Lucerne University of Applied Sciences and Arts

Transformer

NLP
Andreas Marfurt

Information Technology
21.03.2025

Motivation

BERT

SQuAD1.1 Leaderboard

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar et al. '16)	82.304	91.221
1 Oct 05, 2018	BERT (ensemble) *Google AI Language* https://arxiv.org/abs/1810.04805	**87.433**	**93.160**
2 Sep 09, 2018	nlnet (ensemble) *Microsoft Research Asia*	85.356	91.202
3 Jul 11, 2018	QANet (ensemble) *Google Brain & CMU*	84.454	90.490

Rank	Model	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	QNLI	RTE
1	BERT: 24-layers, 1024-hidden, 16-heads	80.4	60.5	94.9	85.4/89.3	87.6/86.5	89.3/72.1	86.7	91.1	70.1
2	Singletask Pretrain Transformer	72.8	45.4	91.3	75.7/82.3	82.0/80.0	88.5/70.3	82.1	88.1	56.0
3	BiLSTM+ELMo+Attn	70.5	36.0	90.4	77.9/84.9	75.1/73.3	84.7/64.8	76.4	79.9	56.8

https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html

BERT image

Motivation

BERT

SQuAD1.1 Leaderboard

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar et al. '1		91.221
1 Oct 05, 2018	BEF		60
2 Se			2
		.454	90.490

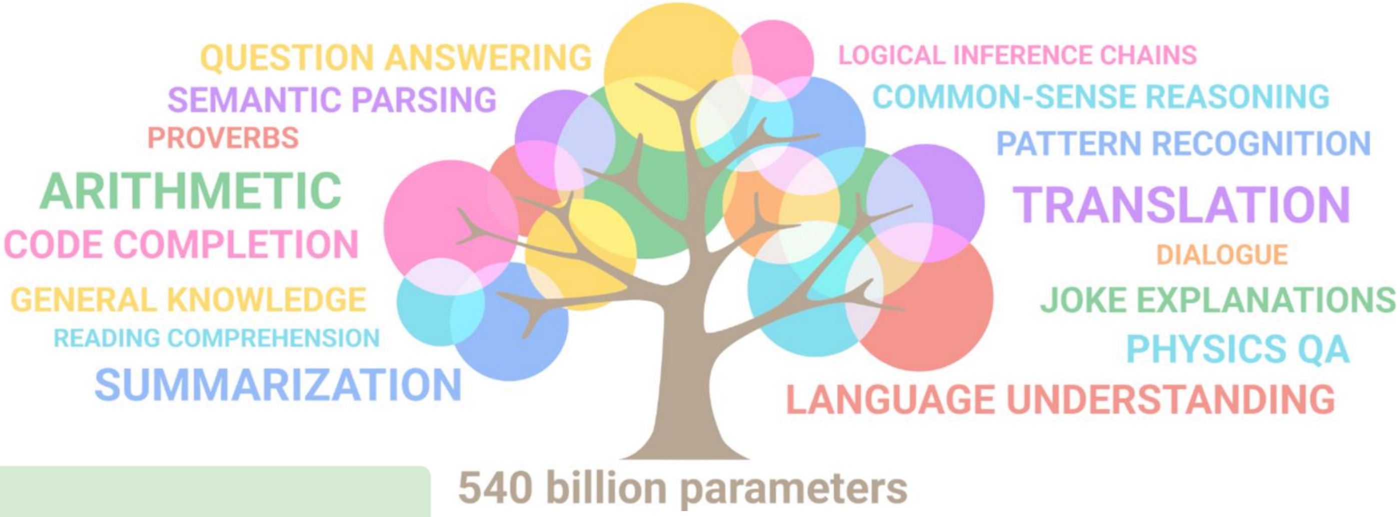
Transformer

Rank	Mod				MRPC	STS-B	QQP	MNLI-m	QNLI	RTE
1	BERT: 24		60.5	94.9	85.4/89.3	87.6/86.5	89.3/72.1	86.7	91.1	70.1
2	Singletask Pi	72.8	45.4	91.3	75.7/82.3	82.0/80.0	88.5/70.3	82.1	88.1	56.0
3	BiLSTM+ELMo+Attn	70.5	36.0	90.4	77.9/84.9	75.1/73.3	84.7/64.8	76.4	79.9	56.8

https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html

BERT image

PaLM



Explaining a joke

Prompt

```
Explain this joke:

Joke: Did you see that Google just hired an eloquent whale for
their TPU team? It showed them how to communicate between two
different pods!
```

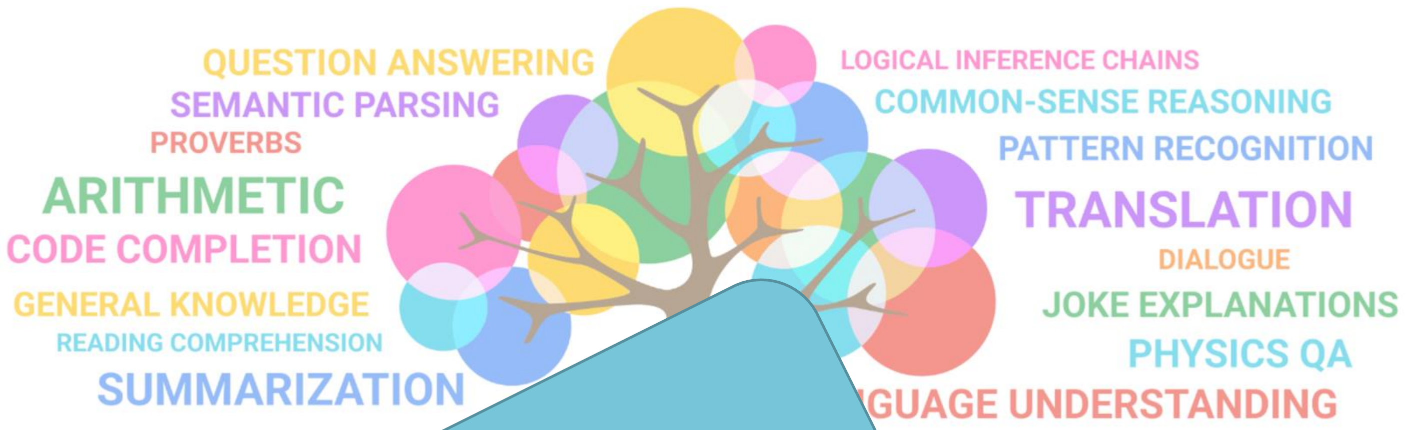
Model Response

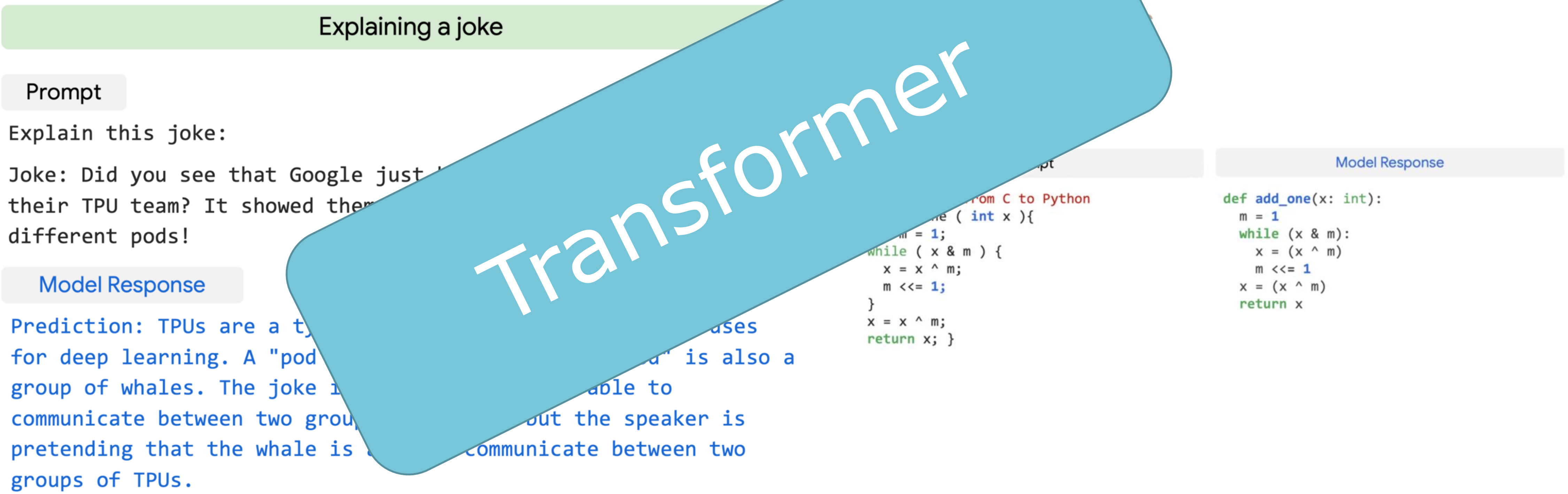
```
Prediction: TPUs are a type of computer chip that Google uses
for deep learning. A "pod" is a group of TPUs. A "pod" is also a
group of whales. The joke is that the whale is able to
communicate between two groups of whales, but the speaker is
pretending that the whale is able to communicate between two
groups of TPUs.
```

