Lecture 9: Attention and Transformers
Administrative

- Assignment 1 regrade requests: submit on Gradescope by EOD today (05/02)
- Assignment 2 due 05/08

- Please read pinned post on Ed regarding midterm logistics. Sample midterm will be posted on Ed by tomorrow.
  - If you need a special accommodation for the midterm and have not contacted us or been contacted about your accommodation yet, please let us know ASAP. We will be contacting SCPD students who asked to take the exam on campus soon.

- Please read Ed post regarding late day clarifications

- AWS credits should have been distributed to your accounts you submitted in the survey
Last Time: Recurrent Neural Networks
Last Time: Variable length computation graph with shared weights

\[ y_1 \xrightarrow{L_1} y_2 \xrightarrow{L_2} y_3 \xrightarrow{L_3} \ldots \xrightarrow{L_T} L \]

\[ h_0 \xrightarrow{W} f_W h_1 \xrightarrow{f_W} h_2 \xrightarrow{f_W} h_3 \xrightarrow{\ldots} h_T \]

\[ x_1 \quad x_2 \quad x_3 \]
Sequence to Sequence with RNNs

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

**Output:** Sequence $y_1, \ldots, y_{T'}$

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

---

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

From final hidden state predict:

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

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 $c = h_T$)

---

**Input:**
- we
- are
- eating
- bread

**Output:**
- estamos
- comiendo

---

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 $c = h_T$)

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 $c=h_T$)

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

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 $c= h_T$)

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

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

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\textsubscript{att} is an MLP)

From final hidden state:

Initial decoder state \( s_0 \)

\( e_{11} \)
\( h_1 \)
\( x_1 \)
we

\( e_{12} \)
\( h_2 \)
\( x_2 \)
are

\( e_{13} \)
\( h_3 \)
\( x_3 \)
eating

\( e_{14} \)
\( h_4 \)
\( x_4 \)
bread

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
\[ e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \quad (f_{\text{att}} \text{ is an MLP}) \]

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 \( s_0 \)

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

From final hidden state:
Initial decoder state $s_0$

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

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

Compute context vector as linear combination 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**

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

Intuition: Context vector attends to the relevant part of the input sequence

- "estamos" = "we are"
- so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

Compute (scalar) alignment scores

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

Normalize alignment scores to get **attention weights**

$$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$$

Compute context vector as linear combination of hidden states

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

Use context vector in decoder:

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

---

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

Fei-Fei Li, Yunzhu Li, Ruohan Gao  

Lecture 9 - 16  

May 02, 2023
Sequence to Sequence with RNNs and **Attention**

**Intuition**: Context vector attends to the relevant part of the input sequence. "estamos" = "we are" so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$.

- Normalize alignment scores to get **attention weights**: $0 < a_{t,i} < 1$; $\sum_i a_{t,i} = 1$.
- Compute context vector as linear combination of hidden states: $c_t = \sum_i a_{t,i} h_i$.
- 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 $c_2$

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 $c_2$

Use $c_2$ to compute $s_2$, $y_2$

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 “comiendo” = “eating” so maybe $a_{21}=a_{24}=0.05$, $a_{22}=0.1$, $a_{23}=0.8$.

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

Use $c_2$ to compute $s_2$, $y_2$.

---

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
**Example**: English to French translation

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

**Output**: “L’accord sur la zone économique européenne a été signé en août 1992.”
**Example**: English to French translation

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

**Output**: “L’accord sur la zone économique européenne a été signé en août 1992.”

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.”

**Output**: “L’accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order

Attention figures out different word orders
Sequence to Sequence with RNNs and Attention

The decoder doesn’t use the fact that $h_i$ form an ordered sequence – it just treats them as an unordered set \{h\}_i\}

Can use similar architecture given any set of input hidden vectors \{h_j\}!

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Image Captioning using spatial features

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

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

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  
$f_w(.)$ is an MLP

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

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:**  
\[ y_t = g_V(y_{t-1}, h_{t-1}, c) \]  
where context vector \( c \) is often \( c = h_0 \)

---

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

**Features:** $H \times W \times D$

**Decoder:** $y_t = g_v(y_{t-1}, h_{t-1}, c)$  
where context vector $c$ is often $c = h_0$

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

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

---


---

Fei-Fei Li, Yunzhu Li, Ruohan Gao

Lecture 9 - 31 May 02, 2023
Image Captioning using spatial features

Problem: 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

CNN
MLP

Fei-Fei Li, Yunzhu Li, Ruohan Gao

Lecture 9 - 32 May 02, 2023
Image Captioning with RNNs and Attention

Attention idea: New context vector at every time step.

