Lecture 11: Attention and Transformers
Administrative: Midterm

- Midterm was this Tuesday
- We will be grading this week and you should have grades by next week.
Administrative: Assignment 3

- A3 is due Friday May 25th, 11:59pm
  ○ Lots of applications of ConvNets
  ○ Also contains an extra credit notebook, which is worth an additional 5% of the A3 grade.
  ○ Extra credit will not be used when curving the class grades.
Last Time: Recurrent Neural Networks
Last Time: Variable length computation graph with shared weights
Let's jump to lecture 10 - slide 43
Today's Agenda:

- **Attention with RNNs**
  - In Computer Vision
  - In NLP
- **General Attention Layer**
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- **Transformers**
Today's Agenda:

- **Attention with RNNs**
  - In Computer Vision
  - In NLP
- **General Attention Layer**
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- **Transformers**
Image Captioning using **spatial features**

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

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

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
$f_w(.)$ 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

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

**MLP**

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


[START]
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 $c = h_0$

---

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

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

---

Extract spatial features from a pretrained CNN

---

**Decoder:**

$$y_t = g_V(y_{t-1}, h_{t-1}, c)$$

where context vector $c$ is often $c = h_0$

---


Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 11 - 13

May 06, 2021
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: $y_t = g_V(y_{t-1}, h_{t-1}, c)$
where context vector $c$ is often $c = h_0$

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$


Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 11 - 14
May 06, 2021
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

Image Captioning with RNNs & 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: \( H \times W \times D \)
Image Captioning with RNNs & Attention

Compute alignments scores (scalars):

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

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

Extract spatial features from a pretrained CNN

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

Image Captioning with RNNs & Attention

Extract spatial features from a pretrained CNN

Features: H x W x D

Alignement scores: H x W

Attention: H x W

Compute alignments scores (scalars):

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

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

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 & Attention

Extract spatial features from a pretrained CNN

Alignment scores: \( H \times W \)

Attention: \( H \times W \)

Normalize to get attention weights:

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

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

Compute context vector:

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

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 & Attention

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

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

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

Image Captioning with RNNs & Attention

Extract spatial features from a pretrained CNN

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

Alignment scores: \( H \times W \)

Attention: \( H \times W \)

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

New context vector at every time step

Image Captioning with RNNs & Attention

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

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

Image Captioning with RNNs & Attention

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

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

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 & Attention

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

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

```
<table>
<thead>
<tr>
<th>0,0</th>
<th>0,1</th>
<th>0,2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,0</td>
<td>1,1</td>
<td>1,2</td>
</tr>
<tr>
<td>2,0</td>
<td>2,1</td>
<td>2,2</td>
</tr>
</tbody>
</table>
```

```
```

Extract spatial features from a pretrained CNN

```
CNN
```

Features: H x W x D

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

```
[START] → person → wearing → hat → [END]
```

```
c_1 \downarrow y_0 \downarrow c_2 \downarrow y_1 \downarrow c_3 \downarrow y_2 \downarrow c_4 \downarrow y_3
```

```
h_0 \rightarrow h_1 \rightarrow h_2 \rightarrow h_3 \rightarrow h_4
```

Fei-Fei Li, Ranjay Krishna, Danfei Xu
Lecture 11 - 24
May 06, 2021
Image Captioning with RNNs & Attention


Alignment scores: $H \times W$

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

Attention: $H \times W$

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

Features: $H \times W \times D$

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

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


Extract spatial features from a pretrained CNN

Alignment scores: $H \times W$

Attention: $H \times W$

Features: $H \times W \times D$

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

$e_{1,0,0}, \ldots, e_{1,2,2}$

$y_1, \ldots, y_4$
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.
Attention can detect Gender Bias

Wrong

Right for the Right Reasons

Right for the Wrong Reasons

Right for the Right Reasons

Baseline: A man sitting at a desk with a laptop computer.

Our Model: A woman sitting in front of a laptop computer.

Baseline: A man holding a tennis racquet on a tennis court.

Our Model: A man holding a tennis racquet on a tennis court.

