Lecture 12: Attention and Transformers
Last Time: Recurrent Neural Networks

- **one to one**
- **one to many**
- **many to one**
- **many to many**
Last Time: Variable length computation graph with shared weights
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 $y = y_1, y_2, ..., y_T$

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

---

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

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

---


---

Ali Farhadi, Aditya Kusupati

Lecture 12 - 8

Nov 14, 2023
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

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

Image Captioning using spatial features

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

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

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

Image Captioning with RNNs & Attention

Extract spatial features from a pretrained CNN

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

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

Alignment scores:

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

Features:

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

\[ h_0 \]

Image Captioning with RNNs & Attention

Compute alignments scores (scalars):

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

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

Extract spatial features from a pretrained CNN

Features:

H x W x D

Alignment scores:

\[ e_{1,0,0} \quad e_{1,0,1} \quad e_{1,0,2} \]
\[ e_{1,1,0} \quad e_{1,1,1} \quad e_{1,1,2} \]
\[ e_{1,2,0} \quad e_{1,2,1} \quad e_{1,2,2} \]

Attention:

\[ a_{1,0,0} \quad a_{1,0,1} \quad a_{1,0,2} \]
\[ a_{1,1,0} \quad a_{1,1,1} \quad a_{1,1,2} \]
\[ a_{1,2,0} \quad a_{1,2,1} \quad a_{1,2,2} \]

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

\[ Z_{0,0} \quad Z_{0,1} \quad Z_{0,2} \]
\[ Z_{1,0} \quad Z_{1,1} \quad Z_{1,2} \]
\[ Z_{2,0} \quad Z_{2,1} \quad Z_{2,2} \]

\[ h_0 \]

Image Captioning with RNNs & Attention

Extract spatial features from a pretrained CNN

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

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

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

$0 < a_{t,i,j} < 1$, attention values sum to 1

Compute context vector:

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

Image Captioning with RNNs & Attention

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

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

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

Alignment 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: $H \times W$

New context vector at every time step

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

Image Captioning with RNNs & Attention

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

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

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

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

Extract spatial features from a pretrained CNN

<table>
<thead>
<tr>
<th>Z₀₀</th>
<th>Z₀₁</th>
<th>Z₀₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z₁₀</td>
<td>Z₁₁</td>
<td>Z₁₂</td>
</tr>
<tr>
<td>Z₂₀</td>
<td>Z₂₁</td>
<td>Z₂₂</td>
</tr>
</tbody>
</table>

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

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

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

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 & Attention

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


Ali Farhadi, Aditya Kusupati
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 Bencho, 2015. Reproduced with permission.
Attention can detect Gender Bias

Wrong

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

Right for the Right Reasons

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

Right for the Wrong Reasons

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

Right for the Right Reasons

Our Model: A man holding a tennis racquet on a tennis court.
Similar tasks in NLP - Language translation example

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

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

$x_0$  personne

$x_1$  portant

$x_2$  un

$x_3$  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 \( z_t = \text{RNN}(x_t, u_{t-1}) \)

\( f_W(\cdot) \) is MLP

\( u \) is the hidden RNN state
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

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

Diagram:
- Encoder: \( z_0 \rightarrow z_1 \rightarrow z_2 \rightarrow h_0 \)
- Decoder: \( y_0 \rightarrow y_1 \rightarrow y_2 \rightarrow [\text{END}] \)

Example:
- Input: \( x = \text{personne, portant, un, chapeau} \)
- Output: \( y = \text{person, wearing, hat} \)
Attention in NLP - Language translation example

Compute alignments scores (scalars):

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

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

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

\[ a_0, a_1, a_2, a_3 \]

\[ \text{softmax} \]

\[ E_0, E_1, E_2, E_3 \]

\[ Z_0, Z_1, Z_2, Z_3 \]

\[ X_0, X_1, X_2, X_3 \]

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

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

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

0 < \( a_{t,i,j} < 1 \), 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: \( y_t = g_V(y_{t-1}, h_{t-1}, c) \)
where context vector \( c \) is often \( c = h_0 \)

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

Ali Farhadi, Aditya Kusupati
Lecture 12 - 32
Nov 14, 2023
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
Image Captioning with RNNs & Attention