Each context vector will attend to different image regions

Extract spatial features from a pretrained CNN

Features:
H x W x D

Image Captioning with RNNs and Attention

Extract spatial features from a pretrained CNN

Compute alignments scores (scalars):

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

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

Features:

H x W x D

Alignment scores:

H x W

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

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

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:
\[ H \times W \times D \]

Alignments scores:
\[ H \times W \]

Attention:
\[ H \times W \]

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

\( 0 < a_{t,i,j} < 1 \), attention values sum to 1

Image Captioning with RNNs and Attention

Extract spatial features from a pretrained CNN

Alignment scores: \( H \times W \)

Attention: \( H \times W \)

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

where \( f_{\text{att}}(.) \) is an MLP

Normalization to get attention weights:
\[
a_{t,:,:} = \text{softmax}(e_{t,:,:})
\]

0 < \( a_{t,i,j} \) < 1, attention values sum to 1

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

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, Yunzhu Li, Ruohan Gao

Lecture 9 - 37 May 02, 2023
Image Captioning with RNNs and Attention

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

CNN

Align. scores: $H \times W$

$e_{t,i,j} = f_{att}(h_{t-1}, z_{t,i,j})$

$A_{t,:,:} = \text{softmax}(e_{t,:,:})$

$C_t = \sum_{i,j} A_{t,i,j} z_{t,i,j}$

Attention: $H \times W$

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

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.

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

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

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.

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

Extract spatial features from a pretrained CNN.

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.

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

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

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

Extract spatial features from a pretrained CNN

Features: H x W x D

Extract spatial features from a pretrained CNN

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

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.
Image Captioning with Attention

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Benchio, 2015. Reproduced with permission.
Image Captioning with RNNs and Attention

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

Alignment scores: $H \times W$

Attention: $H \times W$

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

Attention we just saw in image captioning

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

**Inputs:**
- Features: $\mathbf{z}$ (shape: $H \times W \times D$)
- Query: $\mathbf{h}$ (shape: $D$)
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: \( c = \sum_{i,j} a_{i,j} z_{i,j} \)

**Diagram:**
- Features: \( z_{0,0}, z_{0,1}, z_{0,2}, z_{1,0}, z_{1,1}, z_{1,2}, z_{2,0}, z_{2,1}, z_{2,2} \)
- Query: \( h \)
- Attention: \( e_{0,0}, e_{0,1}, e_{0,2}, e_{1,0}, e_{1,1}, e_{1,2}, e_{2,0}, e_{2,1}, e_{2,2} \)
- Context vector: \( c \)
- Operations: \( \text{softmax}, \text{mul + add} \)
Attention operation is permutation invariant.
- Doesn't care about ordering of the features
- Stretch H x W = N into N vectors

General attention layer

Operations:
Alignment: $e_i = f_{\text{att}}(h, x_i)$
Attention: $a = \text{softmax}(e)$
Output: $c = \sum_i a_i x_i$

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

Inputs:
Input vectors: $x$ (shape: N x D)
Query: $h$ (shape: D)
General attention layer

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

**Operations:**
- Alignment: \( e_i = h \cdot x_i \)
- Attention: \( a = \text{softmax}(e) \)
- Output: \( c = \sum_i a_i x_i \)

Change \( f_{\text{att}}(.) \) to a simple dot product
- only works well with key & value transformation trick (will mention in a few slides)

**Inputs:**
Input vectors: \( x \) (shape: N x D)
Query: \( h \) (shape: D)
Change $f_{att}(.)$ 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

**General attention layer**

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

**Outputs:**
- Context vector: $c$ (shape: D)

**Operations:**
- Alignment: $e_i = h \cdot x_i / \sqrt{D}$
- Attention: $a = \text{softmax}(e)$
- Output: $c = \sum_i a_i x_i$
Multiple query vectors
- each query creates a new output context vector

**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: $D$)

Multiple query vectors
General attention layer

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.
General attention layer

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

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

Outputs:
context vectors: \( \mathbf{y} \) (shape: \( D_{v} \))

Operations:
Key vectors: \( \mathbf{k} = \mathbf{xW}_{k} \)
Value vectors: \( \mathbf{v} = \mathbf{xW}_{v} \)
Alignment: \( e_{i,j} = q_{j} \cdot k_{i} / \sqrt{D} \)
Attention: \( a = \text{softmax}(e) \)
Output: \( y_{j} = \sum a_{i,j} v_{i} \)
General attention layer

Recall that the query vector was a function of the input vectors

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: \( y_j = \sum_i a_{i,j} v_i \)

Encoder: \( h_0 = f_w(z) \)
where \( z \) is spatial CNN features
\( f_w(\cdot) \) is an MLP
Self attention layer