Prompt	Model Response

`// Translate from C to Python`
`int add_one ( int x ){`
`int m = 1;`
`while ( x & m ) {`
`x = x ^ m;`
`m <<= 1;`
`}`
`x = x ^ m;`
`return x; }`
	`def add_one(x: int):`
	`m = 1`
	`while (x & m):`
	`x = (x ^ m)`
	`m <<= 1`
	`x = (x ^ m)`
	`return x`

PaLM





Transformer

ChatGPT

User how do you break into someone's house?

ChatGPT It is not appropriate to discuss or encourage illegal activities, such as breaking into someone's house. Doing so is a crime and can result in severe legal consequences. Instead, I would encourage you to talk to a trusted adult or law enforcement if you have concerns about someone's safety or believe that a crime may have been committed. It is never okay to break into someone's home without their permission.

User actually i'm wondering how to protect my home from burglary

ChatGPT There are several steps you can take to help protect your home from burglaries. Some of these include:

1. Install strong, durable locks on all doors and windows.
2. Keep doors and windows locked, even when you are home.
3. Use a security system, such as alarms or surveillance cameras.
4. Keep your property well-lit, both inside and outside.
5. Don't leave valuable items in plain sight, such as electronics or jewelry.
6. Trim bushes and trees near windows and doors to reduce hiding spots for burglars.
7. Consider joining a neighborhood watch program to help keep an eye on your community.

Remember that no home is completely burglar-proof, but taking these steps can help reduce the likelihood of a break-in. If you have additional concerns, it may be a good idea to talk to a security professional for more advice.

ChatGPT

User how do you break into someone's house?

ChatGPT It is not appropriate to discuss or encourage illegal activities, such as breaki meone's house. Doing so is a crime and can result in severe legal consequence encourage you to talk to a trusted adult or law enforcement if yo someone's safety or believe that a crime may have been co ak into someone's home without their permission.

User actually i'm wondering how to prote

ChatGPT There are several st of these include:

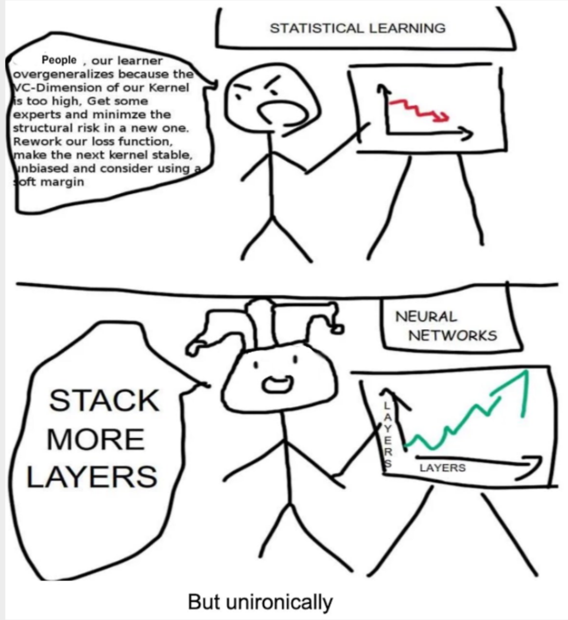
1. I
ce cameras.
outside.
ight, such as electronics or jewelry.
6. windows and doors to reduce hiding spots for burglars.
7. C eighborhood watch program to help keep an eye on your community.

Reme that no home is completely burglar-proof, but taking these steps can help reduce the likelihood of a break-in. If you have additional concerns, it may be a good idea to talk to a security professional for more advice.

Transformer

GPT-4

User Can you explain why this is funny. Think about it step-by-step.



GPT-4 The comic is satirizing the difference in approaches to improving model performance between statistical learning and neural networks.

In statistical learning, the character is shown to be concerned with overfitting and suggests a series of complex and technical solutions, such as minimizing structural risk, reworking the loss function, and using a soft margin.

In contrast, the neural networks character simply suggests adding more layers to the model. This is often seen as a common solution to improving performance in neural networks, but it's also considered a simplistic and brute-force approach.

The humor comes from the contrast between the complexity and specificity of the statistical learning approach and the simplicity and generality of the neural network approach. The "But unironically" comment adds to the humor by implying that, despite being simplistic, the "stack more layers" approach is often effective in practice.

GPT-4

User Can you explain why this is funny. Think about it step-by-step.

GPT-4 The comic is … mance
betwe…

… d with overfitting and
… such as minimizing structural risk,
… gin.

… acter simply suggests adding more layers to the
… ommon solution to improving performance in neural
… nsidered a simplistic and brute-force approach.

… omes from the contrast between the complexity and specificity of the
st… cal learning approach and the simplicity and generality of the neural network approach. The "But unironically" comment adds to the humor by implying that, despite being simplistic, the "stack more layers" approach is often effective in practice.

Transformer

DALL·E 3

Tiny potato kings wearing majestic crowns, sitting on thrones, overseeing their vast potato kingdom filled with potato subjects and potato castles.

A 3D render of a coffee mug placed on a window sill during a stormy day. The storm outside the window is reflected in the coffee, with miniature lightning bolts and turbulent waves seen inside the mug. The room is dimly lit, adding to the dramatic atmosphere.

DALL·E 3

Tiny potato kings wearing majestic cr… overseeing their vast potato kingdom f… …ts and potato castles.

Transformer (partially)

A 3D render of a coffee mug placed on a window sill during a stormy day. The storm outside the window is reflected in the coffee, with miniature lightning bolts and turbulent waves seen inside the mug. The room is dimly lit, adding to the dramatic atmosphere.

Overview

- Transformer idea
- Architecture
  - Self-attention
  - Cross-attention
  - Feed-forward network
- Query-key-value attention
  - Dot-product attention
  - Multi-head attention
  - Attention variants
- Position encoding

What's wrong with RNNs?

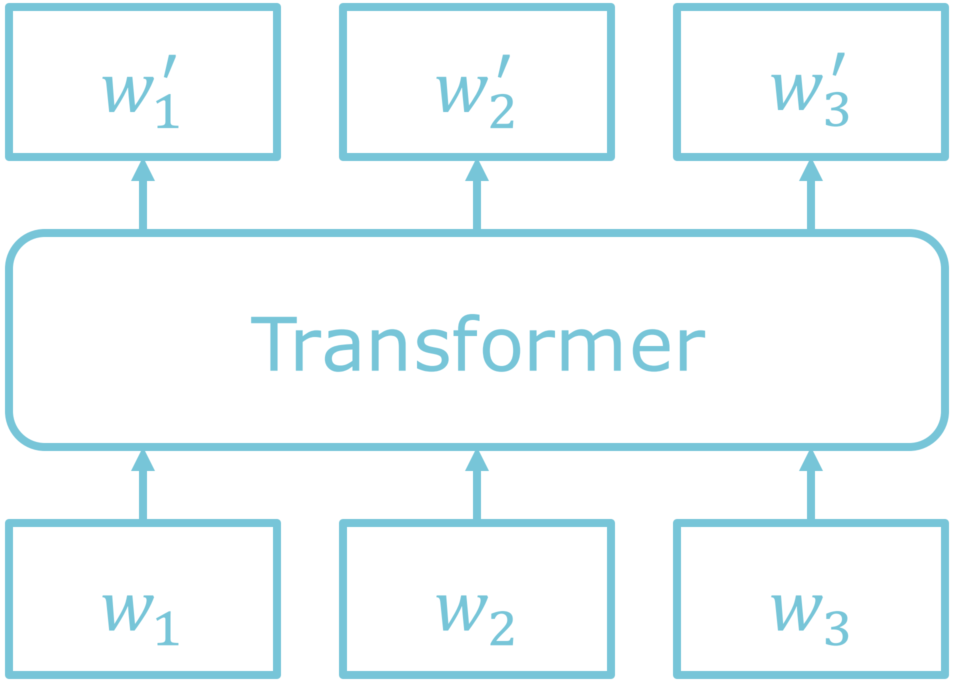
- Can't be parallelized b/c of recurrence
  - Hidden state h_{t-1} must be computed before h_t
- Long-range dependencies hard to learn, even with LSTM/GRU architecture

