Transformers

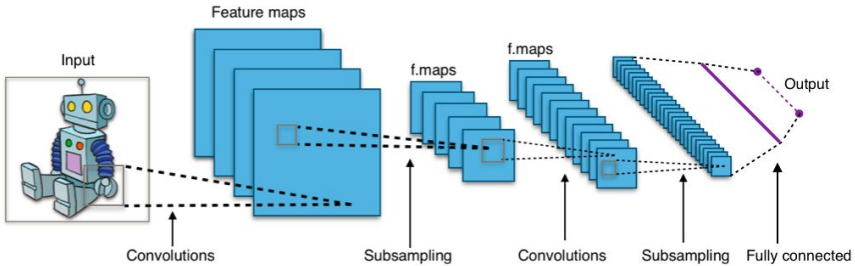
Deep Learning

Slides by Ivan Rubachev, Ya Research/HSE
Aziz Temirkhanov, Lambda

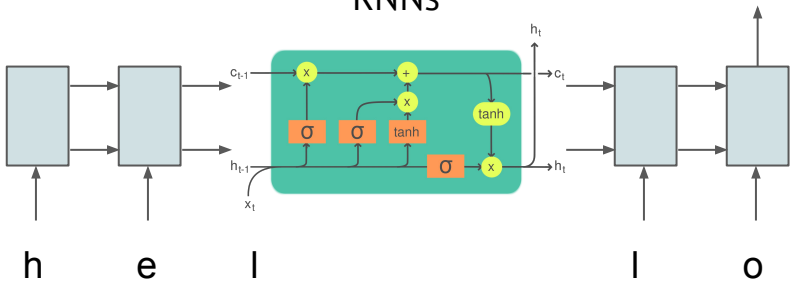
yandex-research/ml-residency

The classic landscape

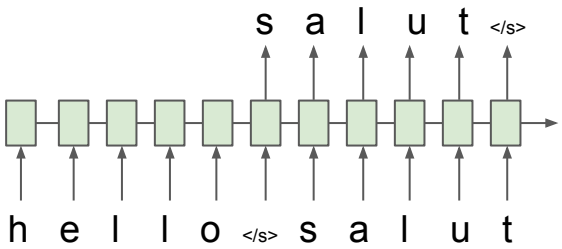
Computer vision
CNNs



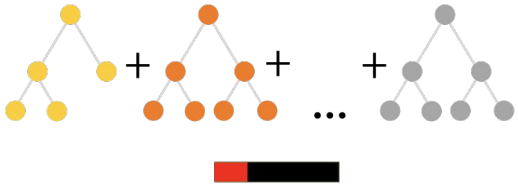
Natural Language Processing
RNNs



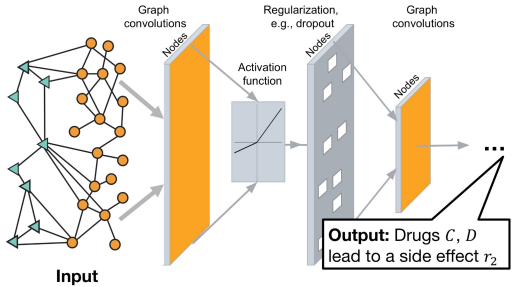
Translation
RNNs



Tabular
GBDTs



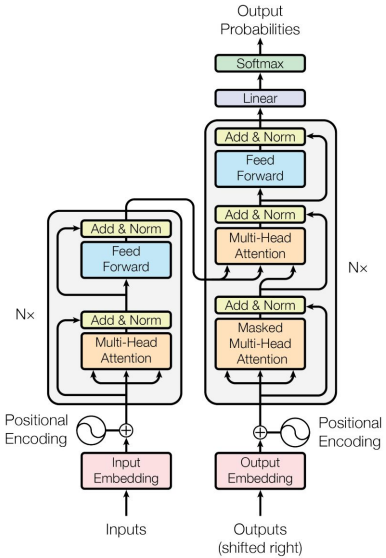
Graphs
GCNs



Transformer Takeover

- **All you need** for a good enough DL solution today is knowing how to encode your modality into tokens (and how to train **the** model, and where to get the data, …)