Burns et al. "Women also Snowboard: Overcoming Bias in Captioning Models" ECCV 2018
Figures from Burns et al, copyright 2018. Reproduced with permission.
Similar tasks in NLP - Language translation example

**Input:** Sequence \( x = x_1, x_2, \ldots, x_T \)

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

\( x_0 \) \( x_1 \) \( x_2 \) \( x_3 \)

personne portant un chapeau
Similar tasks in NLP - Language translation example

**Input**: Sequence $x = x_1, x_2, \ldots, x_T$

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

**Encoder**: $h_0 = f_W(z)$
where $z_t = \text{RNN}(x_t, u_{t-1})$
$f_W(.)$ is MLP
$u$ is the hidden RNN state
Similar tasks in NLP - Language translation example

**Input:** Sequence \( x = x_1, x_2, ..., x_T \)

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

**Encoder:** \( h_0 = f_W(z) \)
where \( z_t = \text{RNN}(x_t, u_{t-1}) \)
\( f_W(.) \) is MLP
\( u \) is the hidden RNN state

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

Diagram:

- **Input:** \( x_0, x_1, x_2, x_3 \)
  - \( x_0 \): personne
  - \( x_1 \): portant
  - \( x_2 \): un
  - \( x_3 \): chapeau

- **Output:** \( y_0, y_1, y_2, y_3, y_4 \)
  - \( y_0 \): [START]
  - \( y_1 \): person
  - \( y_2 \): wearing
  - \( y_3 \): hat
  - \( y_4 \): [END]

- **Context:** \( c \)

Graphical representation of the encoder-decoder model with input and output sequences.
Attention in NLP - Language translation example

Compute alignments scores (_scalars_):

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

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

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
Attention in NLP - Language translation example

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

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

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

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
Attention in NLP - Language translation example

Compute alignments scores (scalars):

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

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

Normalize to get attention weights:

$$a_{t, i} = \text{softmax}(e_{t,i})$$

$$0 < a_{t, i} < 1,$$

attention values sum to 1.

Compute context vector:

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

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Attention in NLP - Language translation example

Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, \mathbf{c})$
where context vector $\mathbf{c}$ is often $c = h_0$

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Similar visualization of attention weights

English to French translation example:

**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."

Without any attention supervision, model learns different word orderings for different languages.

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015
Today's Agenda:

- Attention with RNNs
  - In Computer Vision
  - In NLP
- General Attention Layer
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- Transformers
Attention we just saw in image captioning

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

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

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

**Diagram:**
- Alignment: \( a_{i,j} \)
- Softmax: \( a \)
- Multiplication and addition: \( c \)
General attention layer

Attention operation is permutation invariant.
- Doesn't care about ordering of the features
- Stretch H x W = N into N vectors

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: $c = \sum_i a_i x_i$
General attention layer

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

**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)
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: \( N \times D \))
Queries: \( q \) (shape: \( M \times D \))

**Outputs:**
context vectors: \( y \) (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 output context vector

Fei-Fei Li, Ranjay Krishna, Danfei Xu
Lecture 11 - 45
May 06, 2021
General attention layer

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

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

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.

**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: \( v = xW_v \)
General attention layer

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

Operations:
- Key vectors: \( k = x W_k \)
- Value vectors: \( v = x W_v \)
- 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.

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

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

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

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

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

Recall that the query vector was a function of the input vectors
Self attention layer

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

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{ij} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
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: \( D_v \))

```
\[
\begin{align*}
q_0 & \quad q_1 & \quad q_2 \\
\vdots & & \vdots \\
k_0 & \quad k_1 & \quad k_2 \\
\vdots & & \vdots \\
v_0 & \quad v_1 & \quad v_2 \\
\vdots & & \vdots \\
x_0 & \quad x_1 & \quad x_2 \\
\vdots & & \vdots \\
\end{align*}
\]
```
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: \( e_{i,j} = q_i \cdot k_j / \sqrt{D} \)
Attention: \( \mathbf{a} = \text{softmax}(\mathbf{e}) \)
Output: \( y_j = \sum_i a_{i,j} v_i \)

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

Alignment:
Input vectors: \( \mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2 \)
Key vectors: \( \mathbf{k}_0, \mathbf{k}_1, \mathbf{k}_2 \)
Value vectors: \( \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2 \)
Query vectors: \( \mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2 \)