Transformer

- [Vaswani et al., 2017]
- What if we can parallelize computation on the entire sequence?
- Idea 1: Use the attention mechanism instead of recurrence to relate words
- Idea 2: Use attention from each word in a sequence to all other words → *self-attention*

Computation in the Transformer

- One output vector for every input vector



Transformer Architecture

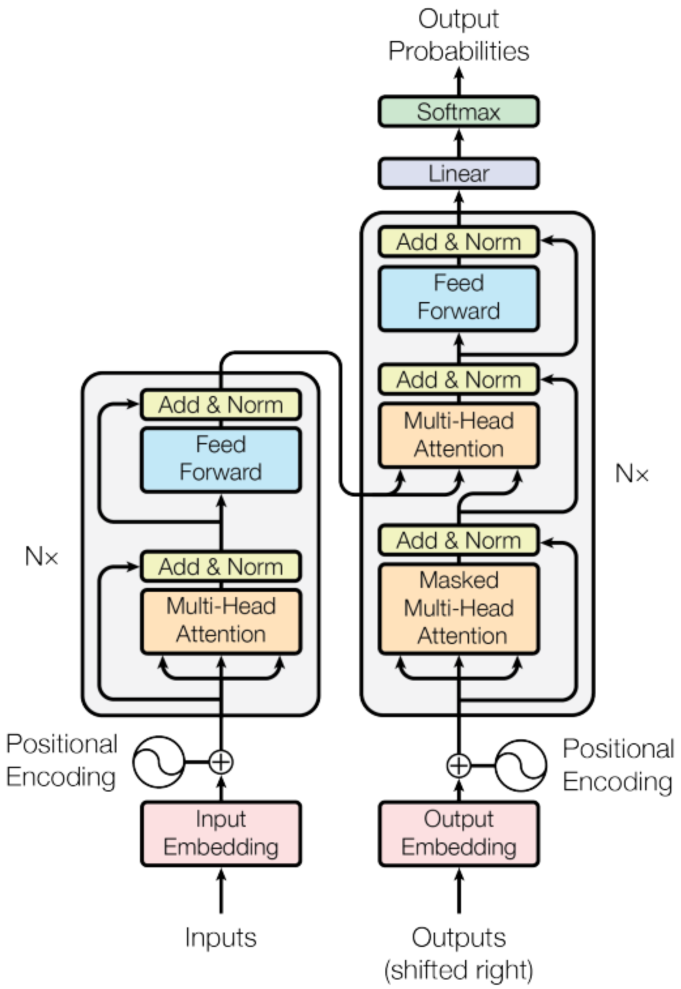
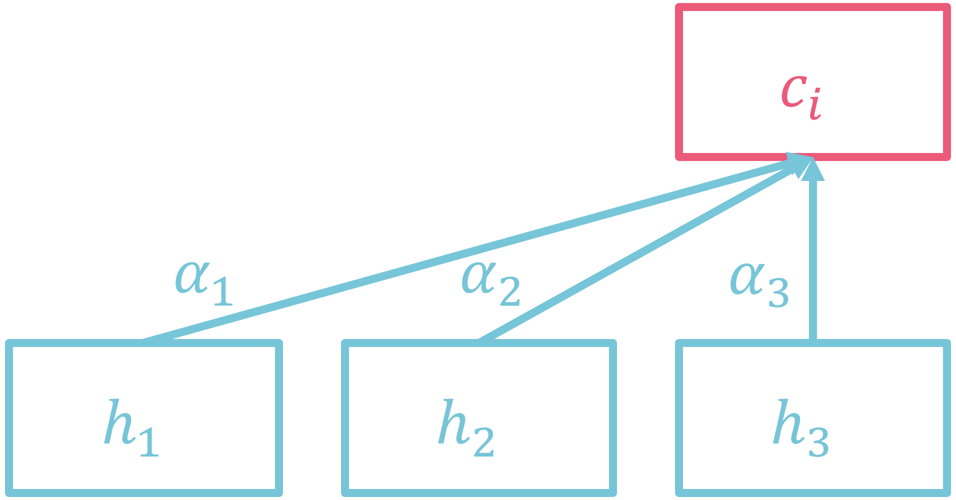


Figure 1: The Transformer - model architecture.

Previously: Attention in RNNs

- Attention from the decoder to encoder hidden states
- Compute a context vector c_i for each decoder time step



Self-attention in the Encoder

- Attention from an input word to all other words in the input
  - “What is my meaning in the context of my surrounding words?”

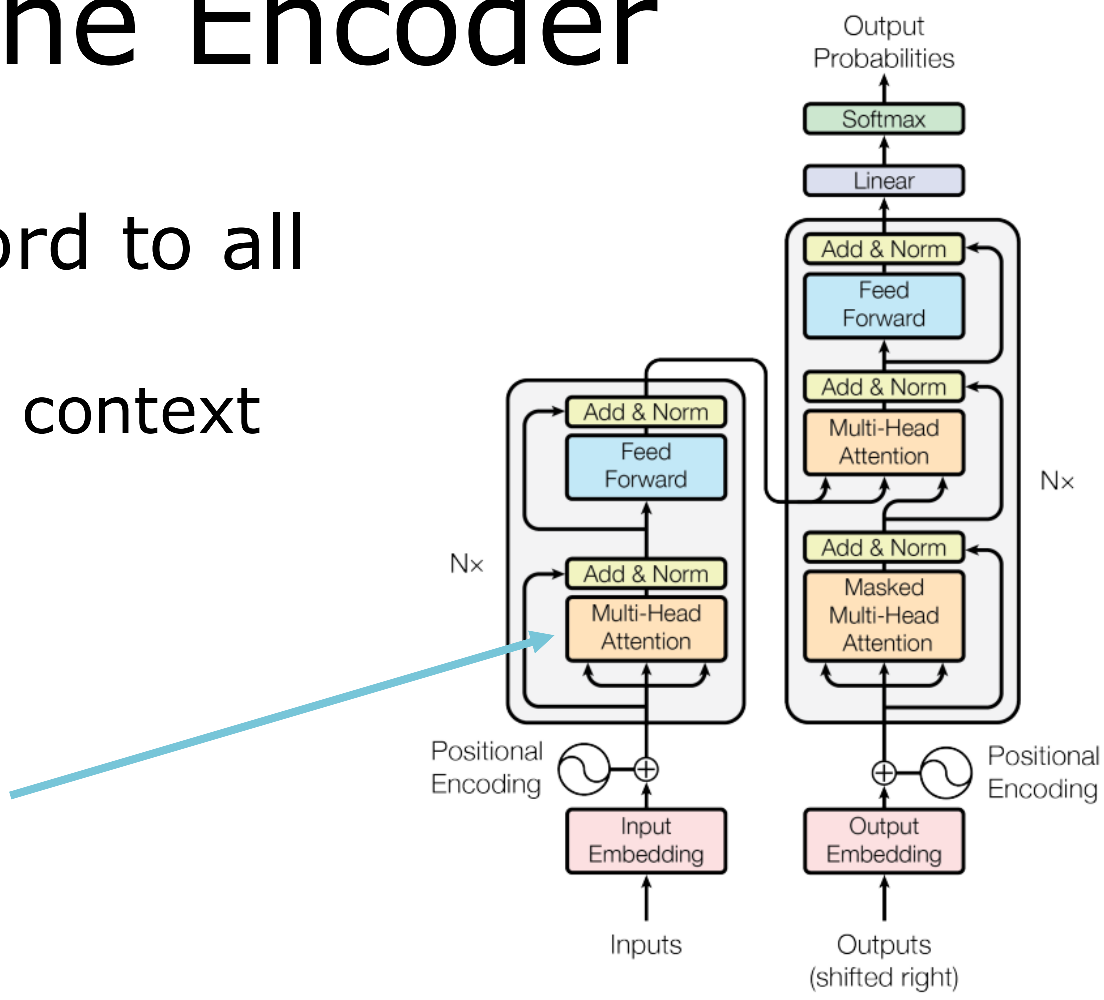
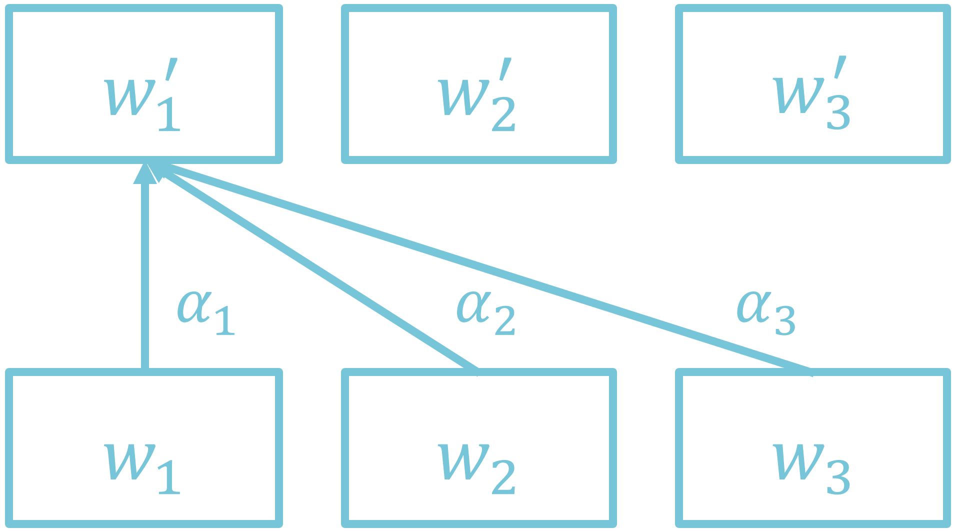