Computer vision



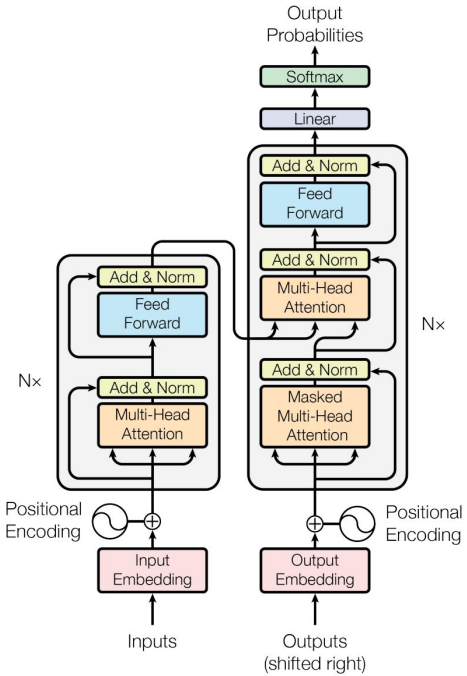
Natural Language Processing

Translation

Speech

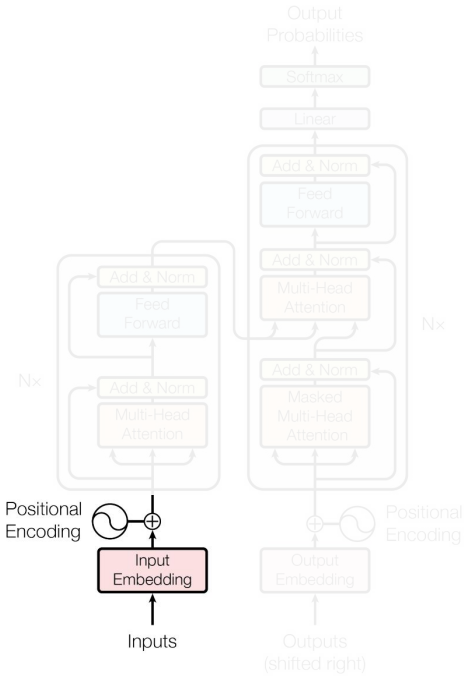
Tabular

Attention Is All You Need – The Transformer model



Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



Input (Tokenization and) Embedding

Input text is first split into pieces. Can be characters, word, "tokens":

```
"The detective investigated" -> [The_] [detective_] [invest] [igat] [ed_]
```

Tokens are indices into the "vocabulary":

```
[The_] [detective_] [invest] [igat] [ed_] -> [3 721 68 1337 42]
```

Each vocab entry corresponds to a learned d_{model} - dimensional vector.

```
[3 721 68 1337 42] -> [ [0.123, -5.234, ...], [...], [...], [...], [...] ]
```

Positional Encoding

Remember attention is permutation invariant, but language is not!

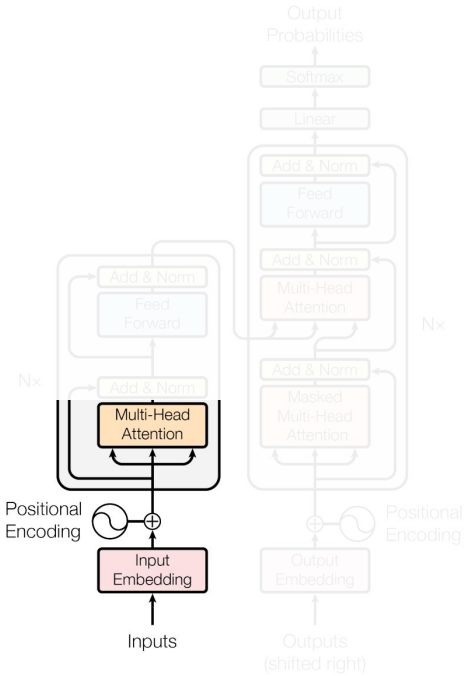
(“The mouse ate the cat” vs “The cat ate the mouse”)

Need to encode position of each word; just add something.

