Attention and Transformers

Arjun Majumdar
Georgia Tech
Lecture Outline

• Machine Translation with RNNs
• RNNs with Attention
• From Attention to Transformers
• What can Transformers do?
Sequence Modeling with RNNs
Machine Translation

we are eating bread  →  estamos comiendo pan
Machine Translation

RNN Encoder

we are eating bread

RNN Decoder

estamos comiendo pan
Machine Translation with RNNs

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

Slide credit: Justin Johnson
Machine Translation with RNNs

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

$s_0 = h_4$
Machine Translation with RNNs

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

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

Slide credit: Justin Johnson
Machine Translation with RNNs

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

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

Slide credit: Justin Johnson
Machine Translation with RNNs

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

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

[START] $s_0$

$e$ $st$ amos $c$omiendo $p$an $[STOP]$

Slide credit: Justin Johnson
Machine Translation with RNNs

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

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

Problem: \( s_i \) is used to encode input and maintain decoder state

Slide credit: Justin Johnson
Machine Translation with RNNs

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

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

Solution: add a context vector $c = h_4$ and predict $s_0$ from $h_4$

Slide credit: Justin Johnson
Machine Translation with RNNs

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

Decoder: $s_t = g_u(y_t, s_{t-1}, c)$

Solution: add a context vector $c = h_4$ and predict $s_0$ from $h_4$
Machine Translation with RNNs

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

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

Problem: Input sequence bottlenecked through fixed-sized vector.

Slide credit: Justin Johnson
Machine Translation with RNNs

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

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

Idea: use new context vector at each step of decoder!
Machine Translation with RNNs and Attention

From final hidden state: 

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

\[
\begin{align*}
  h_1 & \rightarrow h_2 & \rightarrow h_3 & \rightarrow h_4 & \rightarrow s_0 \\
  x_1 & \rightarrow x_2 & \rightarrow x_3 & \rightarrow x_4 \\
  \text{we} & \rightarrow \text{are} & \rightarrow \text{eating} & \rightarrow \text{bread}
\end{align*}
\]

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

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

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

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

Inputs:
- \( x_1 \): we
- \( x_2 \): are
- \( x_3 \): eating
- \( x_4 \): bread

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

From final hidden state:

Initial decoder state $s_0$

**Compute alignment scores**

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

**Normalize to get attention weights**

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

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

Slide credit: Justin Johnson
Machine Translation with RNNs and Attention

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

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

Set context vector \( c \) to a linear combination of hidden states
\[ c_t = \sum_i a_{t,i} h_i \]

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

Slide credit: Justin Johnson

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

we are eating bread
set context vector \( c \) to a linear combination of hidden states

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

compute alignment scores

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

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

normalize to get attention weights

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

from final hidden state:

\[
\text{Initial decoder state } s_0
\]

we are eating bread

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

Slide credit: Justin Johnson
Machine Translation with RNNs and Attention

From final hidden state:

**Initial decoder state** $s_0$

Compute **alignment scores**

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

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

**Estamos**

Normalize to get **attention weights**

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

Set context vector $c_t$ to a linear combination of hidden states

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

This is all differentiable! Do not supervise attention weights – backprop through everything

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

Set context vector $c_t$ to a linear combination of hidden states $c_t = \sum_i a_{t,i} h_i$

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

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

This is all differentiable! Do not supervise attention weights – backprop through everything

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

Slide credit: Justin Johnson
Machine Translation 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

Slide credit: Justin Johnson
Machine Translation with RNNs and Attention

we are eating bread

estamos comiendo

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

Slide credit: Justin Johnson
Machine Translation with RNNs and Attention

Intuition: Context vector attends to the relevant part of the input sequence “comiendo” = “eating”

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

Slide credit: Justin Johnson
Machine Translation 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.”


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.”


Diagonal attention means words correspond in order
Example: English to French translation

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


Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order

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

Slide credit: Justin Johnson
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
**Attention Layer**

**Inputs:**
- **State vector:** $s_i$ (Shape: $D_Q$)
- **Hidden vectors:** $h_i$ (Shape: $N_X \times D_H$)
- **Similarity function:** $f_{\text{att}}$

**Computation:**
- **Similarities:** $e$ (Shape: $N_X$)  $e_i = f_{\text{att}}(s_{t-1}, h_i)$
- **Attention weights:** $a = \text{softmax}(e)$  (Shape: $N_X$)
- **Output vector:** $y = \sum_i a_i h_i$  (Shape: $D_X$)
Attention Layer

Inputs:

Query vector: \( q \) (Shape: \( D_Q \))
Input vectors: \( X \) (Shape: \( N_X \times D_X \))

Similarity function: \( f_{\text{att}} \)

Computation:

Similarities: \( e \) (Shape: \( N_X \)) \( e_i = f_{\text{att}}(q, X_i) \)
Attention weights: \( a = \text{softmax}(e) \) (Shape: \( N_X \))
Output vector: \( y = \sum_i a_i X_i \) (Shape: \( D_X \))
Attention Layer

**Inputs:**
- **Query vector:** $q$ (Shape: $D_Q$)
- **Input vectors:** $X$ (Shape: $N \times D_Q$)

**Similarity function:** dot product

**Computation:**
- **Similarities:** $e$ (Shape: $N$) \[ e_i = q \cdot X_i \]
- **Attention weights:** $a = \text{softmax}(e)$ (Shape: $N$)
- **Output vector:** $y = \sum_i a_i X_i$ (Shape: $D_X$)

**Changes:**
- Use dot product for similarity
Attention Layer

**Inputs:**
- Query vector: $q$ (Shape: $D_Q$)
- Input vectors: $X$ (Shape: $N \times D_Q$)

**Similarity function:** scaled dot product

**Computation:**
- Similarities: $e$ (Shape: $N$)  
  $e_i = \frac{q \cdot X_i}{\sqrt{D_Q}}$
- Attention weights: $a = \text{softmax}(e)$ (Shape: $N$)
- Output vector: $y = \sum_i a_i X_i$ (Shape: $D_X$)