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

Alignment scores: $H \times W$

Attention: $H \times W$

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

Attention we just saw in image captioning

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

**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: $D$)

**Features:**
- $z_{0,0}$, $z_{0,1}$, $z_{0,2}$
- $z_{1,0}$, $z_{1,1}$, $z_{1,2}$
- $z_{2,0}$, $z_{2,1}$, $z_{2,2}$

**Alignments:**
- $e_{0,0}$, $e_{0,1}$, $e_{0,2}$
- $e_{1,0}$, $e_{1,1}$, $e_{1,2}$
- $e_{2,0}$, $e_{2,1}$, $e_{2,2}$

**Attention:**
- $a_{0,0}$, $a_{0,1}$, $a_{0,2}$
- $a_{1,0}$, $a_{1,1}$, $a_{1,2}$
- $a_{2,0}$, $a_{2,1}$, $a_{2,2}$
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 x 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: $\mathbf{x}$ (shape: $N \times D$)
- Query: $\mathbf{h}$ (shape: $D$)

**Alignments:**
- $e_i = h \cdot x_i$
- $a_i = \text{softmax}(e_i)$

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

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

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

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

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

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

Change $f_{\text{att}}(\cdot)$ to a **scaled** simple dot product:
- High dimensionality means more terms in the dot product sum.
- Large magnitude vectors will cause softmax to peak and assign very little weight to all others.
- Dividing by $\sqrt{D}$ will reduce effect of large magnitude vectors.
**General attention layer**

**Outputs:**
- context vectors: \( y \) (shape: \( M \times D \))

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

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

**Multiple query vectors**
- each query creates a new output context vector
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.
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: $v = xW_v$

**Diagram:**

- Input vectors: $x_0, x_1, x_2$
- Key vectors: $k_0, k_1, k_2$
- Value vectors: $v_0, v_1, v_2$
- Queries: $q_0, q_1, q_2$
General attention layer

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

**Alignments:**
- 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: \( M \times D_v \))

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

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.
Deriving self-attention from this general attention layer
Recall that the query vector was a function of the input vectors

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

**Operations:**
Key vectors: \( \mathbf{k} = \mathbf{xW}_k \)
Value vectors: \( \mathbf{v} = \mathbf{xW}_v \)
Alignment: \( e_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D} \)
Attention: \( a = \text{softmax}(\mathbf{e}) \)
Output: \( y_j = \sum_i a_{i,j} \mathbf{v}_i \)

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

**Encoder:** \( h_0 = f_W(\mathbf{z}) \)
where \( \mathbf{z} \) is spatial CNN features
\( f_W(\cdot) \) is an MLP
Self attention layer

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: \( N \times D \))

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

**Outputs:**
Context vectors: \( y \) (shape: \( D_v \))
Self attention layer - attends over sets of inputs

**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: \( y_j = \sum_i a_{ij} v_i \)

**Inputs:**
Input vectors: \( x \) (shape: \( N \times D \))
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 $\text{pos}(.)$

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.

Desiderata of $\text{pos}(.)$:

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

So, $p_j = \text{pos}(j)$
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 $\omega_k = \frac{1}{10000^{2k/d}}$

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, $p_j = pos(j)$
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
   - \( p(t) = \begin{bmatrix} \sin(\omega_1.t) \\ \cos(\omega_1.t) \\ \sin(\omega_2.t) \\ \cos(\omega_2.t) \\ \vdots \\ \sin(\omega_{d/2}.t) \\ \cos(\omega_{d/2}.t) \end{bmatrix}_d \)
   - \( \omega_k = \frac{1}{10000^{2k/d}} \)

Intuition:

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 8 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 9 & 1 & 0 & 0 & 1 \\
2 & 0 & 0 & 1 & 0 & 10 & 1 & 0 & 1 & 0 \\
3 & 0 & 0 & 1 & 1 & 11 & 1 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 12 & 1 & 1 & 0 & 0 \\
5 & 0 & 1 & 0 & 1 & 13 & 1 & 1 & 0 & 1 \\
6 & 0 & 1 & 1 & 0 & 14 & 1 & 1 & 1 & 0 \\
7 & 0 & 1 & 1 & 1 & 15 & 1 & 1 & 1 & 1 \\
\end{array}
\]

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

**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: \( \mathbf{a} = \text{softmax}(\mathbf{e}) \)
- Output: \( y_j = \sum_i a_{ij} v_i \)

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

**Alignment**
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity.
Multi-head self attention layer
- Multiple self-attention heads in parallel

Self-attention

Add or concatenate

Split

$y_0$ $y_1$ $y_2$

head<sub>0</sub>

head<sub>1</sub>

head<sub>H-1</sub>

x<sub>0</sub> x<sub>1</sub> x<sub>2</sub>

x<sub>0</sub> x<sub>1</sub> x<sub>2</sub>

x<sub>0</sub> x<sub>1</sub> x<sub>2</sub>
General attention versus self-attention
Attention layers can process sequential inputs

Self-attention layers are masked
Comparing RNNs to masked multi-headed attention

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.

