attends over sets of inputs

Permutation equivariant

Self-attention layer doesn’t care about the orders of the inputs!

**Problem:** How can we encode ordered sequences like language or spatially ordered image features?
Positional encoding

Concatenate or add special positional encoding \( p_j \) to each input vector \( x_j \)

We use a function \( pos: \mathbb{N} \rightarrow \mathbb{R}^d \) to process the position \( j \) of the vector into a \( d \)-dimensional vector

So, \( p_j = pos(j) \)

Possible desirable properties of \( pos(.) \):

1. It should output a unique encoding for each time-step (word's position in a sentence)
2. Distance between any two time-steps should be consistent across sentences with different lengths.
3. Our model should generalize to longer sentences without any efforts. Its values should be bounded.
4. It must be deterministic.
Positional encoding

Options for \( \text{pos}(\cdot) \)

1. Learn a lookup table:
   - Learn parameters to use for \( \text{pos}(t) \) for \( t \in [0, T) \)
   - Lookup table contains \( T \times d \) parameters.

Possible desirable properties of \( \text{pos}(\cdot) \):

1. It should output a \textbf{unique} encoding for each time-step (word’s position in a sentence)
2. \textbf{Distance} between any two time-steps should be consistent across sentences with different lengths.
3. Our model should generalize to longer sentences without any efforts. Its values should be bounded.
4. It must be \textbf{deterministic}.

Vaswani et al, “Attention is all you need”, NeurIPS 2017
Positional encoding

Options for $\text{pos}(.)$

1. Learn a lookup table:
   - Learn parameters to use for $\text{pos}(t)$ for $t \in [0, T)$
   - Lookup table contains $T \times d$ parameters.

2. Design a fixed function with the desired properties

We use a function $\text{pos}: \mathbb{N} \rightarrow \mathbb{R}^d$ to process the position $j$ of the vector into a $d$-dimensional vector

So, $p_j = \text{pos}(j)$

Vaswani et al, “Attention is all you need”, NeurIPS 2017

$$p(t) = \begin{bmatrix} \sin(\omega_1 t) \\ \cos(\omega_1 t) \\ \sin(\omega_2 t) \\ \cos(\omega_2 t) \\ \vdots \\ \sin(\omega_{d/2} t) \\ \cos(\omega_{d/2} t) \end{bmatrix}_{d \times 1} \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$
Positional encoding

Options for $pos(\cdot)$

1. Learn a lookup table:
   - Learn parameters to use for $pos(t)$ for $t \in [0, T)$
   - Lookup table contains $T \times d$ parameters.

2. Design a fixed function with the desired properties

Concatenate special positional encoding $p_j$ to each input vector $x_j$

We use a function $pos: \mathbb{N} \rightarrow \mathbb{R}^d$
to process the position $j$ of the vector into a $d$-dimensional vector

So, $p_j = pos(j)$

Intuition:

$$p(t) = \begin{bmatrix}
\sin(\omega_1 t) \\
\cos(\omega_1 t) \\
\sin(\omega_2 t) \\
\cos(\omega_2 t) \\
\vdots \\
\sin(\omega_{d/2} t) \\
\cos(\omega_{d/2} t)
\end{bmatrix}_{d}$$

where $\omega_k = \frac{1}{10000^{2k/d}}$
Masked self-attention layer

Inputs:
Input vectors: \( x \) (shape: \( N \times D \))

Operations:
Key vectors: \( k = xW_k \)
Value vectors: \( v = xW_v \)
Query vectors: \( q = xW_q \)
Alignment: \( e_{i,j} = q_i \cdot k_j / \sqrt{D} \)
Attention: \( a = \text{softmax}(e) \)
Output: \( y_j = \sum_i a_{i,j} v_i \)

Outputs:
context vectors: \( y \) (shape: \( D_v \))

- Allows us to parallelize attention across time
- Don’t need to calculate the context vectors from the previous timestep first!
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to \( -\infty \) (-nan)
Multi-head self-attention layer
- Multiple self-attention “heads” in parallel

Q: Why do this?
Multi-head self-attention layer

- Multiple self-attention “heads” in parallel

A: We may want to have multiple sets of queries/keys/values calculated in the layer. This is a similar idea to having multiple conv filters learned in a layer.
General attention versus self-attention

Transformer models rely on many, stacked self-attention layers
Comparing RNNs to Transformer

**RNNs**

(+) LSTMs work reasonably well for long sequences.
(-) Expects an ordered sequences of inputs
(-) Sequential computation: subsequent hidden states can only be computed after the previous ones are done.