**Changes:**
- Use scaled dot product for similarity

Slide credit: Justin Johnson
**Attention Layer**

**Inputs:**

- **Query vectors:** \( Q \) (Shape: \( N_Q \times D_Q \))
- **Input vectors:** \( X \) (Shape: \( N_X \times D_Q \))

**Computation:**

- **Similarities:** \( E = QX^T \) (Shape: \( N_Q \times N_X \)) \( E_{i,j} = Q_i \cdot X_j / \sqrt{D_Q} \)
- **Attention weights:** \( A = \text{softmax}(E, \text{dim}=1) \) (Shape: \( N_Q \times N_X \))
- **Output vectors:** \( Y = AX \) (Shape: \( N_Q \times D_X \)) \( Y_i = \sum_j A_{i,j}X_j \)

**Changes:**
- Use dot product for similarity
- Multiple query vectors

Slide credit: Justin Johnson
**Attention Layer**

**Inputs:**
- **Query vectors:** $Q$ (Shape: $N_Q \times D_Q$)
- **Input vectors:** $X$ (Shape: $N_X \times D_X$)

**Key matrix:** $W_K$ (Shape: $D_X \times D_Q$)
**Value matrix:** $W_V$ (Shape: $D_X \times D_V$)

**Computation:**
- **Key vectors:** $K = XW_K$ (Shape: $N_X \times D_Q$)
- **Value vectors:** $V = XW_V$ (Shape: $N_X \times D_V$)
- **Similarities:** $E = QK^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = Q_i \cdot K_j / \sqrt{D_Q}$
- **Attention weights:** $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
- **Output vectors:** $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j}V_j$

**Changes:**
- Use dot product for similarity
- Multiple query vectors
- Separate key and value

Slide credit: Justin Johnson
Attention Layer

**Inputs:**
- Query vectors: $Q$ (Shape: $N_Q \times D_Q$)
- Input vectors: $X$ (Shape: $N_X \times D_X$)
- Key matrix: $W_K$ (Shape: $D_X \times D_Q$)
- Value matrix: $W_V$ (Shape: $D_X \times D_V$)

**Computation:**
- Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
- Value vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
- Similarities: $E = QK^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = Q_i \cdot K_j / \sqrt{D_Q}$
- Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
- Output vectors: $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} V_j$
Attention Layer

**Inputs:**
- **Query vectors:** $Q$ (Shape: $N_Q \times D_Q$)
- **Input vectors:** $X$ (Shape: $N_X \times D_X$)
- **Key matrix:** $W_K$ (Shape: $D_X \times D_Q$)
- **Value matrix:** $W_V$ (Shape: $D_X \times D_V$)

**Computation:**
- **Key vectors:** $K = X W_K$ (Shape: $N_X \times D_Q$)
- **Value vectors:** $V = X W_V$ (Shape: $N_X \times D_V$)
- **Similarities:** $E = Q K^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = Q_i \cdot K_j / \sqrt{D_Q}$
- **Attention weights:** $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
- **Output vectors:** $Y = A V$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

Slide credit: Justin Johnson
Attention Layer

**Inputs:**
Query vectors: $Q$ (Shape: $N_Q \times D_Q$)
Input vectors: $X$ (Shape: $N_X \times D_X$)
Key matrix: $W_K$ (Shape: $D_X \times D_Q$)
Value matrix: $W_V$ (Shape: $D_X \times D_V$)

**Computation:**
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T$ (Shape: $N_Q \times N_X$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{ij}V_j$
Attention Layer

**Inputs:**
- Query vectors: \( Q \) (Shape: \( N_Q \times D_Q \))
- Input vectors: \( X \) (Shape: \( N_X \times D_X \))
- Key matrix: \( W_K \) (Shape: \( D_X \times D_Q \))
- Value matrix: \( W_V \) (Shape: \( D_X \times D_V \))

**Computation:**
- Key vectors: \( K = XW_K \) (Shape: \( N_X \times D_Q \))
- Value vectors: \( V = XW_V \) (Shape: \( N_X \times D_V \))
- Similarities: \( E = QK^T \) (Shape: \( N_Q \times N_X \)) \( E_{ij} = Q_i \cdot K_j / \sqrt{D_Q} \)
- Attention weights: \( A = \text{softmax}(E, \text{dim}=1) \) (Shape: \( N_Q \times N_X \))
- Output vectors: \( Y = AV \) (Shape: \( N_Q \times D_V \)) \( Y_i = \sum_j A_{ij} V_j \)
Attention Layer

**Inputs:**
- **Query vectors:** $Q$ (Shape: $N_Q \times D_Q$)
- **Input vectors:** $X$ (Shape: $N_X \times D_X$)
- **Key matrix:** $W_K$ (Shape: $D_X \times D_Q$)
- **Value matrix:** $W_V$ (Shape: $D_X \times D_V$)

**Computation:**
- **Key vectors:** $K = XW_K$ (Shape: $N_X \times D_Q$)
- **Value vectors:** $V = XW_V$ (Shape: $N_X \times D_V$)
- **Similarities:** $E = QK^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = Q_i \cdot K_j / \sqrt{D_Q}$
- **Attention weights:** $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
- **Output vectors:** $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

Slide credit: Justin Johnson
Attention Layer

**Inputs:**
- Query vectors: $Q$ (Shape: $N_Q \times D_Q$)
- Input vectors: $X$ (Shape: $N_X \times D_X$)
- Key matrix: $W_K$ (Shape: $D_X \times D_Q$)
- Value matrix: $W_V$ (Shape: $D_X \times D_V$)

**Computation:**
- Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
- Value vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
- Similarities: $E = QK^T$ (Shape: $N_Q \times N_X$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
- Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
- Output vectors: $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j}V_j$
Self-Attention Layer