Think `[The_] + 10  [detective_] + 20  [invest] + 30 …` but smarter.

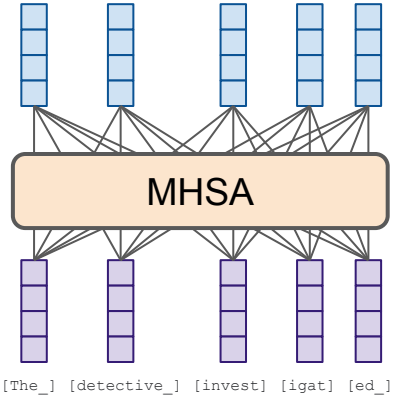
Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



Multi-headed Self-Attention

Meaning the input sequence is used to create queries, keys, and values!
Each token can "look around" the whole input, and decide how to update its representation based on what it sees.



Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)
Transformer image source: "Attention Is All You Need" paper

Neural Machine Translation by Jointly Learning to Align and Translate

2014, Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio

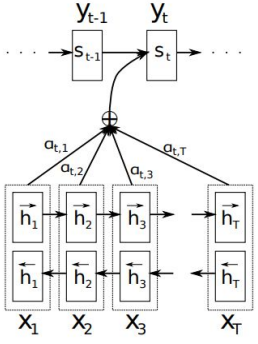
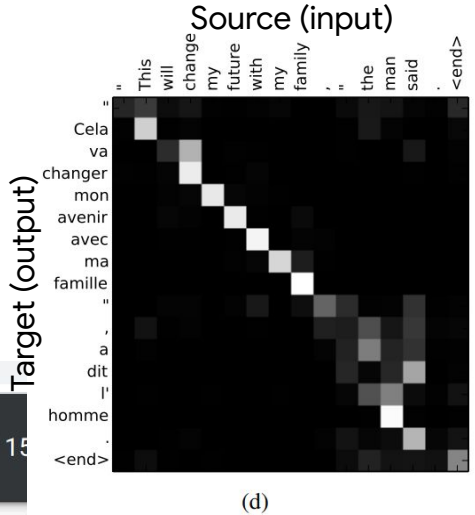
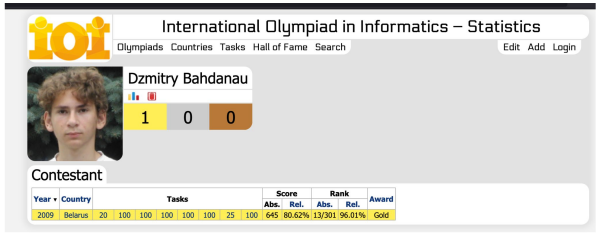


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.



(d)



4 / 15

attention 1/3

The probability α_{ij}, or its associated energy e_{ij}, reflects the importance of the annotation h_j with respect to the previous hidden state s_{i-1} in deciding the next state s_i and generating y_i. Intuitively, this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed-length vector. With this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly.

Attention is a function similar to a "soft" kv dictionary lookup:

