Introduction to Transformers
LLMs are built out of transformers

Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

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

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin‡
illia.polosukhin@gmail.com

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.
A very approximate timeline

1990 Static Word Embeddings
2003 Neural Language Model
2008 Multi-Task Learning
2015 Attention
2017 Transformer
2018 Contextual Word Embeddings and Pretraining
2019 Prompting
Transformers

Attention
Instead of starting with the big picture

Let's consider the embeddings for an individual word from a particular layer
Problem with static embeddings (word2vec)

They are static! The embedding for a word doesn't reflect how its meaning changes in context.

The chicken didn't cross the road because it was too tired

What is the meaning represented in the static embedding for "it"?
Contextual Embeddings

• Intuition: a representation of meaning of a word should be different in different contexts!

• **Contextual Embedding**: each word has a different vector that expresses different meanings depending on the surrounding words

• How to compute contextual embeddings?
  • **Attention**
Contextual Embeddings

The chicken didn't cross the road because it

What should be the properties of "it"?

The chicken didn't cross the road because it was too tired
The chicken didn't cross the road because it was too wide

At this point in the sentence, it's probably referring to either the chicken or the street
Intuition of attention

Build up the contextual embedding from a word by selectively integrating information from all the neighboring words.

We say that a word "attends to" some neighboring words more than others.
Intuition of attention:

The chicken didn’t cross the road because it was too tired.

Layer k+1

self-attention distribution

Layer k
Attention definition

A mechanism for helping compute the embedding for a token by selectively attending to and integrating information from surrounding tokens (at the previous layer).

More formally: a method for doing a weighted sum of vectors.
Attention is left-to-right
Simplified version of attention: a sum of prior words weighted by their similarity with the current word

Given a sequence of token embeddings:

\[ \mathbf{x}_1 \quad \mathbf{x}_2 \quad \mathbf{x}_3 \quad \mathbf{x}_4 \quad \mathbf{x}_5 \quad \mathbf{x}_6 \quad \mathbf{x}_7 \quad \mathbf{x}_i \]

Produce: \( \mathbf{a}_i = \) a weighted sum of \( \mathbf{x}_1 \) through \( \mathbf{x}_7 \) (and \( \mathbf{x}_i \))

Weighted by their similarity to \( \mathbf{x}_i \)

\[
\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \\
\alpha_{ij} = \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \quad \forall j \leq i \\
\mathbf{a}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j
\]
Intuition of attention:

Layer k:
- The
- chicken
- didn’t
- cross
- the
- road
- because
- it

Layer k+1:
- The
- chicken
- didn’t
- cross
- the
- road
- because
- it

self-attention distribution

columns corresponding to input tokens
An Actual Attention Head: slightly more complicated

High-level idea: instead of using vectors (like $x_i$ and $x_4$) directly, we'll represent 3 separate roles each vector $x_i$ plays:

- **query**: As the current element being compared to the preceding inputs.
- **key**: as a preceding input that is being compared to the current element to determine a similarity
- **value**: a value of a preceding element that gets weighted and summed
The chicken didn’t cross the road because it was too tired.

Attention intuition

Layer k+1

Layer k

x1  x2  x3  x4  x5  x6  x7  xi

self-attention distribution

values

query

The  chicken  didn’t  cross  the  road  because  it

was  too  tired.
Intuition of attention:

Layer k:
- The
- chicken
- didn’t
- cross
- the
- road
- because
- it

Layer k+1:
- was
- too
- tired

self-attention distribution:

Keys: x1 x2 x3 x4 x5 x6 x7 xi

Values: k v k v k v k v k v k v k v k v k v k v

Query: it

The chicken didn’t cross the road because it was too tired.
An Actual Attention Head: slightly more complicated

We'll use matrices to project each vector $x_i$ into a representation of its role as query, key, value:

- **query**: $W^Q$
- **key**: $W^K$
- **value**: $W^V$

\[
q_i = x_i W^Q; \quad k_i = x_i W^K; \quad v_i = x_i W^V
\]
An Actual Attention Head: slightly more complicated

Given these 3 representation of $x_i$

$$q_i = x_i W^Q; \quad k_i = x_i W^K; \quad v_i = x_i W^V$$

To compute similarity of current element $x_i$ with some prior element $x_j$

We’ll use dot product between $q_i$ and $k_j$.