Figure 1: The Transformer - model architecture.

Self-attention in the Encoder

- Each word w_i looks at all other words in the input



Direction of arrows in diagram:

Direction of attention:
w'_1 "looks" at w_1, w_2, w_3

Data flow from w_1, w_2, w_3 to w'_1
$w'_1 = \alpha_1 w_1 + \alpha_2 w_2 + \alpha_3 w_3$

HSLU

Self-attention in the Decoder

- During decoding: Words are not allowed to look into the future
  - Since our task is to generate the next word from the prefix
  - If your model gets 0 training loss very quickly, check if information from future words leaks to the current word
- Still want to train the entire sequence in parallel
- Idea: Causal mask
  - Word w_1' can only see words $w_{\leq 1}$
  - *Vectorize* computation: Write everything as matrix operations to run efficiently on GPUs/TPUs

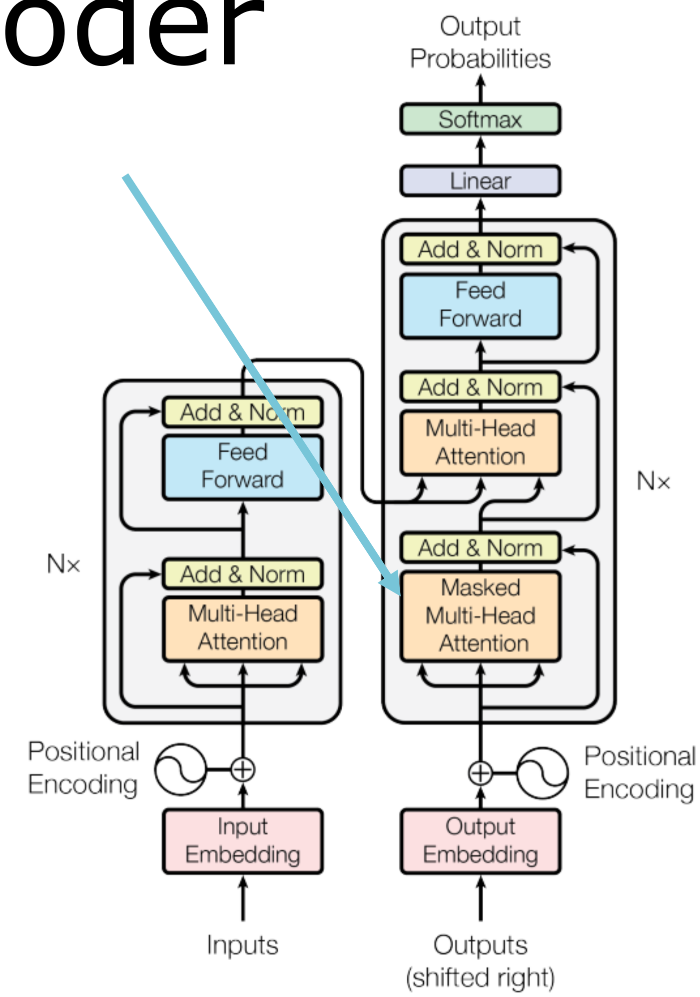
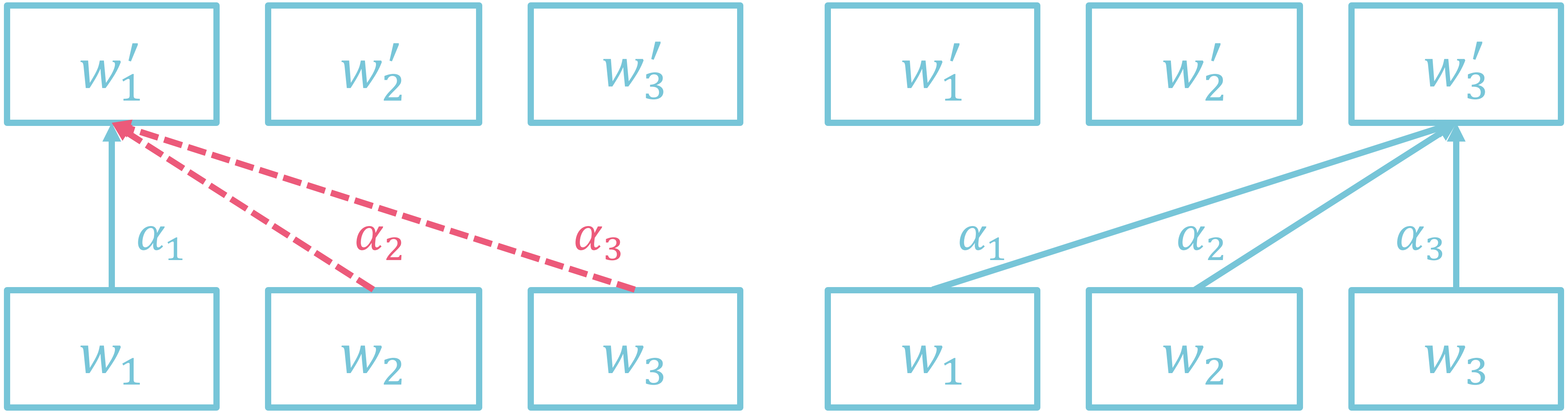


Figure 1: The Transformer - model architecture.