1. Attention weights $a_{1:N}$ are query-key similarities:

   $\hat{a}_i = q \cdot k_i$

   Normalized via softmax: $a_i = e^{\hat{a}i} / \sum_j e^{\hat{a}j}$

2. Output z is attention-weighted average of values $v_{1:N}$:

   $z = \sum_i \hat{a}_i v_i = \hat{a} \cdot v$

3. Usually, k and v are derived from the same input x:

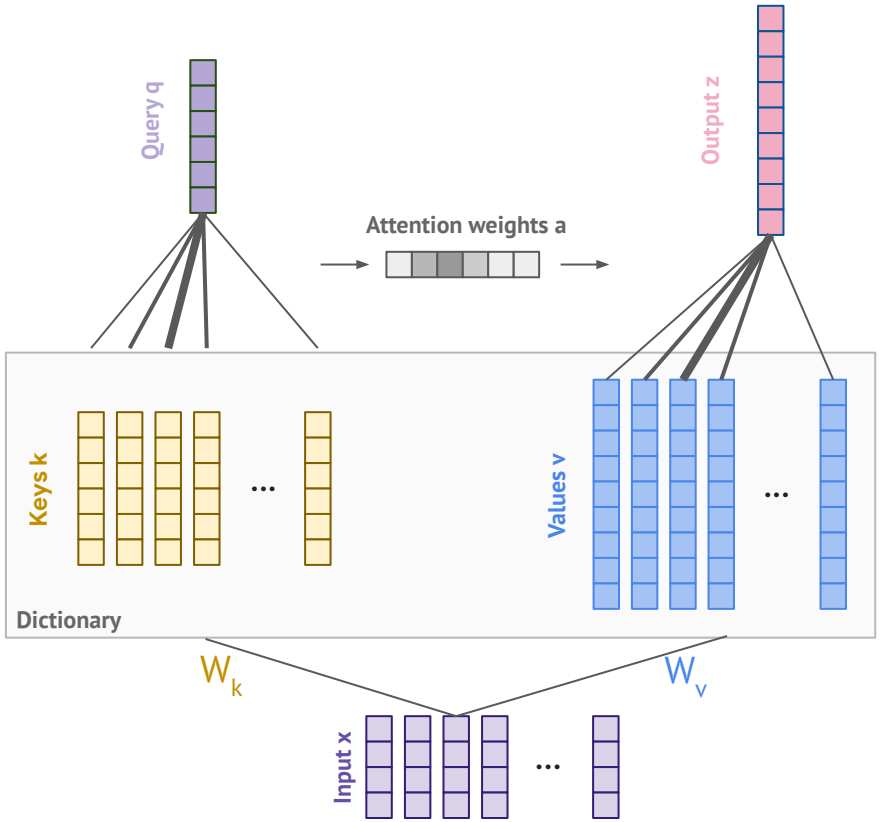
   $k = W_k \cdot x \qquad v = W_v \cdot x$

   The query q can come from a separate input y:

   $q = W_q \cdot y$

   Or from the same input x! Then we call it "self attention":

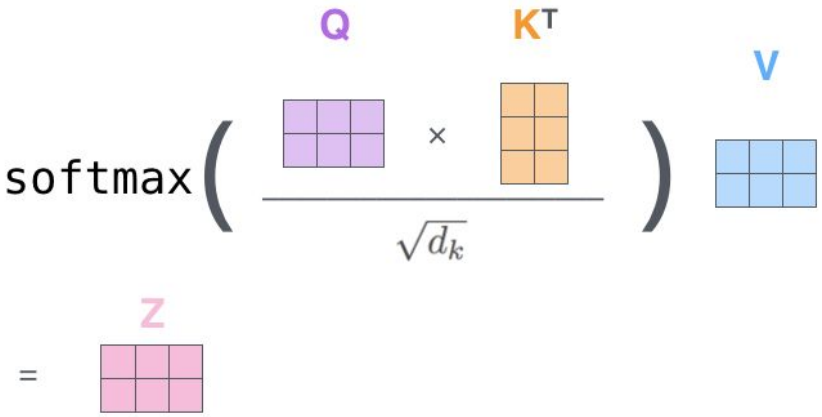
   $q = W_q \cdot x$



Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer for the slides https://www.youtube.com/watch?v=Eixl6t5oif0)

Transformer image source: "Attention Is All You Need" paper

In matmul's and with normalization:

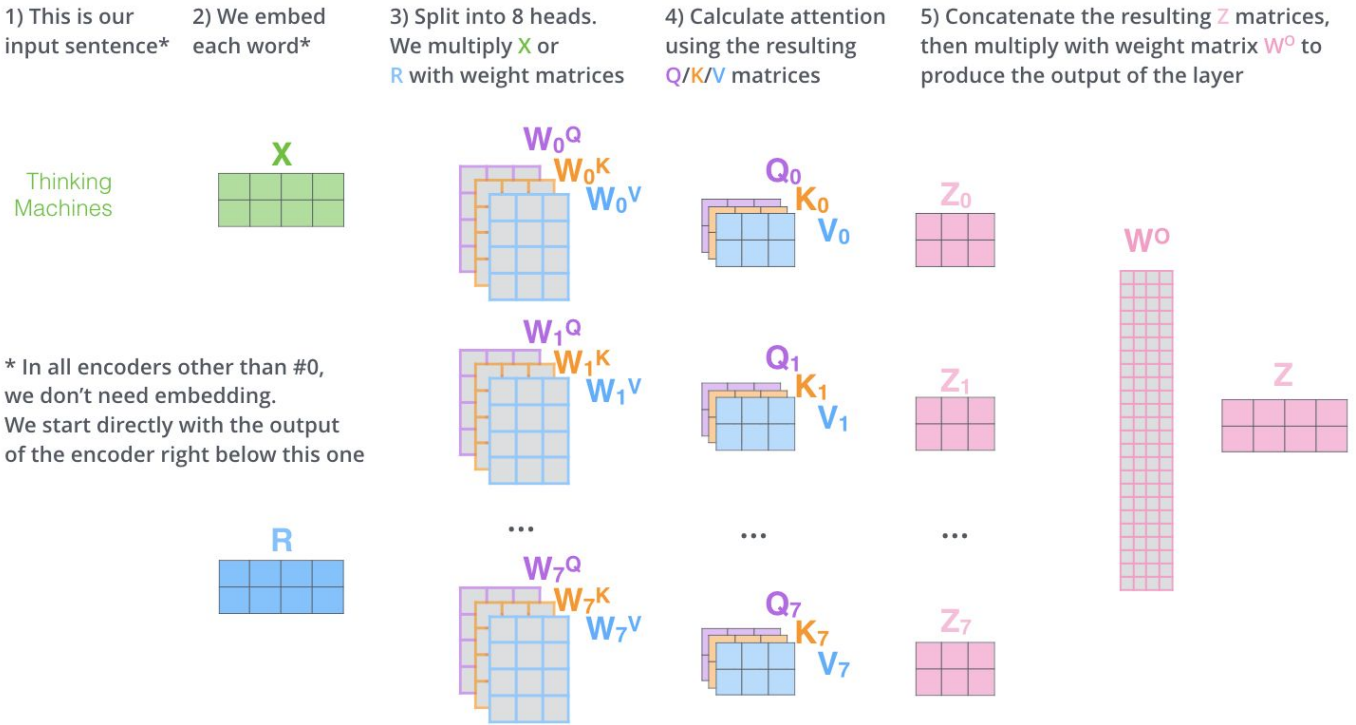


http://jalammar.github.io/illustrated-transformer

Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper

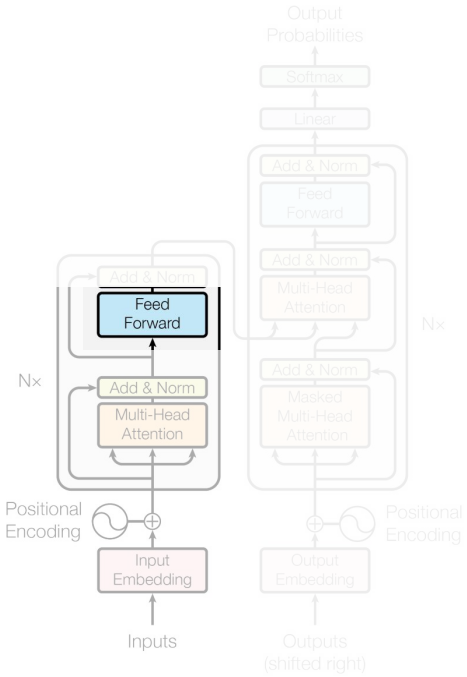
MultiHead



http://jalammar.github.io/illustrated-transformer

Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



Point-wise MLP

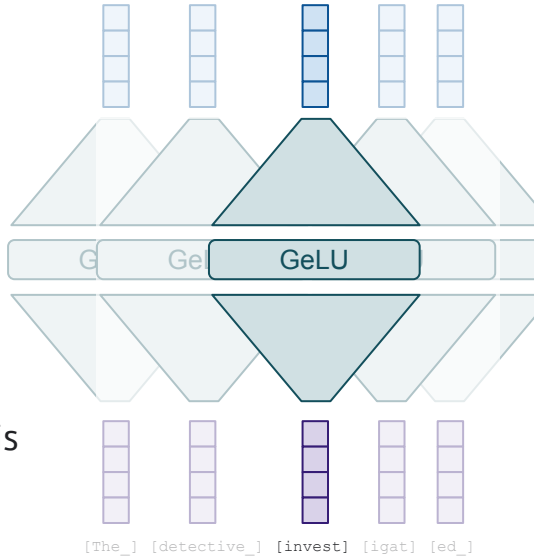
A simple MLP applied to each token individually:

$$z_i = W_2\,\mathrm{GeLU}(W_1 x + b_1) + b_2$$

Think of it as each token pondering for itself about what it has observed previously.
There's some weak evidence this is where "world knowledge" is stored, too.

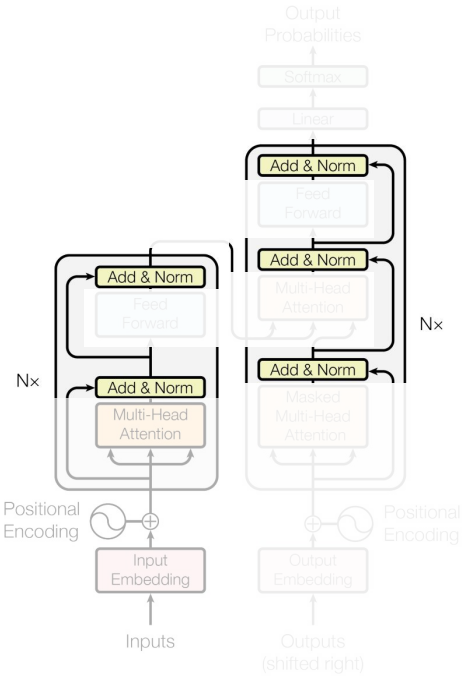
It contains the bulk of the parameters. When people make giant models and sparse/moe, this is what becomes giant.

Some people like to call it 1x1 convolution.



Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



Residual connections

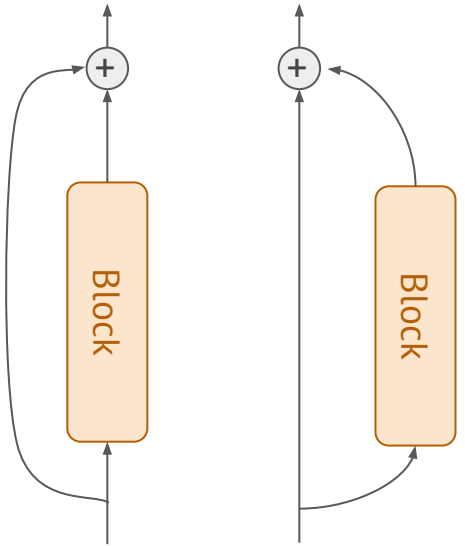
"Skip connection" == "Residual block"

Each module's output has the exact same shape as its input.

Following ResNets, the module computes a "residual" instead of a new value:

$$z_i = \text{Module}(x_i) + x_i$$

This was shown to dramatically improve trainability.



LayerNorm

Normalization also dramatically improves trainability.

There's **post-norm** (original)

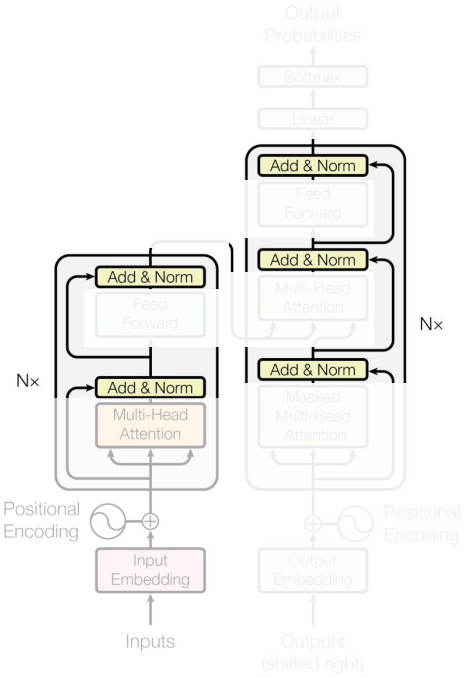
$$z_i = \text{LN}(\text{Module}(x_i) + x_i)$$