And instead of summing up $x_j$, we'll sum up $v_j$
Final equations for one attention head

\[ q_i = x_i W^Q; \quad k_j = x_j W^K; \quad v_j = x_j W^V \]

\[ \text{score}(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}} \]

\[ \alpha_{i,j} = \text{softmax}(\text{score}(x_i, x_j)) \quad \forall j \leq i \]

\[ a_i = \sum_{j \leq i} \alpha_{i,j} v_j \]
Calculating the value of $a_3$

1. Generate key, query, value vectors
2. Compare $x_3$’s query with the keys for $x_1$, $x_2$, and $x_3$
3. Divide score by $\sqrt{d_k}$
4. Turn into $\alpha_{i,j}$ weights via softmax
5. Weigh each value vector
6. Sum the weighted value vectors

Output of self-attention

$\alpha_{3,1}$ $\alpha_{3,2}$ $\alpha_{3,3}$
Actual Attention: slightly more complicated

- Instead of one attention head, we'll have lots of them!
- Intuition: each head might be attending to the context for different purposes
  - Different linguistic relationships or patterns in the context

\[
\begin{align*}
q^c_i &= x_i W^Q^c, \quad k^c_j &= x_j W^K^c, \quad v^c_j &= x_j W^V^c, \quad \forall \ c \ 1 \leq c \leq h \\
\text{score}^c(x_i, x_j) &= \frac{q^c_i \cdot k^c_j}{\sqrt{d_k}} \\
\alpha^c_{i,j} &= \text{softmax}(\text{score}^c(x_i, x_j)) \quad \forall j \leq i \\
\text{head}^c_i &= \sum_{j \leq i} \alpha^c_{i,j} v^c_j \\
a_i &= (\text{head}^1 \oplus \text{head}^2 \ldots \oplus \text{head}^h) W^O \\
\text{MultiHeadAttention}(x_i, [x_1, \ldots, x_N]) &= a_i
\end{align*}
\]
Multi-head attention

Project down to d
Concatenate Outputs
Each head attends differently to context

... $x_{i-3}$ $x_{i-2}$ $x_{i-1}$
Summary

Attention is a method for enriching the representation of a token by incorporating contextual information.

The result: the embedding for each word will be different in different contexts!

Contextual embeddings: a representation of word meaning in its context.

We'll see in the next lecture that attention can also be viewed as a way to move information from one token to another.
Attention

Transformers
The Transformer Block

Transformers
Reminder: transformer language model

Next token
Language Modeling Head
Stacked Transformer Blocks
Input Encoding
Input tokens

So long and thanks for all

long and thanks for all
The residual stream: each token gets passed up and modified

\[
x_i + h_{i-1}
\]

\[
x_i \rightarrow \text{Layer Norm} \rightarrow \text{MultiHead Attention} \rightarrow \text{Feedforward} \rightarrow h_i
\]

\[
x_{i+1}
\]

\[
x_{i-1}
\]

\[
x_i \rightarrow \text{Layer Norm} \rightarrow h_i
\]
We'll need nonlinearities, so a feedforward layer

\[ \text{FFN}(x_i) = \text{ReLU}(x_iW_1 + b_1)W_2 + b_2 \]
Layer norm: the vector $x_i$ is normalized twice
Layer Norm

Layer norm is a variation of the z-score from statistics, applied to a single vector in a hidden layer.

\[
\begin{align*}
\mu &= \frac{1}{d} \sum_{i=1}^{d} x_i \\
\sigma &= \sqrt{\frac{1}{d} \sum_{i=1}^{d} (x_i - \mu)^2} \\
\hat{x} &= \frac{x - \mu}{\sigma} \\
\text{LayerNorm}(x) &= \gamma \frac{x - \mu}{\sigma} + \beta
\end{align*}
\]
Putting together a single transformer block

\[ t_i^1 = \text{LayerNorm}(x_i) \]
\[ t_i^2 = \text{MultiHeadAttention}(t_i^1, [x_1^1, \cdots, x_N^1]) \]
\[ t_i^3 = t_i^2 + x_i \]
\[ t_i^4 = \text{LayerNorm}(t_i^3) \]
\[ t_i^5 = \text{FFN}(t_i^4) \]
\[ h_i = t_i^5 + t_i^3 \]
A transformer is a stack of these blocks so all the vectors are of the same dimensionality $d$. 

Block 1

Block 2
Residual streams and attention

Notice that all parts of the transformer block apply to 1 residual stream (1 token).