Self-attention:
Output: \( y_0, y_1, y_2 \)

Self attention layer - attends over sets of inputs

Permutation invariant

**Problem:** how can we encode ordered sequences like language or spatially ordered image features?
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, \( p_j = pos(j) \)

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 $pos: N \rightarrow \mathbb{R}^d$ 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 $\text{pos}(.)$

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 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 $\omega_k = \frac{1}{10000^{2k/d}}$

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

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
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 desiderata
   - Input: \( \text{Intuition:} \)

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

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, \( p_j = \text{pos}(j) \)
**Masked self-attention layer**

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

**Alignment:**
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity

**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:**
- Visual representation of the self-attention layer with inputs, key, value, and query vectors.
Multi-head self attention layer
- Multiple self-attention heads in parallel

Add or concatenate

$y_0 \ y_1 \ y_2$

head$_0$

Self-attention

$y_0 \ y_1 \ y_2$

$x_0 \ x_1 \ x_2$

head$_1$

Self-attention

$y_0 \ y_1 \ y_2$

$x_0 \ x_1 \ x_2$

head$_H-1$

Self-attention

$y_0 \ y_1 \ y_2$

$x_0 \ x_1 \ x_2$

Split

$x_0 \ x_1 \ x_2$
General attention versus self-attention

![Diagram of attention and self-attention processes]
Comparing RNNs to Transformers

**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.

**Transformers:**

(+) 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)
Today's Agenda:

- Attention with RNNs
  - In Computer Vision
  - In NLP
- General Attention Layer
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- Transformers
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:
$H \times W \times D$

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

---

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

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

---

Extract spatial features from a pretrained CNN

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

---

Transformer encoder

Transformer decoder

---

Input: Image \( I \)

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

\[
\begin{align*}
\mathbf{c}_{0,0} & \quad \mathbf{c}_{0,1} & \quad \mathbf{c}_{0,2} & \quad \cdots \quad \mathbf{c}_{2,2} \\
\vdots & \quad \vdots & \quad \vdots & \quad \cdots \\
\mathbf{z}_{0,0} & \quad \mathbf{z}_{0,1} & \quad \mathbf{z}_{0,2} & \quad \cdots \quad \mathbf{z}_{2,2} \\
\end{align*}
\]

\[\vdots \times N\]

Positional encoding

Add positional encoding

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

- Multi-head self-attention
- Positional encoding

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

Transformer encoder block consists of:

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

Multi-head self-attention:

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

MLP over each vector individually
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

Transformer encoder

\[ \begin{align*}
&c_{0,0} \quad c_{0,1} \quad c_{0,2} \ldots \quad c_{2,2} \\
&\vdots \quad x \quad N
\end{align*} \]

\[ \begin{align*}
&z_{0,0} \quad z_{0,1} \quad z_{0,2} \ldots \quad z_{2,2}
\end{align*} \]

Residual connection
MLP over each vector individually
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

Transformer Encoder Block:

**Inputs:** Set of vectors $\mathbf{x}$
**Outputs:** Set of vectors $\mathbf{y}$

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 as 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 captions, 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 \( x \) 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

y_1 \ y_2 \ y_3 \ y_4

person wearing hat

y_0 \ y_1 \ y_2 \ y_3

[START] person wearing hat
Image Captioning using transformers

- Perhaps we don't need convolutions at all?

Extract spatial features from a pretrained CNN:

Features: H x W x D

Transformer encoder

Transformer decoder

[START] person wearing hat [END]

y_0 y_1 y_2 y_3 y_4

y_0 y_1 y_2 y_3

y_1 y_2 y_3 y_4

y_0 y_1 y_2 y_3

y_1 y_2 y_3 y_4

y_0 y_1 y_2 y_3

y_1 y_2 y_3 y_4

y_0 y_1 y_2 y_3
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
Image Captioning using ONLY transformers

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
New large-scale transformer models

An illustration of a baby daikon radish in a tutu walking a dog

An armchair in the shape of an avocado [...]

(link to more examples)
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: Unsupervised learning
VAEs and GANs