One query per input vector

**Inputs:**

Input vectors: \( X \) (Shape: \( N_x \times D_x \))

Key matrix: \( W_K \) (Shape: \( D_x \times D_Q \))

Value matrix: \( W_V \) (Shape: \( D_x \times D_V \))

Query matrix: \( W_Q \) (Shape: \( D_x \times D_Q \))

**Computation:**

Query vectors: \( Q = XW_Q \)

Key vectors: \( K = XW_K \) (Shape: \( N_x \times D_Q \))

Value vectors: \( V = XW_V \) (Shape: \( N_x \times D_V \))

Similarities: \( E = QK^T \) (Shape: \( N_x \times N_x \)) \( E_{ij} = Q_i \cdot K_j / \sqrt{D_Q} \)

Attention weights: \( A = \text{softmax}(E, \text{dim}=1) \) (Shape: \( N_x \times N_x \))

Output vectors: \( Y = AV \) (Shape: \( N_x \times D_V \)) \( Y_i = \sum_j A_{ij} V_j \)
Self-Attention Layer

One query per input vector

Inputs:
Input vectors: $X$ (Shape: $N \times D_x$)
Key matrix: $W_K$ (Shape: $D_x \times D_Q$)
Value matrix: $W_V$ (Shape: $D_x \times D_V$)
Query matrix: $W_Q$ (Shape: $D_x \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N \times D_Q$)
Value vectors: $V = XW_V$ (Shape: $N \times D_V$)
Similarities: $E = QK^T$ (Shape: $N \times N_x$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N \times N_x$)
Output vectors: $Y = AV$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{ij} V_j$
Self-Attention Layer

One **query** per **input vector**

**Inputs:**
Input vectors: $X$ (Shape: $N \times D_X$)
Key matrix: $W_K$ (Shape: $D_X \times D_Q$)
Value matrix: $W_V$ (Shape: $D_X \times D_V$)
Query matrix: $W_Q$ (Shape: $D_X \times D_Q$)

**Computation:**
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N \times D_Q$)
Value vectors: $V = XW_V$ (Shape: $N \times D_V$)
Similarities: $E = QK^T$ (Shape: $N \times N$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N \times N$)
Output vectors: $Y = AV$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{i,j}V_j$
Self-Attention Layer

One query per input vector

**Inputs:**
- **Input vectors:** $X$ (Shape: $N \times D_X$)
- **Key matrix:** $W_K$ (Shape: $D_X \times D_Q$)
- **Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
- **Query matrix:** $W_Q$ (Shape: $D_X \times D_Q$)

**Computation:**
- **Query vectors:** $Q = XW_Q$
- **Key vectors:** $K = XW_K$ (Shape: $N \times D_Q$)
- **Value vectors:** $V = XW_V$ (Shape: $N \times D_V$)
- **Similarities:** $E = QK^T$ (Shape: $N \times N_X$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
- **Attention weights:** $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N \times N_X$)
- **Output vectors:** $Y = AV$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{ij} V_j$

Slide credit: Justin Johnson
Self-Attention Layer

One query per input vector

Inputs:
Input vectors: $X$ (Shape: $N \times D_X$)
Key matrix: $W_K$ (Shape: $D_X \times D_Q$)
Value matrix: $W_V$ (Shape: $D_X \times D_V$)
Query matrix: $W_Q$ (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N \times D_Q$)
Value vectors: $V = XW_V$ (Shape: $N \times D_V$)
Similarities: $E = QK^T$ (Shape: $N \times N$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N \times N$)
Output vectors: $Y = AV$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{ij}V_j$
Self-Attention Layer

One query per input vector

Inputs:
Input vectors: \( \mathbf{X} \) (Shape: \( N \times D_X \))
Key matrix: \( \mathbf{W}_K \) (Shape: \( D_X \times D_Q \))
Value matrix: \( \mathbf{W}_V \) (Shape: \( D_X \times D_V \))
Query matrix: \( \mathbf{W}_Q \) (Shape: \( D_X \times D_Q \))

Computation:
Query vectors: \( \mathbf{Q} = \mathbf{XW}_Q \)
Key vectors: \( \mathbf{K} = \mathbf{XW}_K \) (Shape: \( N \times D_Q \))
Value vectors: \( \mathbf{V} = \mathbf{XW}_V \) (Shape: \( N \times D_V \))
Similarities: \( \mathbf{E} = \mathbf{QK}^T \) (Shape: \( N \times N \)) \( E_{ij} = Q_i \cdot K_j / \sqrt{D_Q} \)
Attention weights: \( \mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1) \) (Shape: \( N \times N \))
Output vectors: \( \mathbf{Y} = \mathbf{AV} \) (Shape: \( N \times D_V \)) \( Y_i = \sum_j A_{ij} V_j \)
Self-Attention Layer

One query per input vector

Inputs:
Input vectors: \( \mathbf{X} \) (Shape: \( N \times D \times X \))
Key matrix: \( \mathbf{W}_K \) (Shape: \( D \times D \times Q \))
Value matrix: \( \mathbf{W}_V \) (Shape: \( D \times D \times V \))
Query matrix: \( \mathbf{W}_Q \) (Shape: \( D \times D \times Q \))

Computation:
Query vectors: \( \mathbf{Q} = \mathbf{XW}_Q \)
Key vectors: \( \mathbf{K} = \mathbf{XW}_K \) (Shape: \( N \times D \times Q \))
Value vectors: \( \mathbf{V} = \mathbf{XW}_V \) (Shape: \( N \times D \times V \))
Similarities: \( \mathbf{E} = \mathbf{QK}^T \) (Shape: \( N \times N \times X \)) \( E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \sqrt{D_Q} \)
Attention weights: \( \mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1) \) (Shape: \( N \times N \times X \))
Output vectors: \( \mathbf{Y} = \mathbf{AV} \) (Shape: \( N \times D \times V \)) \( Y_i = \sum_j A_{ij} V_j \)