Except attention, which takes information from other tokens

*Elhage et al. (2021)* show that we can view attention heads as literally moving information from the residual stream of a neighboring token into the current stream.
The Transformer Block
Parallelizing Attention Computation

Transformers
Parallelizing computation using $X$

For attention/transformer block we've been computing a single output at a single time step $i$ in a single residual stream. But we can pack the $N$ tokens of the input sequence into a single matrix $X$ of size $[N \times d]$.

Each row of $X$ is the embedding of one token of the input. $X$ can have 1K-32K rows, each of the dimensionality of the embedding $d$ (the model dimension).

\[
Q = XW^Q; \quad K = XW^K; \quad V = XW^V
\]
Now can do a single matrix multiply to combine $Q$ and $K^T$

$$\begin{array}{cccc}
q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 & q_1 \cdot k_4 \\
q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 & q_2 \cdot k_4 \\
q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 & q_3 \cdot k_4 \\
q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 & q_4 \cdot k_4 \\
\end{array}$$
Parallelizing attention

- Scale the scores, take the softmax, and then multiply the result by V resulting in a matrix of shape $N \times d$
- An attention vector for each input token

\[
A = \text{softmax} \left( \text{mask} \left( \frac{QK^T}{\sqrt{d_k}} \right) \right)V
\]
Masking out the future

\[ A = \text{softmax} \left( \text{mask} \left( \frac{QK^T}{\sqrt{d_k}} \right) \right) V \]

• What is this mask function? 
  \( QK^T \) has a score for each query dot every key, including those that follow the query.

• Guessing the next word is pretty simple if you already know it!
Masking out the future

\[ A = \text{softmax} \left( \text{mask} \left( \frac{QK^T}{\sqrt{d_k}} \right) \right) V \]

Add \(-\infty\) to cells in upper triangle

The softmax will turn it to 0

<table>
<thead>
<tr>
<th>q1·k1</th>
<th>q2·k1</th>
<th>q3·k1</th>
<th>q4·k1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-\infty</td>
<td>q2·k2</td>
<td>q3·k2</td>
<td>q4·k2</td>
</tr>
<tr>
<td>-\infty</td>
<td>-\infty</td>
<td>q3·k3</td>
<td>q4·k3</td>
</tr>
<tr>
<td>-\infty</td>
<td>-\infty</td>
<td>-\infty</td>
<td>-\infty</td>
</tr>
</tbody>
</table>
Another point: Attention is quadratic in length

$$A = \text{softmax} \left( \text{mask} \left( \frac{QK^T}{\sqrt{d_k}} \right) \right) V$$

Figure 9.8 shows the resulting masked $QK^T$ matrix.
Attention again

\[
\begin{align*}
X & \quad W^Q \quad Q \\
\text{Input Token 1} & \quad \text{Query Token 1} \\
\text{Input Token 2} & \quad \text{Query Token 2} \\
\text{Input Token 3} & \quad \text{Query Token 3} \\
\text{Input Token 4} & \quad \text{Query Token 4} \\
N \times d & \quad N \times d_k \\
\end{align*}
\]

\[
\begin{align*}
X & \quad W^K \quad K \\
\text{Input Token 1} & \quad \text{Key Token 1} \\
\text{Input Token 2} & \quad \text{Key Token 2} \\
\text{Input Token 3} & \quad \text{Key Token 3} \\
\text{Input Token 4} & \quad \text{Key Token 4} \\
N \times d & \quad N \times d_k \\
\end{align*}
\]

\[
\begin{align*}
X & \quad W^V \quad V \\
\text{Value Token 1} & \quad \text{Value Token 2} \\
\text{Value Token 3} & \quad \text{Value Token 4} \\
N \times d & \quad N \times d_v \\
\end{align*}
\]

\[
\begin{align*}
Q & \quad K^T \\
q_1 & \quad k_1 \\
q_2 & \quad k_2 \\
q_3 & \quad k_3 \\
q_4 & \quad k_4 \\
d_k \times N & \quad N \times N \\
\end{align*}
\]

\[
\begin{align*}
QK^T & = Qk_1, \ldots, q_k, k_4, q_k \\
q_1 \cdot k_1 & \quad q_1 \cdot k_2 \quad q_1 \cdot k_3, \quad q_1 \cdot k_4 \\
q_2 \cdot k_1 & \quad q_2 \cdot k_2 \quad q_2 \cdot k_3, \quad q_2 \cdot k_4 \\
q_3 \cdot k_1 & \quad q_3 \cdot k_2 \quad q_3 \cdot k_3, \quad q_3 \cdot k_4 \\
q_4 \cdot k_1 & \quad q_4 \cdot k_2 \quad q_4 \cdot k_3, \quad q_4 \cdot k_4 \\
N \times N & \quad N \times N \\
\end{align*}
\]