Masked multi-headed attention:
(+): 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 $y = y_1, y_2, ..., y_T$

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

---

**Diagram**:

- **CNN**: Extract spatial features from a pretrained CNN

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

- **Transformer encoder**: 

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

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

$T_w(.)$ is the transformer encoder

<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>

Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

**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**

$y_0$ $y_1$ $y_2$ $y_3$ $y_4$

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

where $T_D(.)$ is the transformer decoder

<table>
<thead>
<tr>
<th>person</th>
<th>wearing</th>
<th>hat</th>
<th>[END]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>$y_2$</td>
<td>$y_3$</td>
<td>$y_4$</td>
</tr>
</tbody>
</table>

**[START]**

person wearing hat

Ali Farhadi, Aditya Kusupati

Lecture 12 - 65

Nov 14, 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... x N

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

Transformer encoder

\[ \begin{array}{c}
  c_{0,0} \quad c_{0,1} \quad c_{0,2} \quad \ldots \quad c_{2,2} \\
  \vdots \quad x \quad N \\
  z_{0,0} \quad z_{0,1} \quad z_{0,2} \quad \ldots \quad z_{2,2} \\
\end{array} \]

Add positional encoding

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

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

\[ \vdots \times N \]

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

Multi-head self-attention

Positional encoding

\[ x_0, x_1, x_2, x_2 \]

Attention attends over all the vectors

Add positional encoding

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

Residual connection
Attention attends over all the vectors
Add positional encoding

Transformer encoder

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

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

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

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

- Positional encoding
- Multi-head self-attention
- Layer norm
- MLP

Transformer encoder

- 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

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

Transformer encoder

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

Transformer encoder

\[ \vdots \times N \]

Transformer encoder

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

MLP

LayerNorm

Multi-head self-attention

Positional encoding

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 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

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.
The Transformer Decoder block

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 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_0 y_1 y_2 y_3

y_0 y_1 y_2 y_3

y_0 y_1 y_2 y_3
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

person wearing hat [END]

y_1 y_2 y_3 y_4

y_0 y_1 y_2 y_3

[START] person wearing hat

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](https://colab.research.google.com) to an implementation of vision transformers
New large-scale transformer models

**TEXT PROMPT**
an illustration of a baby daikon radish in a tutu walking a dog

**AI-GENERATED IMAGES**

**TEXT PROMPT**
an armchair in the shape of an avocado [...]

**AI-GENERATED IMAGES**

[Link to more examples]
Transformers today

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
</tbody>
</table>

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018
### Transformers today

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>~340M</td>
<td>126GB</td>
<td>512x TPU-v3 (2.5 days)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>355M</td>
<td>160GB</td>
<td>1024x V100 GPUs (1 day)</td>
</tr>
</tbody>
</table>

Yang et al, XLNet: Generalized Autoregressive Pretraining for Language Understanding”, 2019 (Google)
Liu et al, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019 (Meta)
## Transformers today

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>~340M</td>
<td>126GB</td>
<td>512x TPU-v3 (2.5 days)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>355M</td>
<td>160GB</td>
<td>1024x V100 GPUs (1 day)</td>
</tr>
<tr>
<td>GPT-2</td>
<td>48</td>
<td>1600</td>
<td>?</td>
<td>1.5B</td>
<td>40GB</td>
<td>?</td>
</tr>
</tbody>
</table>

Radford et al, "Language models are unsupervised multitask learners", 2019 (OpenAI)
## Transformers today

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>~340M</td>
<td>126GB</td>
<td>512x TPU-v3 (2.5 days)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>355M</td>
<td>160GB</td>
<td>1024x V100 GPUs (1 day)</td>
</tr>
<tr>
<td>GPT-2</td>
<td>48</td>
<td>1600</td>
<td>?</td>
<td>1.5B</td>
<td>40GB</td>
<td>?</td>
</tr>
<tr>
<td>Megatron-LM</td>
<td>72</td>
<td>3072</td>
<td>32</td>
<td>8.3B</td>
<td>174GB</td>
<td>512x V100 GPU (9 days)</td>
</tr>
</tbody>
</table>