Slide credit: Justin Johnson
Consider **permuting** the input vectors:

**Inputs:**
- Input vectors: $\mathbf{X}$ (Shape: $N \times D_X$)
- Key matrix: $\mathbf{W}_K$ (Shape: $D_X \times D_Q$)
- Value matrix: $\mathbf{W}_V$ (Shape: $D_X \times D_V$)
- Query matrix: $\mathbf{W}_Q$ (Shape: $D_X \times D_Q$)

**Computation:**
- Query vectors: $\mathbf{Q} = \mathbf{XW}_Q$
- Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N \times D_Q$)
- Value vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N \times D_V$)
- Similarities: $\mathbf{E} = \mathbf{QK}^T$ (Shape: $N \times N$) $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \sqrt{D_Q}$
- Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N \times N$)
- Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{ij} V_j$
Self-Attention Layer

Consider **permuting** the input vectors:
Queries and Keys will be the same, but permuted

**Inputs:**
- **Input vectors:** \( X \) (Shape: \( N \times D_x \))
- **Key matrix:** \( W_K \) (Shape: \( D_x \times D_Q \))
- **Value matrix:** \( W_V \) (Shape: \( D_x \times D_V \))
- **Query matrix:** \( W_Q \) (Shape: \( D_x \times D_Q \))

**Computation:**
- **Query vectors:** \( Q = XW_Q \)
- **Key vectors:** \( K = XW_K \) (Shape: \( N \times D_Q \))
- **Value Vectors:** \( V = XW_V \) (Shape: \( N \times D_V \))
- **Similarities:** \( E = QK^T \) (Shape: \( N \times N \)) \( E_{ij} = Q_i \cdot K_j / \sqrt{D_Q} \)
- **Attention weights:** \( A = \text{softmax}(E, \text{dim}=1) \) (Shape: \( N \times N \))
- **Output vectors:** \( Y = AV \) (Shape: \( N \times D_V \)) \( Y_i = \sum_j A_{ij} V_j \)
Self-Attention Layer

**Inputs:**
- **Input vectors:** $X$ (Shape: $N \times D_x$)
- **Key matrix:** $W_K$ (Shape: $D_x \times D_Q$)
- **Value matrix:** $W_V$ (Shape: $D_x \times D_V$)
- **Query matrix:** $W_Q$ (Shape: $D_x \times D_Q$)

**Consider permuting the input vectors:**

**Similarities will be the same, but permuted**

**Computation:**
- **Query vectors:** $Q = XW_Q$
- **Key vectors:** $K = XW_K$ (Shape: $N \times D_Q$)
- **Value vectors:** $V = XW_V$ (Shape: $N \times D_V$)
- **Similarities:** $E = QK^T$ (Shape: $N \times N$) $E_{i,j} = Q_i \cdot K_j / \sqrt{D_Q}$
- **Attention weights:** $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N \times N$)
- **Output vectors:** $Y = AV$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{i,j}V_j$
Consider **permuting** the input vectors:

Attention weights will be the same, but permuted

**Inputs:**
- Input vectors: $X$  (Shape: $N \times D_X$)
- Key matrix: $W_K$  (Shape: $D_X \times D_Q$)
- Value matrix: $W_V$  (Shape: $D_X \times D_V$)
- Query matrix: $W_Q$  (Shape: $D_X \times D_Q$)

**Computation:**
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$  (Shape: $N \times D_Q$)
- Value vectors: $V = XW_V$  (Shape: $N \times D_V$)
- Similarities: $E = QK^T$  (Shape: $N \times N$)  $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
- Attention weights: $A = \text{softmax}(E, \text{dim}=1)$  (Shape: $N \times N$)
- Output vectors: $Y = AV$  (Shape: $N \times D_V$)  $Y_i = \sum_j A_{i,j} V_j$
Self-Attention Layer

**Inputs:**
- Input vectors: \( \mathbf{X} \) (Shape: \( N \times D_x \))
- Key matrix: \( \mathbf{W}_K \) (Shape: \( D_x \times D_Q \))
- Value matrix: \( \mathbf{W}_V \) (Shape: \( D_x \times D_V \))
- Query matrix: \( \mathbf{W}_Q \) (Shape: \( D_x \times D_Q \))

**Computation:**
- Query vectors: \( \mathbf{Q} = \mathbf{XW}_Q \)
- Key vectors: \( \mathbf{K} = \mathbf{XW}_K \) (Shape: \( N \times D_Q \))
- Value vectors: \( \mathbf{V} = \mathbf{XW}_V \) (Shape: \( N \times D_V \))
- Similarities: \( \mathbf{E} = \mathbf{QK}^T \) (Shape: \( N \times N \)) \( E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \sqrt{D_Q} \)
- Attention weights: \( \mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1) \) (Shape: \( N \times N \))
- Output vectors: \( \mathbf{Y} = \mathbf{AV} \) (Shape: \( N \times D_V \)) \( Y_i = \sum_j A_{ij} V_j \)

Consider **permuting** the input vectors:
- Values will be the same, but permuted

Slide credit: Justin Johnson
Self-Attention Layer

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

**Inputs:**
- **Input vectors:** \( X \) (Shape: \( N_x \times D_x \))
- **Key matrix:** \( W_k \) (Shape: \( D_x \times D_Q \))
- **Value matrix:** \( W_v \) (Shape: \( D_x \times D_V \))
- **Query matrix:** \( W_q \) (Shape: \( D_x \times D_Q \))

**Computation:**
- **Query vectors:** \( Q = X W_q \)
- **Key vectors:** \( K = X W_k \) (Shape: \( N_x \times D_Q \))
- **Value vectors:** \( V = X W_v \) (Shape: \( N_x \times D_V \))
- **Similarities:** \( E = Q K^T \) (Shape: \( N_x \times N_x \)) \( E_{i,j} = \frac{Q_i \cdot K_j}{\sqrt{D_Q}} \)
- **Attention weights:** \( A = \text{softmax}(E, \text{dim}=1) \) (Shape: \( N_x \times N_x \))
- **Output vectors:** \( Y = AV \) (Shape: \( N_x \times D_V \)) \( Y_i = \sum_j A_{i,j} V_j \)
Self-Attention Layer