**Transformer:**

(+) Good at long sequences. Each attention calculation looks at all inputs.
(+) Can operate over unordered sets or ordered sequences with positional encodings.
(+) Parallel computation: All alignment and attention scores for all inputs can be done in parallel.
(-) Requires a lot of memory: N x M alignment and attention scalers need to be calculated and stored for a single self-attention head. (but GPUs are getting bigger and better)
Attention Is All You Need

“ImageNet Moment for Natural Language Processing”

Pretraining:
Download a lot of text from the internet

Train a giant Transformer model for language modeling

Finetuning:
Fine-tune the Transformer on your own NLP task

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@gmail.com

Illia Polosukhin* †
illia.polosukhin@gmail.com
On the Opportunities and Risks of Foundation Models


Center for Research on Foundation Models (CRFM)
Stanford Institute for Human-Centered Artificial Intelligence (HAI)
Stanford University
Image Captioning using **Transformers**

**Input:** Image $I$

**Output:** Sequence $y = y_1, y_2, \ldots, y_T$

Extract spatial features from a pretrained CNN

Features:

$H \times W \times D$

<table>
<thead>
<tr>
<th>$Z_{0,0}$</th>
<th>$Z_{0,1}$</th>
<th>$Z_{0,2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z_{1,0}$</td>
<td>$Z_{1,1}$</td>
<td>$Z_{1,2}$</td>
</tr>
<tr>
<td>$Z_{2,0}$</td>
<td>$Z_{2,1}$</td>
<td>$Z_{2,2}$</td>
</tr>
</tbody>
</table>
Image Captioning using Transformers

**Input:** Image \( I \)

**Output:** Sequence \( y = y_1, y_2, \ldots, y_T \)

**Encoder:** \( c = T_w(z) \)
where \( z \) is spatial CNN features
\( T_w(.) \) is the transformer encoder

Extract spatial features from a pretrained CNN

<table>
<thead>
<tr>
<th>Features: ( H \times W \times D )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Z_{0,0} )</td>
</tr>
<tr>
<td>( Z_{1,0} )</td>
</tr>
<tr>
<td>( Z_{2,0} )</td>
</tr>
</tbody>
</table>

Transformer encoder

\( c_{0,0} \) | \( c_{0,1} \) | \( c_{0,2} \) | \ldots | \( c_{2,2} \)
Image Captioning using Transformers

**Input:** Image I

**Output:** Sequence \( y = y_1, y_2, \ldots, y_T \)

**Encoder:** \( c = T_W(z) \)

where \( z \) is spatial CNN features

\( T_W(\cdot) \) is the transformer encoder

\[ Z_{0,0} \quad Z_{0,1} \quad Z_{0,2} \]
\[ \begin{array}{ccc} Z_{1,0} & Z_{1,1} & Z_{1,2} \\ Z_{2,0} & Z_{2,1} & Z_{2,2} \end{array} \]

**Transformer encoder**

\[ c_{0,0} \quad c_{0,1} \quad c_{0,2} \quad \ldots \quad c_{2,2} \]

**Transformer decoder**

\[ y_t = T_D(y_{0:t-1}, c) \]

where \( T_D(\cdot) \) is the transformer decoder

\[ \begin{array}{cccc} y_1 \quad y_2 \quad y_3 \quad y_4 \end{array} \]

Extract spatial features from a pretrained CNN

Features: \( H \times W \times D \)
The Transformer encoder block

Made up of N encoder blocks.