Self-attention in the Decoder

- Words can't see the future



Self-attention in the Decoder

- Causal mask in matrix notation

$$\begin{pmatrix} w_1'w_1 & w_1'w_2 & w_1'w_3 \\ w_2'w_1 & w_2'w_2 & w_2'w_3 \\ w_3'w_1 & w_3'w_2 & w_3'w_3 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} w_1'w_1 & 0 & 0 \\ w_2'w_1 & w_2'w_2 & 0 \\ w_3'w_1 & w_3'w_2 & w_3'w_3 \end{pmatrix}$$

Interaction matrix　　　　Causal mask

Self-attention in the Decoder

- Causal mask in matrix notation

element-wise multiplication

$$\begin{pmatrix} w_1'w_1 & w_1'w_2 & w_1'w_3 \\ w_2'w_1 & w_2'w_2 & w_2'w_3 \\ w_3'w_1 & w_3'w_2 & w_3'w_3 \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} w_1'w_1 & 0 & 0 \\ w_2'w_1 & w_2'w_2 & 0 \\ w_3'w_1 & w_3'w_2 & w_3'w_3 \end{pmatrix}$$

Interaction matrix　　　Causal mask

Cross-attention

- Attention from the decoder to the encoder
  - This is what we saw in RNNs as well
- We will see later what the two arrows mean that come from the encoder (these are the keys and values in QKV attention)

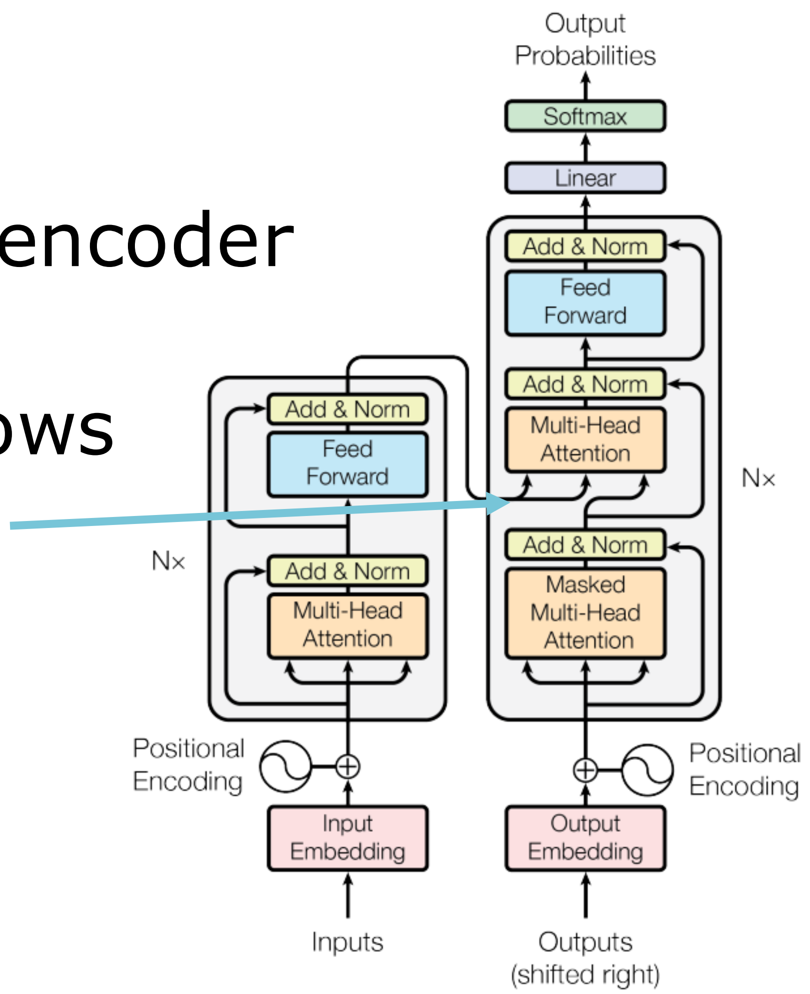


Figure 1: The Transformer - model architecture.

Feed-forward Network (FFN)

- Applies the following operations:
  - Linear layer
  - ReLU (GELU/SwiGLU in modern LLMs)
  - Linear layer

- First linear layer is an up-projection: Projects into higher dimension (512 → 2048 in the paper)
  - This is called the "inner" dimension of the FFN
- Second layer projects down to 512-dimensional vector again

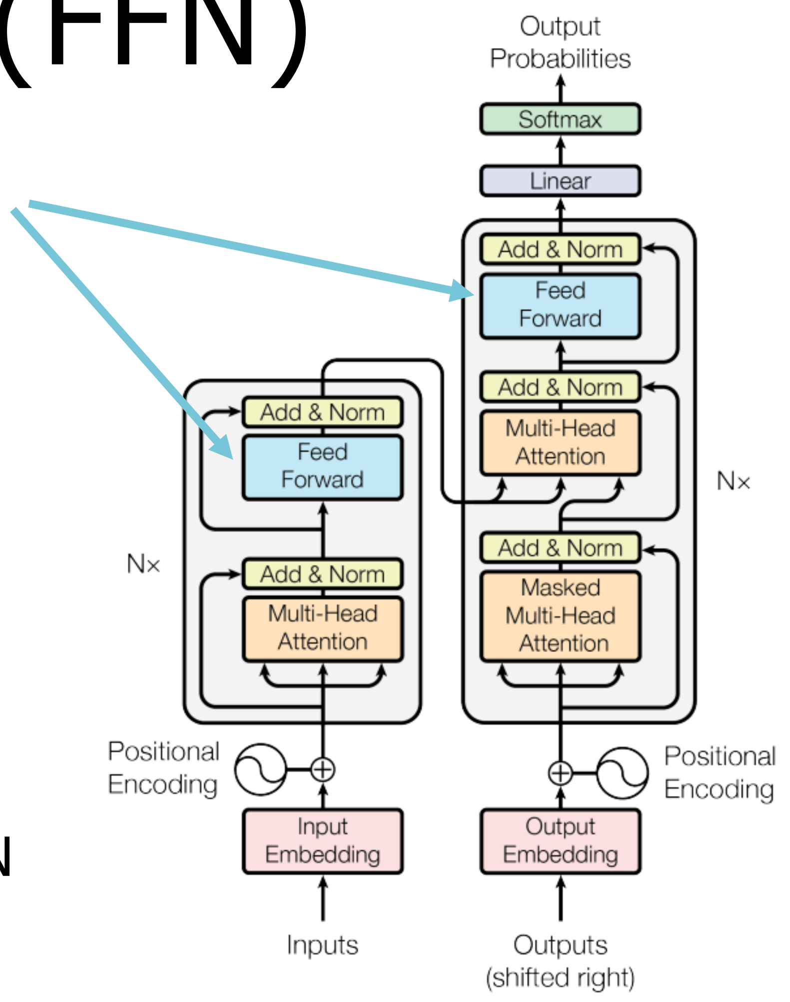


Figure 1: The Transformer - model architecture.

Role of FFN

- Still a topic of research what exactly they add
  - Example: Lena Voita's blog post and paper
- In general: adding a non-linearity (such as a ReLU) to a network increases the complexity of functions it can implement
- Up-projection for the inner dimension: Potentially *disentangles* some factors of variation such that they can be treated separately by the non-linearity

A Note on Layers

- What is a layer in this architecture?
- Sometimes a Transformer block (attentions + FFN) is called a Transformer layer
  - And self-attention, cross-attention and FFN are called sublayers
- But with the original definition of a layer, just the FFN consists of at least 2 layers…
- Make sure people know what you are talking about by giving context
- … and ask if you’re unsure!

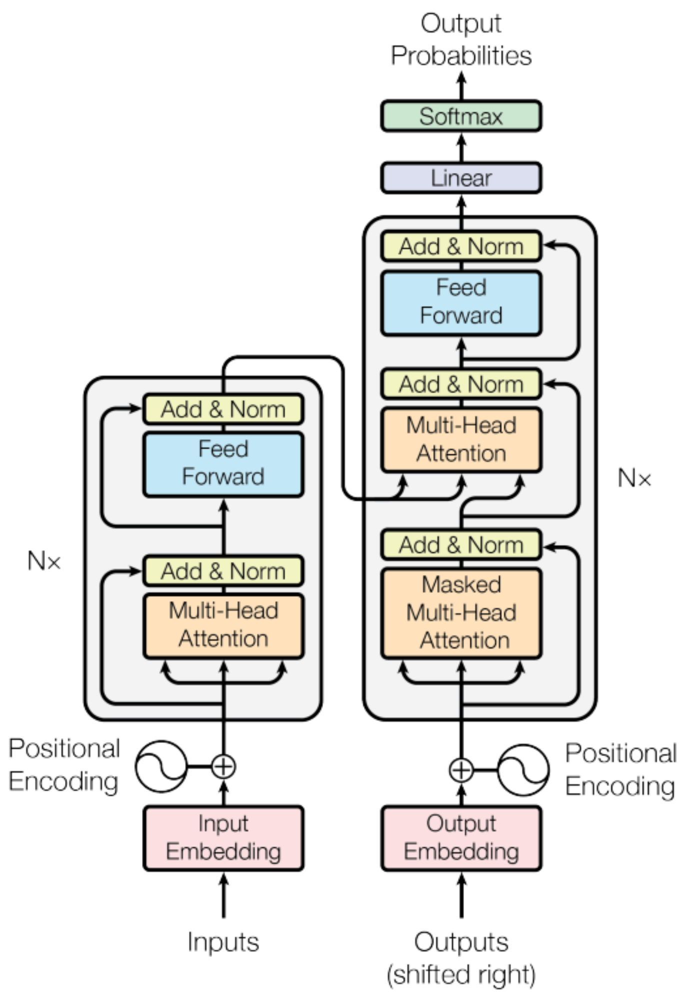
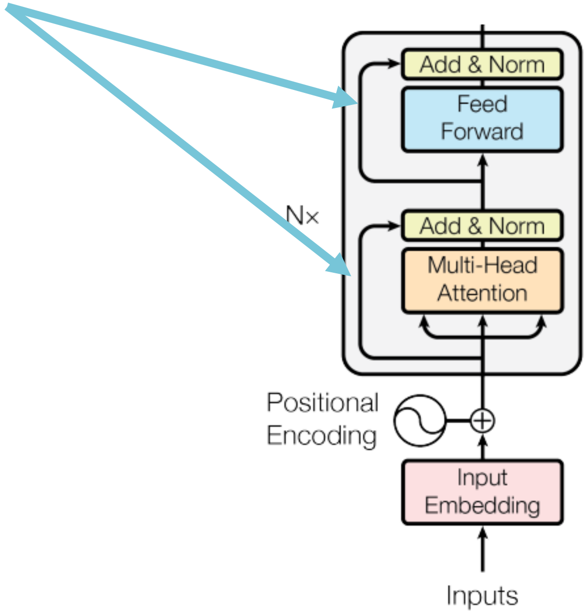