Inputs:
Input vectors: $X$ (Shape: $N \times D_X$)
Key matrix: $W_K$ (Shape: $D_X \times D_Q$)
Value matrix: $W_V$ (Shape: $D_X \times D_V$)
Query matrix: $W_Q$ (Shape: $D_X \times D_Q$)

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Self-attention layer is **Permutation Equivariant**

$f(s(x)) = s(f(x))$

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N \times D_Q$)
Value vectors: $V = XW_V$ (Shape: $N \times D_V$)
Similarities: $E = QK^T$ (Shape: $N \times N$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N \times N$)
Output vectors: $Y = AV$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{ij}V_j$
Self-Attention Layer

**Inputs:**
- **Input vectors**: $X$ (Shape: $N \times D_X$)
- **Key matrix**: $W_K$ (Shape: $D_X \times D_Q$)
- **Value matrix**: $W_V$ (Shape: $D_X \times D_V$)
- **Query matrix**: $W_Q$ (Shape: $D_X \times D_Q$)

**Computation:**
- **Query vectors**: $Q = X W_Q$
- **Key vectors**: $K = X W_K$ (Shape: $N \times D_Q$)
- **Value vectors**: $V = X W_V$ (Shape: $N \times D_V$)
- **Similarities**: $E = Q K^T$ (Shape: $N \times N$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
- **Attention weights**: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N \times N$)
- **Output vectors**: $Y = A V$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{ij} V_j$

Self attention doesn’t “know” the order of the vectors it is processing!

Slide credit: Justin Johnson
Self-Attention Layer

**Inputs:**
- **Input vectors:** $X$ (Shape: $N_x \times D_x$)
- **Key matrix:** $W_K$ (Shape: $D_x \times D_Q$)
- **Value matrix:** $W_V$ (Shape: $D_x \times D_V$)
- **Query matrix:** $W_Q$ (Shape: $D_x \times D_Q$)

**Computation:**
- **Query vectors:** $Q = XW_Q$
- **Key vectors:** $K = XW_K$ (Shape: $N_x \times D_Q$)
- **Value vectors:** $V = XW_V$ (Shape: $N_x \times D_V$)
- **Similarities:** $E = QK^T$ (Shape: $N_x \times N_x$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
- **Attention weights:** $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_x \times N_x$)
- **Output vectors:** $Y = AV$ (Shape: $N_x \times D_V$) $Y_i = \sum_j A_{ij}V_j$

Self attention doesn’t “know” the order of the vectors it is processing!

In order to make processing position-aware, concatenate input with **positional encoding**

E can be learned lookup table, or fixed function

Slide credit: Justin Johnson
**Masked Self-Attention Layer**

**Inputs:**
- Input vectors: $X$ (Shape: $N_x \times D_x$)
- Key matrix: $W_K$ (Shape: $D_x \times D_Q$)
- Value matrix: $W_V$ (Shape: $D_x \times D_V$)
- Query matrix: $W_Q$ (Shape: $D_x \times D_Q$)

**Computation:**
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$ (Shape: $N_x \times D_Q$)
- Value vectors: $V = XW_V$ (Shape: $N_x \times D_V$)
- Similarities: $E = QK^T$ (Shape: $N_x \times N_x$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
- Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_x \times N_x$)
- Output vectors: $Y = AV$ (Shape: $N_x \times D_V$) $Y_i = \sum_j A_{i,j}V_j$

Don’t let vectors “look ahead” in the sequence

Used for language modeling (predict next word)

Slide credit: Justin Johnson
Multihead Self-Attention Layer

**Inputs:**
- **Input vectors:** $X$ (Shape: $N \times D_X$)
- **Key matrix:** $W_K$ (Shape: $D_X \times D_Q$)
- **Value matrix:** $W_V$ (Shape: $D_X \times D_V$)
- **Query matrix:** $W_Q$ (Shape: $D_X \times D_Q$)

**Computation:**
- **Query vectors:** $Q = XW_Q$
- **Key vectors:** $K = XW_K$ (Shape: $N \times D_Q$)
- **Value vectors:** $V = XW_V$ (Shape: $N \times D_V$)
- **Similarities:** $E = QK^T$ (Shape: $N \times N_X$) $E_{ij} = Q_i \cdot K_j / \sqrt{D_Q}$
- **Attention weights:** $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N \times N_X$)
- **Output vectors:** $Y = AV$ (Shape: $N \times D_V$) $Y_i = \sum_j A_{ij} V_j$

Use $H$ independent “Attention Heads” in parallel.
Three Ways of Processing Sequences

Recurrent Neural Network

Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, $h_T$ "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

Slide credit: Justin Johnson
Three Ways of Processing Sequences

Recurrent Neural Network

- Works on **Ordered Sequences**
  - (+) Good at long sequences: After one RNN layer, $h_T$ "sees" the whole sequence
  - (-) Not parallelizable: need to compute hidden states sequentially

1D Convolution

- Works on **Multidimensional Grids**
  - (-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence
  - (+) Highly parallel: Each output can be computed in parallel

Slide credit: Justin Johnson
Three Ways of Processing Sequences

**Recurrent Neural Network**
- Works on **Ordered Sequences**
  - (+) Good at long sequences: After one RNN layer, $h_T$ "sees" the whole sequence
  - (-) Not parallelizable: need to compute hidden states sequentially

**1D Convolution**
- Works on **Multidimensional Grids**
  - (-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence
  - (+) Highly parallel: Each output can be computed in parallel

**Self-Attention**
- Works on **Sets of Vectors**
  - (+) Good at long sequences: after one self-attention layer, each output "sees" all inputs!
  - (+) Highly parallel: Each output can be computed in parallel
  - (-) Very memory intensive