and **pre-norm** (modern)

$$z_i = \text{Module}(\text{LN}(x_i)) + x_i$$

Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



Residual connections

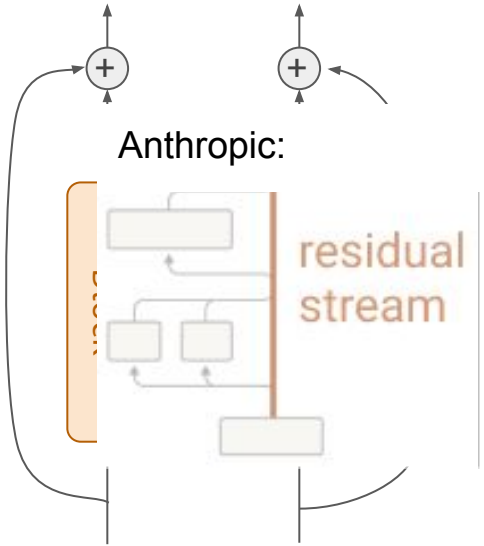
"Skip connection" == "Residual block"

Each module's output has the exact same shape as its input.

Following ResNets, the module computes a "residual" instead of a new value:

$$z_i = \text{Module}(x_i) + x_i$$

This was shown to dramatically improve trainability.



LayerNorm

Normalization also dramatically improves trainability.

There's **post-norm** (original)

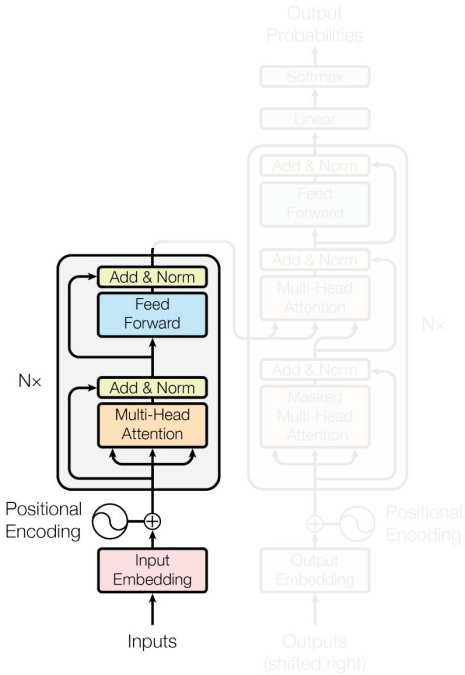
$$z_i = \text{LN}(\text{Module}(x_i) + x_i)$$

and **pre-norm** (modern)

$$z_i = \text{Module}(\text{LN}(x_i)) + x_i$$

Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



Encoding / Encoder

Since input and output shapes are identical, we can stack N such blocks.

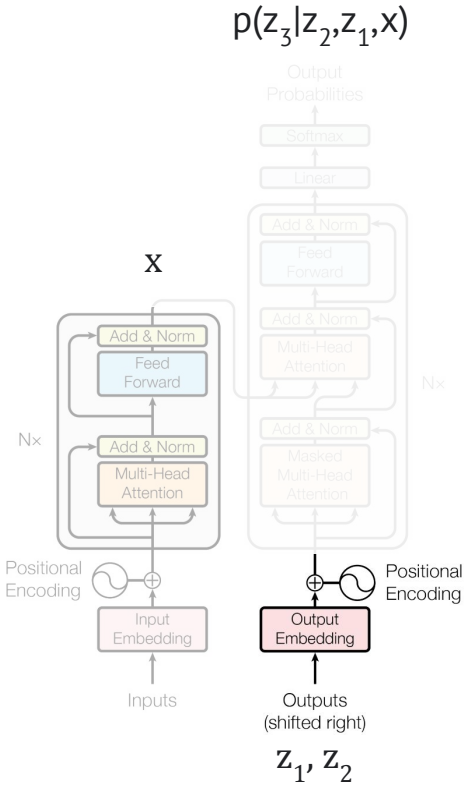
Typically, N=6 ("base"), N=12 ("large") or more.