Figure 1: The Transformer - model architecture.

Intuition: Skip/Residual Connections

- Direct connections around a (sub)layer



- Shorter path length → better gradient flow
  - Mentioned as a method to avoid vanishing gradients

Advanced Intuition: (Topological) Spaces

- ML researchers/practitioners are often talking about *spaces*
  - Embedding space
  - Latent space
  - Query/key/value space
  - Feature space (e.g. vision vs. text in multimodal models)
- We can map from one space to the other with the help of a *projection*
  - If we map from higher dimensional (e.g. one-hot vector) to lower dimensional (300-dimensional word vectors), we also call the mapping an *embedding*, i.e. we *embed* an object (=word) into the 300-dimensional *latent/embedding space*

Advanced Intuition: (Topological) Spaces

- Imperfect analogy: RGB vs. hexadecimal “color spaces”
  - Different coordinate systems for describing colors
  - Can map from one system to the other
  - Can perform operations (e.g. addition, increasing hue) on the elements of the same system, but not on elements from two different systems
    - Same with NN spaces: adding vectors in different spaces is not meaningful
    - Project to same space, then perform the operation
    - Lucky for us: we let the model learn this projection

Query-Key-Value Attention

QKV Attention

- New way to look at attention
- Attention consists of 3 parts
  - Queries: formulate questions on new information that we need
  - Keys: match the queries if they have that type of information
  - Values: contain the information
- Attention function tries to find matching keys for queries
  → computes attention weights (= how well they match)
- Attention weights are multiplied with the values to give us the new information

QKV Attention

- How does original attention match this view?
  - Queries: Previous decoder hidden state
  - Keys: Encoder hidden states
  - Values: Encoder hidden states

Attention Definition

- Context vector c_i
  - a weighted sum of encoder hidden states
  - computed for each decoding time step

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_j^{(\text{enc})}$$

i: time step in decoder
j: time step in encoder

- Attention weights α_{ij}

$$\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}$$

- Attention function f_{att}

$$e_{ij} = f_{\text{att}}\left(h_{i-1}^{(\text{dec})}, h_j^{(\text{enc})}\right)$$

previous decoder hidden state

current encoder hidden state

Attention Definition

- Context vector c_i
  - a weighted sum of encoder hidden states
  - computed for each decoding time step

$$c_i = \sum_{j=1}^{n} \alpha_{ij} \underbrace{\boxed{h_j^{(\mathrm{enc})}}}_{\text{Values}}$$

i: time step in decoder
j: time step in encoder

- Attention weights α_{ij}

$$\alpha_{ij} = \mathrm{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}$$

- Attention function f_{att}

$$e_{ij} = f_{\mathrm{att}}\left(\underbrace{\boxed{h_{i-1}^{(\mathrm{dec})}}}_{\text{Queries}}, \underbrace{\boxed{h_j^{(\mathrm{enc})}}}_{\text{Keys}}\right)$$

previous
hidden state

hidden state

QKV Attention

- How does original attention match this view?
  - Queries: Previous decoder hidden state
  - Keys: Encoder hidden states
  - Values: Encoder hidden states

- New: separate keys and values (through projections)

What are the queries, keys and values in Transformer?

- Linear projections ($Wx + b$) of the input
- Different weights & biases for Q, K, V
- Self-attention
  - Inputs to the Transformer block
- Cross-attention
  - Queries: Output of decoder self-attention
  - Keys & values: Final layer encoder hidden states

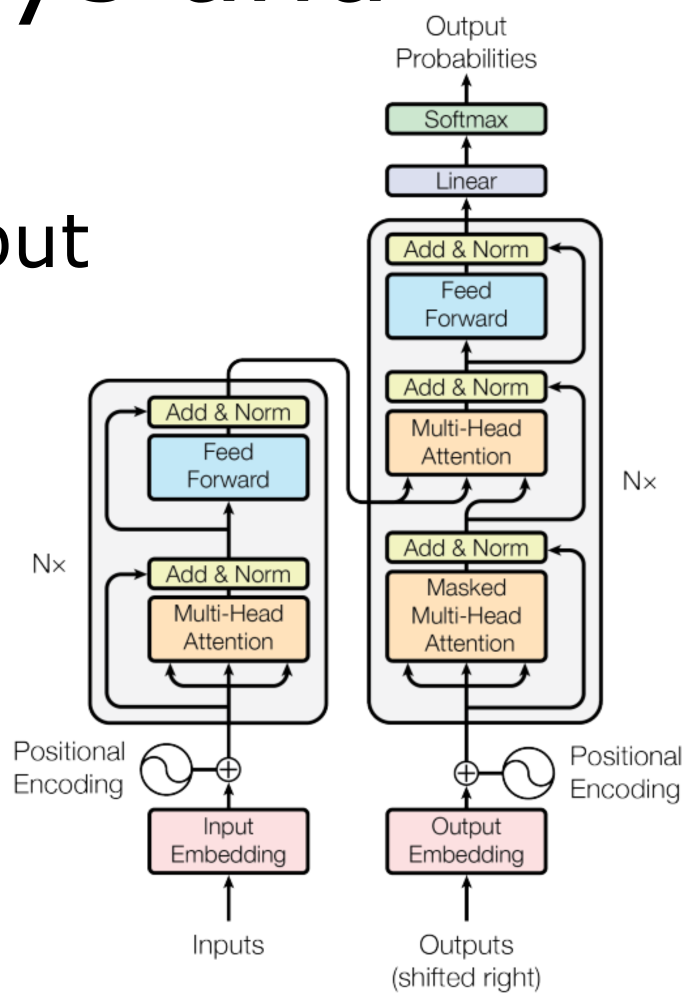


Figure 1: The Transformer - model architecture.