### Transformers today

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>~340M</td>
<td>126GB</td>
<td>512x TPU-v3 (2.5 days)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>355M</td>
<td>160GB</td>
<td>1024x V100 GPUs (1 day)</td>
</tr>
<tr>
<td>GPT-2</td>
<td>48</td>
<td>1600</td>
<td>?</td>
<td>1.5B</td>
<td>40GB</td>
<td>?</td>
</tr>
<tr>
<td>Megatron-LM</td>
<td>72</td>
<td>3072</td>
<td>32</td>
<td>8.3B</td>
<td>174GB</td>
<td>512x V100 GPU (9 days)</td>
</tr>
<tr>
<td>Turing-NLG</td>
<td>78</td>
<td>4256</td>
<td>28</td>
<td>17B</td>
<td>?</td>
<td>256x V100 GPUs</td>
</tr>
</tbody>
</table>

Microsoft, "Turing-NLG: A 17-billion parameter language model by Microsoft", 2020
## Transformers today

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>~340M</td>
<td>126GB</td>
<td>512x TPU-v3 (2.5 days)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>355M</td>
<td>160GB</td>
<td>1024x V100 GPUs (1 day)</td>
</tr>
<tr>
<td>GPT-2</td>
<td>48</td>
<td>1600</td>
<td>?</td>
<td>1.5B</td>
<td>40GB</td>
<td>?</td>
</tr>
<tr>
<td>Megatron-LM</td>
<td>72</td>
<td>3072</td>
<td>32</td>
<td>8.3B</td>
<td>174GB</td>
<td>512x V100 GPU (9 days)</td>
</tr>
<tr>
<td>Turing-NLG</td>
<td>78</td>
<td>4256</td>
<td>28</td>
<td>17B</td>
<td>?</td>
<td>256x V100 GPUs</td>
</tr>
<tr>
<td>GPT-3</td>
<td>96</td>
<td>12288</td>
<td>96</td>
<td>175B</td>
<td>694GB</td>
<td>?</td>
</tr>
</tbody>
</table>

Brown et al, "Language Models are Few-Shot Learners", NeurIPS 2020
### Transformers today

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>~340M</td>
<td>126GB</td>
<td>512x TPU-v3 (2.5 days)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>355M</td>
<td>160GB</td>
<td>1024x V100 GPUs (1 day)</td>
</tr>
<tr>
<td>GPT-2</td>
<td>48</td>
<td>1600</td>
<td>?</td>
<td>1.5B</td>
<td>40GB</td>
<td>?</td>
</tr>
<tr>
<td>Megatron-LM</td>
<td>72</td>
<td>3072</td>
<td>32</td>
<td>8.3B</td>
<td>174GB</td>
<td>512x V100 GPU (9 days)</td>
</tr>
<tr>
<td>Turing-NLG</td>
<td>78</td>
<td>4256</td>
<td>28</td>
<td>17B</td>
<td>?</td>
<td>256x V100 GPUs</td>
</tr>
<tr>
<td>GPT-3</td>
<td>96</td>
<td>12288</td>
<td>96</td>
<td>175B</td>
<td>694GB</td>
<td>?</td>
</tr>
<tr>
<td>Gopher</td>
<td>80</td>
<td>16384</td>
<td>128</td>
<td>280B</td>
<td>10.55TB</td>
<td>4096x TPU-v3 (38 days)</td>
</tr>
</tbody>
</table>

*Rae et al, "Scaling Language Models: Methods, Analysis, & Insights from Training Gopher", arXiv 2021 (Google)*
<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>~340M</td>
<td>126GB</td>
<td>512x TPU-v3 (2.5 days)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>355M</td>
<td>160GB</td>
<td>1024x V100 GPUs (1 day)</td>
</tr>
<tr>
<td>GPT-2</td>
<td>48</td>
<td>1600</td>
<td>?</td>
<td>1.5B</td>
<td>40GB</td>
<td>?</td>
</tr>
<tr>
<td>Megatron-LM</td>
<td>72</td>
<td>3072</td>
<td>32</td>
<td>8.3B</td>
<td>174GB</td>
<td>512x V100 GPU (9 days)</td>
</tr>
<tr>
<td>Turing-NLG</td>
<td>78</td>
<td>4256</td>
<td>28</td>
<td>17B</td>
<td>?</td>
<td>256x V100 GPUs</td>
</tr>
<tr>
<td>GPT-3</td>
<td>96</td>
<td>12288</td>
<td>96</td>
<td>175B</td>
<td>694GB</td>
<td>?</td>
</tr>
<tr>
<td>Gopher</td>
<td>80</td>
<td>16384</td>
<td>128</td>
<td>280B</td>
<td>10.55TB</td>
<td>4096x TPU-v3 (38 days)</td>
</tr>
</tbody>
</table>
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: Modern architectures