---

Slide credit: Justin Johnson
### Three Ways of Processing Sequences

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recurrent Neural Network</td>
<td>Works on <strong>Ordered Sequences</strong></td>
<td>(+) Good at long sequences: After one RNN layer, $h_T$ &quot;sees&quot; the whole sequence</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-) Bad at long sequences: Need to stack many conv layers for outputs to &quot;see&quot; the whole sequence</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+) Highly parallel: Each output can be computed in parallel</td>
<td>(-) Not parallelizable: need to compute hidden states sequentially</td>
</tr>
<tr>
<td>1D Convolution</td>
<td>Works on <strong>Multidimensional Grids</strong></td>
<td>(+) Good at long sequences: after one self-attention layer, each output &quot;sees&quot; all inputs!</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+) Highly parallel: Each output can be computed in parallel</td>
<td>(-) Very memory intensive</td>
</tr>
<tr>
<td>Self-Attention</td>
<td>Works on <strong>Sets of Vectors</strong></td>
<td>(+) Good at long sequences: after one self-attention layer, each output &quot;sees&quot; all inputs!</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(+) Highly parallel: Each output can be computed in parallel</td>
<td>(-) Very memory intensive</td>
</tr>
</tbody>
</table>

---

**Attention is all you need**

Vaswani et al, NeurIPS 2017

---

Slide credit: Justin Johnson
The Transformer

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

All vectors interact with each other

Self-Attention

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

MLP independently on each vector

All vectors interact with each other

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

Slide credit: Justin Johnson
The Transformer

MLP independently on each vector

Residual connection
All vectors interact with each other

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

Slide credit: Justin Johnson
The Transformer

Recall **Layer Normalization:**
- Given $h_1, ..., h_N$ (Shape: D)
- scale: $\gamma$ (Shape: D)
- shift: $\beta$ (Shape: D)
- $\mu_i = (1/D) \sum_j h_{ij}$ (scalar)
- $\sigma_i = (\sum_j (h_{ij} - \mu_i)^2)^{1/2}$ (scalar)
- $z_i = (h_i - \mu_i) / \sigma_i$
- $y_i = \gamma * z_i + \beta$

Ba et al, 2016

MLP independently on each vector

Residual connection

All vectors interact with each other

MLP

Layer Normalization

Self-Attention

Ba et al, 2016

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

MLP independently on each vector

Residual connection

All vectors interact with each other

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

Slide credit: Justin Johnson
The Transformer

Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other

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

**Transformer Block:**
*Input:* Set of vectors $x$  
*Output:* Set of vectors $y$

Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

---

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

Slide credit: Justin Johnson
The Transformer

Transformer Block:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interaction between vectors!
Layer norm and MLP work independently per vector
Highly scalable, highly parallelizable