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

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

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

---

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: $e_{ij} = q_i \cdot k_j / \sqrt{D}$
- Attention: $a = \text{softmax}(e)$
- Output: $y_j = \sum_i a_{ij} v_i$

**Outputs:**
Context vectors: $y$ (shape: $D_v$)

**Diagram:**
- $x$: Input vectors
- $k$: Key vectors
- $v$: Value vectors
- $q$: Query vectors
- $a$: Attention
- $e$: Alignment
- $y$: Output

Fei-Fei Li, Yunzhu Li, Ruohan Gao
**Self attention layer** - attends over sets of inputs

### Inputs:
- Input vectors: $\mathbf{x}$ (shape: $N \times D$)

### Operations:
- Key vectors: $\mathbf{k} = \mathbf{xW}_k$
- Value vectors: $\mathbf{v} = \mathbf{xW}_v$
- Query vectors: $\mathbf{q} = \mathbf{xW}_q$
- Alignment: $e_{ij} = q_i \cdot k_j / \sqrt{D}$
- Attention: $a = \text{softmax}(e)$
- Output: $y_j = \sum_i a_{ij} v_i$

### Outputs:
- Context vectors: $\mathbf{y}$ (shape: $D_v$)

### Diagram:
- Self-attention layer diagram showing inputs, attention mechanism, and outputs.
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

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, \( p_j = pos(j) \)

Desiderata 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 \( pos(.) \)

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

Desiderata 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**.

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, \( p_j = pos(j) \)

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

Options for \( pos(.) \)

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 desiderata

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

where \( \omega_k = \frac{1}{10000^{2k/d}} \)

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

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

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

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

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

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

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

Intuition:

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 8 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 9 & 1 & 0 & 0 & 1 \\
2 & 0 & 1 & 0 & 10 & 1 & 0 & 1 & 0 \\
3 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
5 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
6 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
7 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

where \( \omega_k = \frac{1}{10000^{2k/d}} \)

Vaswani et al, "Attention is all you need", NeurIPS 2017
Masked self-attention layer

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

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

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

- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity
Multi-head self-attention layer
- Multiple self-attention heads in parallel
General attention versus self-attention
Example: CNN with Self-Attention
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
Example: CNN with Self-Attention

Input Image

Features: $C \times H \times W$

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

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

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

1x1 Conv

1x1 Conv

1x1 Conv

Attention Weights

$(H \times W) \times (H \times W)$

Transpose

softmax

$C' \times H \times W$

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

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

- **Input Image**
- **CNN**
- **Features**: $C \times H \times W$
- **Queries**: $C' \times H \times W$
- **Keys**: $C' \times H \times W$
- **Values**: $C' \times H \times W$
- **1x1 Conv**
- **Transpose**
- **softmax**
- **Attention Weights**: $(H \times W) \times (H \times W)$
- **$X$**
- **$C \times H \times H$**
- **$C' \times H \times W$**
- **$1x1 Conv$**

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
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: \(N \times M\) 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@google.com

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

Rishi Bommasani  Drew A. Hudson  Ehsan Adeli  Russ Altman  Simran Arora
Sydney von Arx  Michael S. Bernstein  Jeannette Bohg  Antoine Bosselut  Emma Brunskill
Erik Brynjolfsson  Shyamal Buch  Dallas Card  Rodrigo Castellon  Niladri Chatterji
Annie Chen  Kathleen Creel  Jared Quincy Davis  Dorottya Demszyk  Chris Donahue
Moussa Doumbouya  Esin Durmus  Stefano Ermon  John Etchemendy  Kavin Ethayarajh
Li Fei-Fei  Chelsea Finn  Trevor Gale  Lauren Gillespie  Karan Goel  Noah Goodman
Shelby Grossman  Neel Guha  Tatsunori Hashimoto  Peter Henderson  John Hewitt
Daniel E. Ho  Jenny Hong  Kyle Hsu  Jing Huang  Thomas Icard  Saahil Jain
Dan Jurafsky  Pratyusha Kalluri  Siddharth Karamcheti  Geoff Keeling  Fereshte Khani
Omar Khattab  Pang Wei Koh  Mark Krass  Ranjay Krishna  Rohith Kuditipudi
Ananya Kumar  Faisal Ladhak  Mina Lee  Tony Lee  Jure Leskovec  Isabelle Levent
Xiang Lisa Li  Xuechen Li  Tengyu Ma  Ali Malik  Christopher D. Manning
Suvir Mirchandani  Eric Mitchell  Zanele Munyikwa  Suraj Nair  Avaniya Narayan
Deepak Narayanan  Ben Newman  Allen Nie  Juan Carlos Niebles  Hamed Nilforoshan
Julian Nyarko  Giray Ogut  Laurel Orr  Isabel Papadimitriou  Joon Sung Park  Chris Piech
Eva Portelance  Christopher Potts  Aditi Raghunathan  Rob Reich  Hongyu Ren
Frieda Rong  Yusuf Roohani  Camilo Ruiz  Jack Ryan  Christopher Ré  Dorsa Sadigh
Shiori Sagawa  Keshav Santhanam  Andy Shih  Krishnan Srinivasan  Alex Tamkin
Rohan Taori  Armin W. Thomas  Florian Tramèr  Rose E. Wang  William Wang  Bohan Wu
Jiajun Wu  Yuhuai Wu  Sang Michael Xie  Michihiro Yasunaga  Jiaxuan You  Matei Zaharia
Michael Zhang  Tianyi Zhang  Xikun Zhang  Yuhui Zhang  Lucia Zheng  Kaitlyn Zhou
Percy Liang*