\[
\begin{align*}
V & \quad A \\
v_1 & \quad a_1 \\
v_2 & \quad a_2 \\
v_3 & \quad a_3 \\
v_4 & \quad a_4 \\
N \times d_v & \quad N \times d_v \\
\end{align*}
\]
Parallelizing Multi-head Attention

\[ Q^i = XW^{Q_i}; \quad K^i = XW^{K_i}; \quad V^i = XW^{V_i} \]

\[ \text{head}_i = \text{SelfAttention}(Q^i, K^i, V^i) = \text{softmax} \left( \frac{Q^iK^i}{\sqrt{d_k}} \right) V^i \]

\[ \text{MultiHeadAttention}(X) = (\text{head}_1 \oplus \text{head}_2 \ldots \oplus \text{head}_h)W^{O} \]
Parallelizing Multi-head Attention

\[
O = \text{LayerNorm}(X + \text{MultiHeadAttention}(X))
\]
\[
H = \text{LayerNorm}(O + \text{FFN}(O))
\]

or

\[
T^1 = \text{MultiHeadAttention}(X)
\]
\[
T^2 = X + T^1
\]
\[
T^3 = \text{LayerNorm}(T^2)
\]
\[
T^4 = \text{FFN}(T^3)
\]
\[
T^5 = T^4 + T^3
\]
\[
H = \text{LayerNorm}(T^5)
\]
Parallelizing Attention Computation
Transformers

Input and output: Position embeddings and the Language Model Head
Token and Position Embeddings

The matrix $X$ (of shape $[N \times d]$) has an embedding for each word in the context. This embedding is created by adding two distinct embedding for each input

- token embedding
- positional embedding
Token Embeddings

Embedding matrix $E$ has shape $[|V| \times d]$.  
- One row for each of the $|V|$ tokens in the vocabulary.
- Each word is a row vector of $d$ dimensions

Given: string "Thanks for all the"

1. Tokenize with BPE and convert into vocab indices
   
   $$w = [5,4000,10532,2224]$$

2. Select the corresponding rows from $E$, each row an embedding
   
   (row 5, row 4000, row 10532, row 2224).
Position Embeddings

There are many methods, but we'll just describe the simplest: absolute position.

Goal: learn a position embedding matrix $E_{pos}$ of shape $[1 \times N]$.

Start with randomly initialized embeddings

• one for each integer up to some maximum length.

• i.e., just as we have an embedding for token *fish*, we’ll have an embedding for position 3 and position 17.

• As with word embeddings, these position embeddings are learned along with other parameters during training.
Each $x$ is just the sum of word and position embeddings

$$X = \text{Composite Embeddings} \quad \text{(word + position)}$$

Transforming Block

Word Embeddings

Position Embeddings
Language modeling head

**Language Model Head**

takes $h^L_N$ and outputs a distribution over vocabulary $V$

Layer $L$ Transformer Block

<table>
<thead>
<tr>
<th>$h^L_1$</th>
<th>$h^L_2$</th>
<th>...</th>
<th>$h^L_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1$</td>
<td>$w_2$</td>
<td>...</td>
<td>$w_N$</td>
</tr>
</tbody>
</table>

Softmax over vocabulary $V$

Logits $1 \times |V|$  

Word probabilities $1 \times |V|$  

Unembedding layer $d \times |V|$
Language modeling head

**Unembedding layer:** linear layer projects from $h^L_N$ (shape $[1 \times d]$) to logit vector

Why "unembedding"? **Tied** to $E^T$

**Weight tying**, we use the same weights for two different matrices

Unembedding layer maps from an embedding to a $1 \times |V|$ vector of logits
Language modeling head

**Logits**, the score vector $u$

One score for each of the $|V|$ possible words in the vocabulary $V$. Shape $1 \times |V|$.

**Softmax** turns the logits into probabilities over vocabulary. Shape $1 \times |V|$.

$$u = h_N^L E^T$$

$$y = \text{softmax}(u)$$
The final transformer model

Language Modeling Head

softmax

logits

$y_1$, $y_2$, ..., $y_{|V|}$

Token probabilities

Sample token to generate at position $i+1$

$w_{i+1}$

Input token

$w_i$

Input Encoding

$E$
Input and output: Position embeddings and the Language Model Head