A Transformer is a sequence of transformer blocks

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

Slide credit: Justin Johnson
The Transformer

Encoder-Decoder

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

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Model</th>
<th>URL</th>
<th>Score</th>
<th>CoLA</th>
<th>SST-2</th>
<th>MRPC</th>
<th>STS-B</th>
<th>QQP</th>
<th>MNLI-m</th>
<th>MNLI-mm</th>
<th>QNLI</th>
<th>RTE</th>
<th>WNLI</th>
<th>AX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HFL IFLYTEK</td>
<td>MacALBERT + DKM</td>
<td>90.7</td>
<td>74.8</td>
<td>97.0</td>
<td>94.5/92.6</td>
<td>92.8/92.6</td>
<td>74.7/90.6</td>
<td>91.3</td>
<td>91.1</td>
<td>97.8</td>
<td>92.0</td>
<td>94.5</td>
<td>52.6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Alibaba DAMO NLP</td>
<td>StructBERT + TAPT</td>
<td>90.6</td>
<td>75.3</td>
<td>97.3</td>
<td>93.9/91.9</td>
<td>93.2/92.7</td>
<td>74.8/91.0</td>
<td>90.9</td>
<td>90.7</td>
<td>97.4</td>
<td>91.2</td>
<td>94.5</td>
<td>49.1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>PING-AN Omni-Sinic</td>
<td>ALBERT + DAA + NAS</td>
<td>90.6</td>
<td>73.5</td>
<td>97.2</td>
<td>94.0/92.0</td>
<td>93.0/92.4</td>
<td>76.1/91.0</td>
<td>91.6</td>
<td>91.3</td>
<td>97.5</td>
<td>91.7</td>
<td>94.5</td>
<td>51.2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ERNIE Team - Baidu</td>
<td>ERNIE</td>
<td>90.4</td>
<td>74.4</td>
<td>97.5</td>
<td>93.5/91.4</td>
<td>93.0/92.6</td>
<td>75.2/90.9</td>
<td>91.4</td>
<td>91.0</td>
<td>96.6</td>
<td>90.9</td>
<td>94.5</td>
<td>51.7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>T5 Team - Google</td>
<td>T5</td>
<td>90.3</td>
<td>71.6</td>
<td>97.5</td>
<td>92.8/90.4</td>
<td>93.1/92.8</td>
<td>75.1/90.6</td>
<td>92.2</td>
<td>91.9</td>
<td>96.9</td>
<td>92.8</td>
<td>94.5</td>
<td>53.1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Microsoft D365 AI &amp; MSR AI &amp; GATECH</td>
<td>MT-DNN-SMART</td>
<td>89.9</td>
<td>69.5</td>
<td>97.5</td>
<td>93.7/91.6</td>
<td>92.9/92.5</td>
<td>73.9/90.2</td>
<td>91.0</td>
<td>90.8</td>
<td>99.2</td>
<td>89.7</td>
<td>94.5</td>
<td>50.2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Zihang Dai</td>
<td>Funnel-Transformer (Ensemble B10-10-10H1024)</td>
<td>89.7</td>
<td>70.5</td>
<td>97.5</td>
<td>93.4/91.2</td>
<td>92.6/92.3</td>
<td>75.4/90.7</td>
<td>91.4</td>
<td>91.1</td>
<td>95.8</td>
<td>90.0</td>
<td>94.5</td>
<td>51.6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ELECTRA Team</td>
<td>ELECTRA-Large + Standard Tricks</td>
<td>89.4</td>
<td>71.7</td>
<td>97.1</td>
<td>93.1/90.7</td>
<td>92.9/92.5</td>
<td>75.6/90.8</td>
<td>91.3</td>
<td>90.8</td>
<td>95.8</td>
<td>89.8</td>
<td>91.8</td>
<td>50.7</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Huawei Noah’s Ark Lab</td>
<td>NEZHA-Large</td>
<td>89.1</td>
<td>69.9</td>
<td>97.3</td>
<td>93.3/91.0</td>
<td>92.4/91.9</td>
<td>74.2/90.6</td>
<td>91.0</td>
<td>90.7</td>
<td>95.7</td>
<td>88.7</td>
<td>93.2</td>
<td>47.9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Microsoft D365 AI &amp; UMD</td>
<td>Frel,B-RoBERTa (ensemble)</td>
<td>88.4</td>
<td>68.0</td>
<td>96.8</td>
<td>93.1/90.8</td>
<td>92.3/92.1</td>
<td>74.8/90.3</td>
<td>91.1</td>
<td>90.7</td>
<td>95.6</td>
<td>88.7</td>
<td>89.0</td>
<td>50.1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Junjie Yang</td>
<td>HIRE-RoBERTa</td>
<td>88.3</td>
<td>68.6</td>
<td>97.1</td>
<td>93.0/90.7</td>
<td>92.4/92.0</td>
<td>74.3/90.2</td>
<td>90.7</td>
<td>90.4</td>
<td>95.5</td>
<td>87.9</td>
<td>89.0</td>
<td>49.3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Facebook AI</td>
<td>RoBERTa</td>
<td>88.1</td>
<td>67.8</td>
<td>96.7</td>
<td>92.3/89.8</td>
<td>92.2/91.9</td>
<td>74.3/90.2</td>
<td>90.8</td>
<td>90.2</td>
<td>95.4</td>
<td>88.2</td>
<td>89.0</td>
<td>48.7</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Microsoft D365 AI &amp; MSR AI</td>
<td>MT-DNN-ensemble</td>
<td>87.6</td>
<td>68.4</td>
<td>96.5</td>
<td>92.7/90.3</td>
<td>91.1/90.7</td>
<td>73.7/89.9</td>
<td>87.9</td>
<td>87.4</td>
<td>96.0</td>
<td>86.3</td>
<td>89.0</td>
<td>42.8</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>GLUE Human Baselines</td>
<td>GLUE Human Baselines</td>
<td>87.1</td>
<td>66.4</td>
<td>97.8</td>
<td>86.3/80.8</td>
<td>92.7/92.6</td>
<td>59.5/80.4</td>
<td>92.0</td>
<td>92.8</td>
<td>91.2</td>
<td>93.6</td>
<td>95.9</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Stanford Hazy Research</td>
<td>Snokkel MeTaL</td>
<td>83.2</td>
<td>63.8</td>
<td>96.2</td>
<td>91.5/88.5</td>
<td>90.1/89.7</td>
<td>73.1/89.9</td>
<td>87.6</td>
<td>87.2</td>
<td>93.9</td>
<td>80.9</td>
<td>65.1</td>
<td>39.9</td>
<td></td>
</tr>
</tbody>
</table>