In vaswani et al. N = 6, D_q = 512

Vaswani et al, “Attention is all you need”, NeurIPS 2017
The Transformer encoder block

Let's dive into one encoder block

Vaswani et al., "Attention is all you need", NeurIPS 2017
The Transformer encoder block

Transformer encoder

Positional encoding

\[ z_{0,0}, z_{0,1}, z_{0,2}, \ldots, z_{2,2} \]

Multi-head self-attention

\[ x_0, x_1, x_2, x_2 \]

Attention attends over all the vectors

Vaswani et al., "Attention is all you need", NeurIPS 2017
The Transformer encoder block

\[ z_{0,0}, z_{0,1}, z_{0,2}, \ldots, z_{2,2} \]

Transformer encoder

\[ \vdots \times N \]

Positional encoding

\[ z_{0,0}, z_{0,1}, z_{0,2}, \ldots, z_{2,2} \]

Multi-head self-attention

\[ x_0, x_1, x_2, x_2 \]

Residual connection

Attention attends over all the vectors

Vaswani et al., “Attention is all you need”, NeurIPS 2017
The Transformer encoder block

Transformer encoder block diagram:

- Positional encoding
- Multi-head self-attention
- Layer norm
- Residual connection

Attention attends over all the vectors.

LayerNorm over each vector individually.

Vaswani et al., "Attention is all you need", NeurIPS 2017
The Transformer encoder block

\[ \begin{array}{c}
\vdots \quad x \quad N \\
\vdots
\end{array} \]

Transformer encoder

Positional encoding

\[ \begin{array}{c}
\vdots \quad x \quad N \\
\vdots
\end{array} \]

\[ \begin{array}{c}
\vdots \quad x \quad N \\
\vdots
\end{array} \]

\[ \begin{array}{c}
\vdots \quad x \quad N \\
\vdots
\end{array} \]

MLP over each vector individually

LayerNorm over each vector individually

Residual connection

Attention attends over all the vectors

Vaswani et al, "Attention is all you need", NeurIPS 2017
The Transformer encoder block

Transformer encoder

\[ z_{0,0}, z_{0,1}, z_{0,2}, \ldots, z_{2,2} \]

Residual connection

MLP over each vector individually

LayerNorm over each vector individually

Residual connection

Attention attends over all the vectors

Positional encoding

Vector nesting

\[ : \times N \]

Layer norm

Multi-head self-attention

MLP

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 9 - 90

April 25, 2024

Vaswani et al, “Attention is all you need”, NeurIPS 2017
The Transformer encoder block

Positional encoding

LayerNorm over each vector individually
Residual connection
MLP over each vector individually
LayerNorm over each vector individually
Residual connection
Attention attends over all the vectors

Vaswani et al., "Attention is all you need", NeurIPS 2017
The Transformer encoder block

Transformer Encoder Block:

**Inputs**: Set of vectors \( x \)

**Outputs**: Set of vectors \( y \)

Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

---

Vaswani et al, "Attention is all you need", NeurIPS 2017
The Transformer decoder

Made up of N decoder blocks.

In vaswani et al. N = 6, D_q = 512

Vaswani et al, “Attention is all you need”, NeurIPS 2017
Let's dive into the transformer decoder block.

Vaswani et al, "Attention is all you need", NeurIPS 2017
The Transformer decoder block

Most of the network is the same as the transformer encoder.

Ensures we only look at the previous tokens (teacher forcing during training).

Vaswani et al., "Attention is all you need", NeurIPS 2017
Multi-head attention block attends over the transformer encoder outputs.

For image captioning, this is how we inject image features into the decoder.

Vaswani et al, “Attention is all you need”, NeurIPS 2017
The Transformer decoder block

Transformer Decoder Block:

**Inputs:** Set of vectors $x$ and Set of context vectors $c$.

**Outputs:** Set of vectors $y$.

Masked Self-attention only interacts with past inputs.

Multi-head attention block is NOT self-attention. It attends over encoder outputs.

Highly scalable, highly parallelizable, but high memory usage.

Vaswani et al., "Attention is all you need", NeurIPS 2017
Image Captioning using transformers

- No recurrence at all

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

Transformer encoder

Transformer decoder

person wearing hat [END]

$y_0$ $y_1$ $y_2$ $y_3$ $y_4$
Image Captioning using transformers

- Perhaps we don't need convolutions at all?