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: $H \times W \times D$
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

Extract spatial features from a pretrained CNN

Features: \( H \times W \times D \)
Image Captioning using Transformers

**Input:** Image I

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

**Encoder:** $c = T_w(z)$

where $z$ is spatial CNN features

$T_w(\cdot)$ is the transformer encoder

**Decoder:** $y_t = T_D(y_{0:t-1}, c)$

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

---

**Features:** $H \times W \times D$

**Transformers:**

- **Transformer encoder**
  - $z_{0,0}$, $z_{0,1}$, $z_{0,2}$
  - $z_{1,0}$, $z_{1,1}$, $z_{1,2}$
  - $z_{2,0}$, $z_{2,1}$, $z_{2,2}$

- **Transformer decoder**
  - $y_0$, $y_1$, $y_2$, $y_3$
  - $y_1$, $y_2$, $y_3$, $y_4$

---

**Image:**

- Extract spatial features from a pretrained CNN

---

Fei-Fei Li, Yunzhu Li, Ruohan Gao

Lecture 9 - 80 May 02, 2023
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

Transformer encoder

$c_{0,0} \quad c_{0,1} \quad c_{0,2} \quad \ldots \quad c_{2,2}$

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

$x_0 \quad x_1 \quad x_2 \quad x_2$

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

Add positional encoding

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

Transformer encoder

Attention attends over all the vectors
Add positional encoding

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

Transformer encoder

Residual connection

Attention attends over all the vectors

Add positional encoding

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

LayerNorm over each vector individually

Residual connection

Attention attends over all the vectors

Add positional encoding

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

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

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

Transformer encoder:
- Add positional encoding
- Attention attends over all the vectors
- Residual connection
- LayerNorm over each vector individually
- MLP over each vector individually

MLP:
- Residual connection
- LayerNorm
- Multi-head self-attention
- MLP over each vector individually

Example:
- $c_{0,0}, c_{0,1}, c_{0,2}, ...$, $z_{0,0}, z_{0,1}, z_{0,2}, ...$, $x_0, x_1, x_2, x_2$
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 block

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.
The Transformer
Decoder block

Most of the network is the same the transformer encoder.

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

**Inputs:** Set of vectors \( \mathbf{x} \) and Set of context vectors \( \mathbf{c} \).

**Outputs:** Set of vectors \( \mathbf{y} \).

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 x W x D

Transformer encoder

Transformer decoder

- person wearing hat [END]

- y_1 y_2 y_3 y_4

- y_0 y_1 y_2 y_3

- [START] person wearing hat
- Perhaps we don't need convolutions at all?

Image Captioning using transformers

Extract spatial features from a pretrained CNN

Features: H x W x D

Transformer encoder

Transformer decoder

```
[START] y_0 y_1 y_2 y_3 y_4 [END]
```

```
person wearing hat
```

```
z_{0,0} z_{0,1} z_{0,2}...
z_{1,0} z_{1,1} z_{1,2}...
z_{2,0} z_{2,1} z_{2,2}...
```

```
c_{0,0} c_{0,1} c_{0,2}...
c_{1,0} c_{1,1} c_{1,2}...
c_{2,0} c_{2,1} c_{2,2}...
```

```
y_0 y_1 y_2 y_3
```

```
[START] person wearing hat
```
Image Captioning using ONLY transformers

- Transformers from pixels to language

[Diagram showing the process of image captioning using transformers]

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 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

Fei-Fei Li, Yunzhu Li, Ruohan Gao
ConvNets strike back!

A ConvNet for the 2020s. Liu et al. CVPR 2022
DeiT III: Revenge of the ViT

Hugo Touvron*,†  Matthieu Cord†  Hervé Jégou*

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: Video Understanding