Dot-product Attention

- We saw multiplicative attention for RNNs:

$$f_{\text{att}}\left(h_{i-1}^{(\text{dec})}, h_j^{(\text{enc})}\right) = h_{i-1}^{(\text{dec})} W h_j^{(\text{enc})}$$

- Transformer uses scaled dot-product attention:

$$f_{\text{att}}(q_i, k_j) = \frac{q_i k_j^\top}{\sqrt{d}}$$

d is the dimension of q_i, k_j

q_i, k_j are the output of projections themselves

Dot-product Attention

- We saw multiplicative attention for RNNs:

$$f_{\text{att}}\left(h_{i-1}^{(\text{dec})}, h_j^{(\text{enc})}\right) = h_{i-1}^{(\text{dec})} W h_j^{(\text{enc})}$$

- If we write out Transformer scaled dot-product attention and use $h_{i-1}^{(\text{dec})}, h_j^{(\text{enc})}$ as inputs:

$$f_{\text{att}}(q_i, k_j) = \frac{q_i k_j^\top}{\sqrt{d}} = \left(W_q h_{i-1}^{(\text{dec})} + b_q\right)\left(W_k h_j^{(\text{enc})} + b_k\right)^\top d^{-\frac{1}{2}}$$

Dot-product Attention

- We saw multiplicative attention for RNNs:

$$f_{\text{att}}\left(h_{i-1}^{(\text{dec})}, h_j^{(\text{enc})}\right) = h_{i-1}^{(\text{dec})} W h_j^{(\text{enc})}$$

- If we write out Transformer scaled dot-product attention and use $h_{i-1}^{(\text{dec})}, h_j^{(\text{enc})}$ as inputs:

$$f_{\text{att}}(q_i, k_j) = \frac{q_i k_j^\top}{\sqrt{d}} = \left(W_q h_{i-1}^{(\text{dec})} + b_q\right)\left(W_k h_j^{(\text{enc})} + b_k\right)^\top d^{-\frac{1}{2}}$$

Added 1 projection matrix, 2 biases, 1 scalar

Dot-product Attention

- Transformer attention function:

$$f_{\mathrm{att}}(q_i, k_j) = \frac{q_i k_j^{\top}}{\sqrt{d}}$$

- Attention weights (as before):

$$\alpha_{ij} = \mathrm{softmax}\left(f_{\mathrm{att}}(q_i, k_j)\right)$$

- Attention output (previously our context vector):

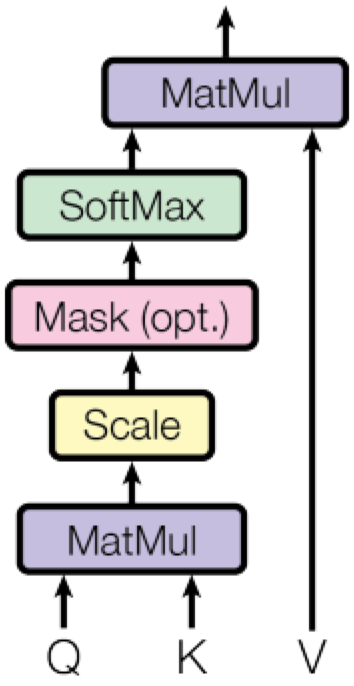
$$\mathrm{output} = \sum_{j=1}^{n} \alpha_{ij} v_j$$

Dot-product Attention

We can write all of this in matrix notation:

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

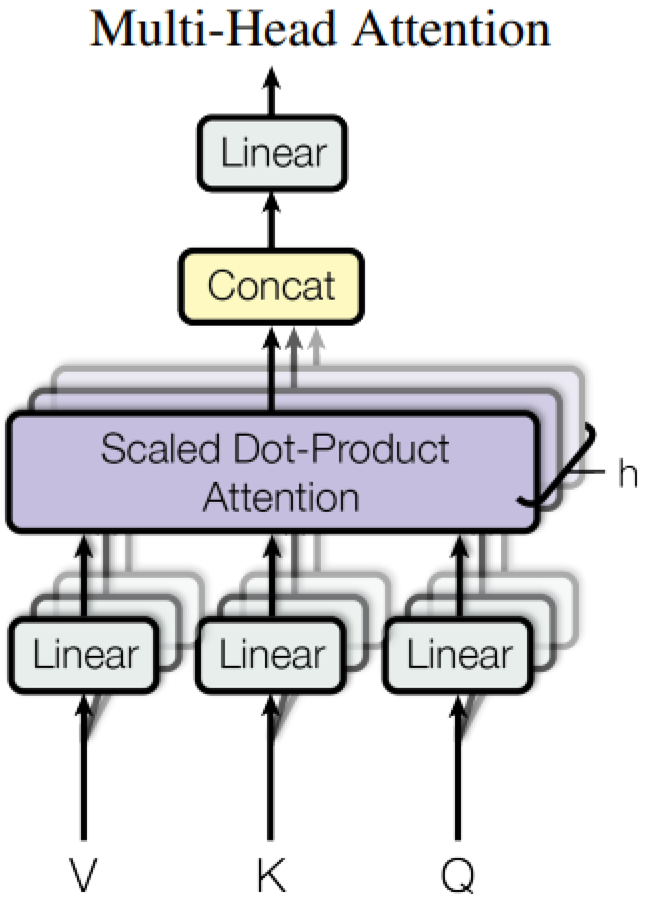
Scaled Dot-Product Attention



Multi-head Attention

- Transformer adds 1 more trick:
  multi-head attention
  1. Split the input to the attention function into h chunks of dimension d/h (h is the number of heads, don't confuse with RNN hidden states)
  2. Compute attention on each chunk separately
  3. Concatenate the result

$$\begin{pmatrix}1\\2\\3\\4\\5\\6\end{pmatrix} \longrightarrow \begin{array}{l}\text{Attention}\left(\begin{pmatrix}1\\2\end{pmatrix}\right)\\[1em] \text{Attention}\left(\begin{pmatrix}3\\4\end{pmatrix}\right)\\[1em] \text{Attention}\left(\begin{pmatrix}5\\6\end{pmatrix}\right)\end{array} \longrightarrow \begin{pmatrix}a\\b\\c\\d\\e\\f\end{pmatrix}$$

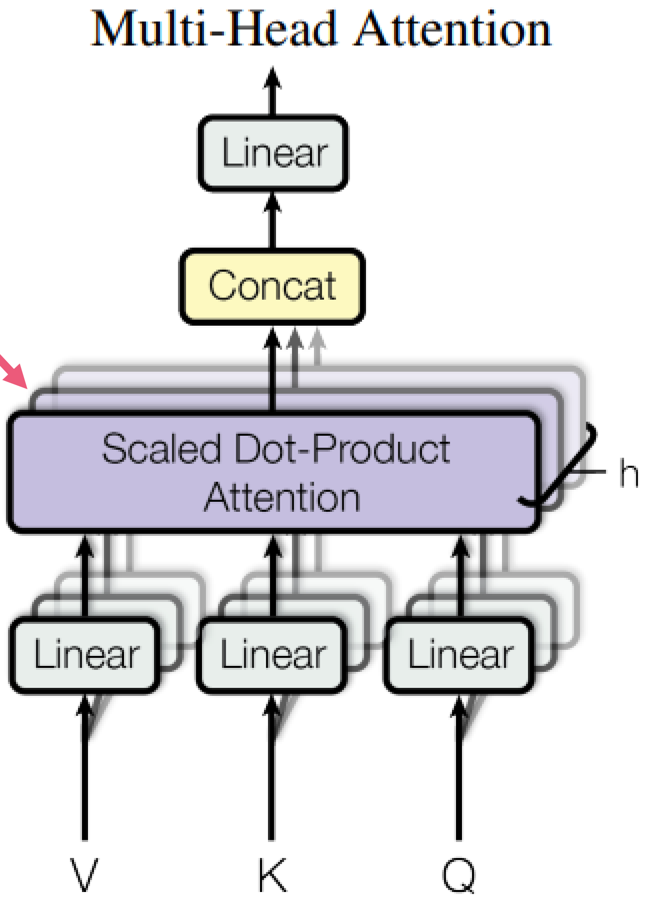


Multi-head Attention

- Transformer adds 1 more trick: *multi-head attention*
  1. Split the input to th… h chunks of dimensio… f heads, don't confuse with RNN hidden states)
  2. Compute attention on each chunk separately
  3. Concatenate the result