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

Transformer encoder

Transformer decoder

[START] person wearing hat [END]

$y_0 \ y_1 \ y_2 \ y_3 \ y_4$

$z_{0,0} \ z_{0,1} \ z_{0,2} \ ...

z_{2,0} \ z_{2,1} \ z_{2,2}$

$c_{0,0} \ c_{0,1} \ c_{0,2} \ ...

c_{2,2}$

$y_0 \ y_1 \ y_2 \ y_3$

person wearing hat

[START] person wearing hat

$y_0 \ y_1 \ y_2 \ y_3 \ y_4$
Image Captioning using ONLY transformers

- Transformers from pixels to language

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020
Colab link to an implementation of vision transformers
ViTs – Vision Transformers

Figure from: Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020
Vision Transformers vs. ResNets

Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ArXiv 2020

Colab link to an implementation of vision transformers
Vision Transformers

Fan et al., “Multiscale Vision Transformers”, ICCV 2021

Carion et al., “End-to-End Object Detection with Transformers”, ECCV 2020

ConvNets strike back!

A ConvNet for the 2020s. Liu et al. CVPR 2022

Fei-Fei Li, Ehsan Adeli, Zane Durante
DeiT III: Revenge of the ViT

Hugo Touvron*,†  Matthieu Cord†  Hervé Jégou*

**Graphs: ImageNet-1k and ImageNet-21k**

- **ImageNet-1k**
  - Ours: ViT
  - ConvNeXt
  - DeiT: ViT
  - EfficientNet
  - EfficientNet-V2
  - RSB: ResNet
  - Swin

- **ImageNet-21k**
  - Ours: ViT
  - ConvNeXt
  - EfficientNet-V2
  - Swin
Summary

- Adding **attention** to RNNs allows them to "attend" to different parts of the input at every time step
- The **general attention layer** is a new type of layer that can be used to design new neural network architectures
- **Transformers** are a type of layer that uses **self-attention** and layer norm.
  - It is highly **scalable** and highly **parallelizable**
  - **Faster** training, **larger** models, **better** performance across vision and language tasks
  - They are quickly replacing RNNs, LSTMs, and may(?) even replace convolutions.
Next time: Object Detection + Segmentation
Appendix Slides from Previous Years
Image Captioning with Attention

Soft attention

Hard attention (requires reinforcement learning)

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.
Example: CNN with Self-Attention

- Input Image
- CNN
- Features: C x H x W

Zhang et al., "Self-Attention Generative Adversarial Networks", ICML 2018

Slide credit: Justin Johnson
Example: CNN with Self-Attention

CNN

Input Image

Features: $C \times H \times W$

1x1 Conv

Queries: $C' \times H \times W$

1x1 Conv

Keys: $C' \times H \times W$

1x1 Conv

Values: $C' \times H \times W$

1x1 Conv

Zhang et al., “Self-Attention Generative Adversarial Networks”, ICML 2018

Slide credit: Justin Johnson
Example: CNN with Self-Attention

Zhang et al., "Self-Attention Generative Adversarial Networks", ICML 2018

Slide credit: Justin Johnson
Example: CNN with Self-Attention

Zhang et al., “Self-Attention Generative Adversarial Networks”, ICML 2018

Slide credit: Justin Johnson
Example: CNN with Self-Attention

CNN

Input Image

Features: C x H x W

1x1 Conv

1x1 Conv

Queries: C' x H x W

Keys: C' x H x W

Values: C' x H x W

Attention Weights (H x W) x (H x W)

Transpose

softmax

C x H x W

C' x H x W

1x1 Conv

X

X

Zhang et al., “Self-Attention Generative Adversarial Networks”, ICML 2018

Slide credit: Justin Johnson
Example: CNN with Self-Attention

Features: \( C \times H \times W \)

Queries: \( C' \times H \times W \)

Keys: \( C \times H \times W \)

Values: \( C' \times H \times W \)

Transpose

Attention Weights: \( (H \times W) \times (H \times W) \)

Self-Attention Module

Residual Connection

1x1 Conv

softmax

\( C \times H \times W \)

\( C' \times H \times W \)

1x1 Conv

Zhang et al., “Self-Attention Generative Adversarial Networks”, ICML 2018

Slide credit: Justin Johnson

Fei-Fei Li, Ehsan Adeli, Zane Durante

Lecture 9 - 115

April 25, 2024