source: https://gluebenchmark.com/leaderboard
<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Model</th>
<th>URL</th>
<th>Score</th>
<th>CoLA</th>
<th>SST-2</th>
<th>MRPC</th>
<th>STS-B</th>
<th>QQP</th>
<th>MNLI-m</th>
<th>MNLI-mm</th>
<th>QNLI</th>
<th>RTE</th>
<th>WNI</th>
<th>AX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HFL IFLYTEK</td>
<td>MacALBERT + DKM</td>
<td></td>
<td>90.7</td>
<td>74.8</td>
<td>97.0</td>
<td>94.5/92.6</td>
<td>92.8/92.6</td>
<td>74.7/90.6</td>
<td>91.1</td>
<td>91.1</td>
<td>97.8</td>
<td>92.0</td>
<td>94.5</td>
<td>52.6</td>
</tr>
<tr>
<td>2</td>
<td>Alibaba DAMO NLP</td>
<td>StructBERT + TAPT</td>
<td></td>
<td>90.6</td>
<td>75.3</td>
<td>97.3</td>
<td>93.9/91.9</td>
<td>93.2/92.7</td>
<td>74.8/91.0</td>
<td>90.9</td>
<td>90.7</td>
<td>97.4</td>
<td>91.2</td>
<td>94.5</td>
<td>49.1</td>
</tr>
<tr>
<td>3</td>
<td>PING-AN Omni-Sinitic</td>
<td>ALBERT + DAAF + NAS</td>
<td></td>
<td>90.6</td>
<td>73.5</td>
<td>97.2</td>
<td>94.0/92.0</td>
<td>93.0/92.4</td>
<td>76.1/91.0</td>
<td>91.6</td>
<td>91.3</td>
<td>97.5</td>
<td>91.7</td>
<td>94.5</td>
<td>51.2</td>
</tr>
<tr>
<td>4</td>
<td>ERNIE Team - Baidu</td>
<td>ERNIE</td>
<td></td>
<td>90.4</td>
<td>74.4</td>
<td>97.5</td>
<td>93.5/91.4</td>
<td>93.0/92.6</td>
<td>75.2/90.9</td>
<td>91.4</td>
<td>91.0</td>
<td>96.6</td>
<td>90.9</td>
<td>94.5</td>
<td>51.7</td>
</tr>
<tr>
<td>5</td>
<td>T5 Team - Google</td>
<td>T5</td>
<td></td>
<td>90.3</td>
<td>71.6</td>
<td>97.5</td>
<td>92.8/90.4</td>
<td>93.1/92.8</td>
<td>75.1/90.6</td>
<td>92.2</td>
<td>91.9</td>
<td>96.9</td>
<td>92.8</td>
<td>94.5</td>
<td>53.1</td>
</tr>
<tr>
<td>6</td>
<td>Microsoft D365 AI &amp; GATECH</td>
<td>MT-DNN-SMART</td>
<td></td>
<td>89.9</td>
<td>69.5</td>
<td>97.5</td>
<td>93.7/91.6</td>
<td>92.9/92.5</td>
<td>73.9/90.2</td>
<td>91.0</td>
<td>90.8</td>
<td>99.2</td>
<td>89.7</td>
<td>94.5</td>
<td>50.2</td>
</tr>
<tr>
<td>7</td>
<td>Zihang Dai</td>
<td>Funnel-Transformer (Ensemble B10-10-10H1024)</td>
<td></td>
<td>89.7</td>
<td>70.5</td>
<td>97.5</td>
<td>93.1/91.2</td>
<td>92.6/92.3</td>
<td>75.4/90.7</td>
<td>91.4</td>
<td>91.1</td>
<td>95.8</td>
<td>90.0</td>
<td>94.5</td>
<td>51.6</td>
</tr>
<tr>
<td>8</td>
<td>ELECTRA Team</td>
<td>ELECTRA-Large + Standard Tricks</td>
<td></td>
<td>89.4</td>
<td>71.7</td>
<td>97.1</td>
<td>93.1/90.7</td>
<td>92.9/92.5</td>
<td>75.6/90.8</td>
<td>91.3</td>
<td>90.8</td>
<td>95.8</td>
<td>89.8</td>
<td>91.8</td>
<td>50.7</td>
</tr>
<tr>
<td>9</td>
<td>Huawei Noah’s Ark Lab</td>
<td>NEZHA-Large</td>
<td></td>
<td>89.1</td>
<td>69.9</td>
<td>97.3</td>
<td>93.3/91.0</td>
<td>92.4/91.9</td>
<td>74.2/90.6</td>
<td>91.0</td>
<td>90.7</td>
<td>95.7</td>
<td>88.7</td>
<td>93.2</td>
<td>47.9</td>
</tr>
<tr>
<td>10</td>
<td>Microsoft D365 AI &amp; UMD</td>
<td>Freel.B-RoBERTa (ensemble)</td>
<td></td>
<td>88.4</td>
<td>68.0</td>
<td>96.8</td>
<td>93.1/90.8</td>
<td>92.3/92.1</td>
<td>74.8/90.3</td>
<td>91.1</td>
<td>90.7</td>
<td>95.6</td>
<td>88.7</td>
<td>89.0</td>
<td>50.1</td>
</tr>
<tr>
<td>11</td>
<td>Junjie Yang</td>
<td>HIRE-RoBERTa</td>
<td></td>
<td>88.3</td>
<td>68.6</td>
<td>97.1</td>
<td>93.0/90.7</td>
<td>92.4/92.0</td>
<td>74.3/90.2</td>
<td>90.7</td>
<td>90.4</td>
<td>95.5</td>
<td>87.9</td>
<td>89.0</td>
<td>49.3</td>
</tr>
<tr>
<td>12</td>
<td>Facebook AI</td>
<td>RoBERTa</td>
<td></td>
<td>88.1</td>
<td>67.8</td>
<td>96.7</td>
<td>92.3/89.8</td>
<td>92.2/91.9</td>
<td>74.3/90.2</td>
<td>90.8</td>
<td>90.2</td>
<td>95.4</td>
<td>88.2</td>
<td>89.0</td>
<td>48.7</td>
</tr>
<tr>
<td>13</td>
<td>Microsoft D365 AI &amp; MSR AI</td>
<td>MT-DNN-ensemble</td>
<td></td>
<td>87.6</td>
<td>68.4</td>
<td>96.5</td>
<td>92.7/90.3</td>
<td>91.1/90.7</td>
<td>73.7/89.9</td>
<td>87.9</td>
<td>87.4</td>
<td>96.0</td>
<td>86.3</td>
<td>89.0</td>
<td>42.8</td>
</tr>
<tr>
<td>14</td>
<td>GLUE Human Baselines</td>
<td>GLUE Human Baselines</td>
<td></td>
<td>87.1</td>
<td>66.4</td>
<td>97.8</td>
<td>86.3/80.8</td>
<td>92.7/92.6</td>
<td>59.5/80.4</td>
<td>92.0</td>
<td>92.8</td>
<td>91.2</td>
<td>93.6</td>
<td>95.9</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>Stanford Hazy Research</td>
<td>Snorkel MeTaL</td>
<td></td>
<td>83.2</td>
<td>63.8</td>
<td>96.2</td>
<td>91.5/88.5</td>
<td>90.1/89.7</td>
<td>73.1/89.9</td>
<td>87.6</td>
<td>87.2</td>
<td>93.9</td>
<td>80.9</td>
<td>65.1</td>
<td>39.9</td>
</tr>
</tbody>
</table>

source: https://gluebenchmark.com/leaderboard
In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. “By the time we reached the top of one peak, the water looked blue, with some crystals on top,” said Pérez.

Source: OpenAI, “Better Language Models and Their Implications”  
https://openai.com/blog/better-language-models/
Can Attention/Transformers be used from more than text processing?
ViLBERT: A Visolinguistic Transformer

- pop artist performs at the festival in a city.
- a worker helps to clear the debris.
- blue sofa in the living room.

ViLBERT: A Visolinguistic Transformer


blue sofa in the living room.
ViLBERT Demo:
https://vilbert.cloudcv.org/
Summary

Self-Attention

Transformer Model

ViLBERT