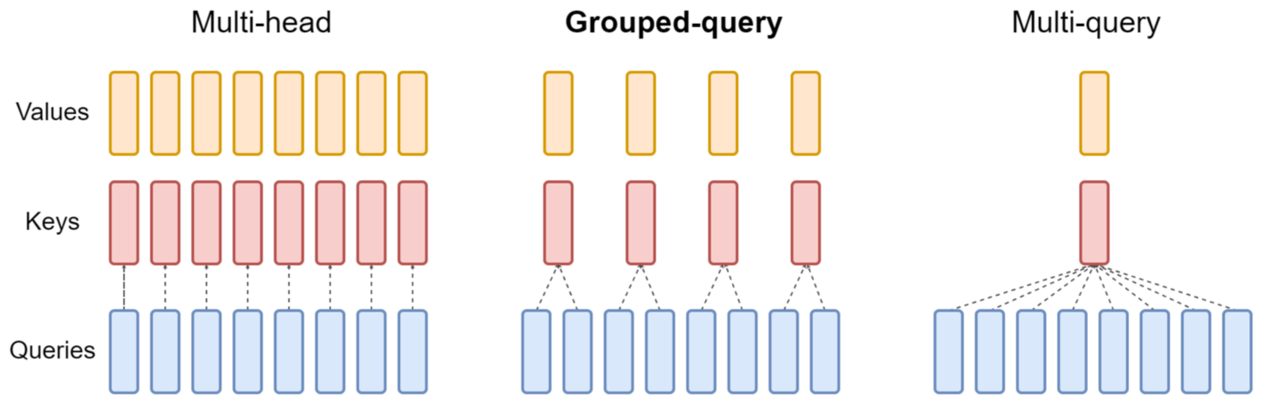
These are called *attention heads*

$$\begin{pmatrix}1\\2\\3\\4\\5\\6\end{pmatrix} \longrightarrow \begin{matrix}\text{Attention}\left(\begin{pmatrix}1\\2\end{pmatrix}\right)\\ \text{Attention}\left(\begin{pmatrix}3\\4\end{pmatrix}\right)\\ \text{Attention}\left(\begin{pmatrix}5\\6\end{pmatrix}\right)\end{matrix} \longrightarrow \begin{pmatrix}a\\b\\c\\d\\e\\f\end{pmatrix}$$



Attention Variants

- Multi-query attention
  - Shazeer, 2019
  - Instead of *h* heads for keys/values, use only 1 head
  - Keep *h* heads for queries
- Grouped query attention
  - Ainslie et al., 2023
  - Groups of keys/values (can be chosen to equal the number of GPUs)
  - Used in modern LLMs for efficiency



Transformer Attention Output

- We no longer use the context vector to predict the next hidden state
- Instead, the attention output is our next hidden state
- We refine this representation over multiple iterations of self-attention → (cross-attention in the decoder →) FFN
- Around each attention and FFN block, there is a residual connection
  - The input to the block gets added to the output, and layer normalization is applied

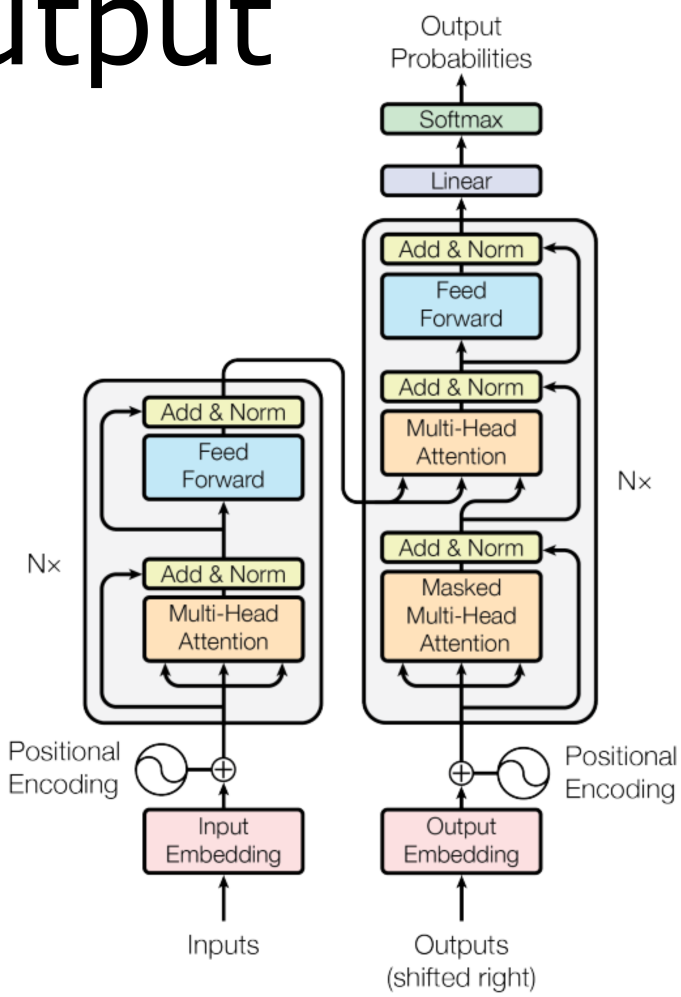


Figure 1: The Transformer - model architecture.

Layer Normalization

- Purpose: Keep activations (= hidden states) in a similar range across layers
  → No explosion/vanishing of activations
  → No explosion/vanishing of gradients
- Goal: Get outputs with 0 mean and unit variance

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

γ, β are learnable parameters

Layer Normalization

- Purpose: Keep activations (= hidden states) in a similar range across layers
  → No explosion/vanishing of activations
  → No explosion/vanishing of gradients
- Goal: Get outputs with 0 mean and unit variance

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

γ, β are learnable parameters

I'm not sure these are necessary…

 https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html

Complexity of Attention

- Every query q_i looks at every key k_j
- This mean m x n x d multiplications in the dot-product
  - m = n for self-attention

query length | key length | vector dim

- For long inputs, this can be quite costly…
  - Self-attention has quadratic complexity in the sequence length (imagine summarizing a book)
  - There are multiple approaches that try to make attention more efficient, e.g. Transformer-XL, Longformer, Performer, …

Exercise: Transformer

Position Encoding

Why do we need position encoding?

- RNNs
  - Process input sequentially
  - It is clear which word came before and after
- Transformers
  - Process the entire sequence in parallel
  - When we compare the word at position i to that at position j, how do we know their order and how far apart they are?

→ Position encoding

Types of Position Encoding

- Absolute position encoding
  - As a function of the position
  - As a learned embedding
- Relative position embeddings

Absolute Position Encoding

- For an input word at position i, create a d-dimensional vector
- The entries at dimension j of this position vector are:

$$\text{pos}(i, 2j) = \sin\left(i/10000^{2j/d}\right)$$

$$\text{pos}(i, 2j+1) = \cos\left(i/10000^{2j/d}\right)$$

Absolute Position Encoding

- For an input word at position i, create a d-dimensional vector
- The entries at dimension j of this position vector are:

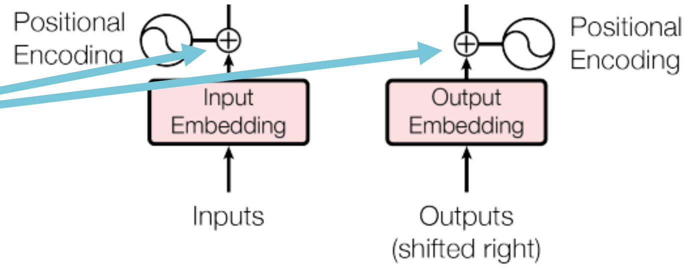
$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ \dots \end{bmatrix}$$

even $\mathrm{pos}(i, 2j) = \sin\left(i/10000^{2j/d}\right)$

odd $\mathrm{pos}(i, 2j+1) = \cos\left(i/10000^{2j/d}\right)$

Absolute Position Embedding

- Learn a separate embedding for each position i = 1, …, max_pos
  - max_pos: Maximum input length that was used during training, typically 512 (BERT) or 1024 (BART)
  - How do we deal with longer inputs?
    - Truncate input
    - Copy the position embedding at the last position for each following position
- Absolute position embeddings are added to the word embeddings at the input



Absolute Position Encoding vs. Embedding

- Performance the same
- Advantage for absolute position encoding: Works for any input length…
- … but absolute position embeddings are always used in practice
  - For standard Transformers, very long inputs are not the typical input
  - Transformers that are designed to deal with long inputs either train larger position embeddings, or have a different strategy to deal with position

Relative Position Embedding (only for self-attention)

- [Shaw et al., 2018]
- Learn embeddings for relative positions: $j - i$ = -max_dist, …, -1, 0, 1, …, max_dist
  - They use max_dist = 16 in the paper
- Distance between words at positions i and j: $j - i = -3$

- Can't add them to input word embeddings: A different embedding has to be used for each output position
  → Embeddings are added during attention computation instead

Relative Position Embedding

- Standard attention function:

$$f_{\text{att}}(q_i, k_j) = \frac{q_i k_j^\top}{\sqrt{d}}$$

- Attention function with relative position embeddings pos_{i-j}:

$$f_{\text{att}}(q_i, k_j) = \frac{q_i \left(k_j + \text{pos}_{i-j}\right)^\top}{\sqrt{d}}$$

- Shaw et al., 2018 has another relative position term for distance of queries to values, but that one doesn't help

HSLU

Exercise:
Position Encoding