Encoder output is a "heavily processed" (think: "high level, contextualized") version of the input tokens, i.e. a sequence.

This has nothing to do with the requested output yet (think: translation). That comes with the decoder.

Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



Decoding / the Decoder (alternatively Generating / the Generator)

What we want to model: $p(z|x)$

for example, in translation: $p(z \mid \text{"the detective investigated"})$ $\forall z$

we can *exactly* decompose into tokens:

$$p(z|x) = p(z_1|x)\ p(z_2|z_1,x)\ p(z_3|z_2,z_1,x)\ldots$$

Meaning, we can generate the answer one token at a time.
Each p is a full pass through the model.

For generating $p(z_3|z_2,z_1,x)$:

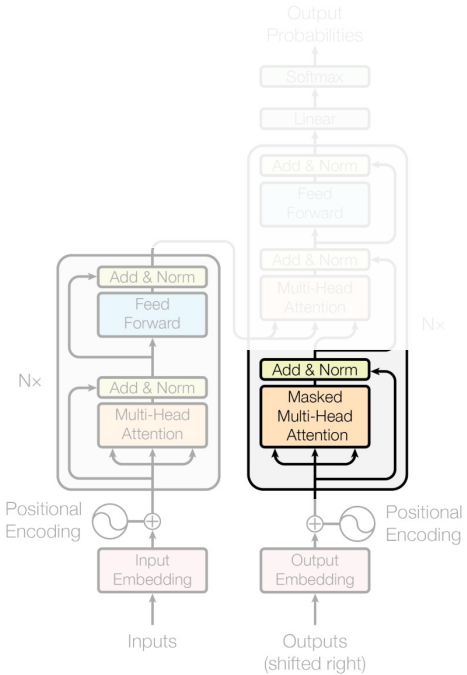
x comes from the encoder,

z_1, z_2 is what we have predicted so far, goes into the decoder.

Once we have p(z|x) we still need to actually sample a sentence such as "le détective a enquêté". Many strategies: greedy, beam-search, ...

Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



At training time: Masked self-attention

This is regular self-attention as in the encoder, to process what's been decoded so far, eg z_2,z_1 in $p(z_3|z_2,z_1,x)$, but with a trick...

If we had to train on one single $p(z_3|z_2,z_1,x)$ at a time: SLOW!

Instead, train on all $p(z_i|z_{1:i},x)$ simultaneously.

How? In the attention weights for z_i, set all entries i:N to 0.

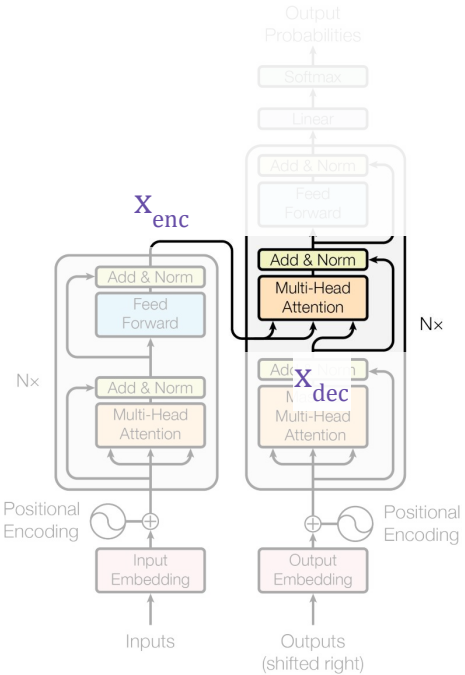
This way, each token only sees the already generated ones.

At generation time

There is no such trick. We need to generate one z_i at a time. This is why autoregressive decoding is extremely slow.

Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

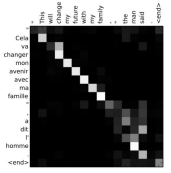
Transformer image source: "Attention Is All You Need" paper



"Cross" attention

Each decoded token can "look at" the encoder's output:

$$\text{Attn}(q=W_q x_{dec}, k=W_k x_{enc}, v=W_v x_{enc})$$



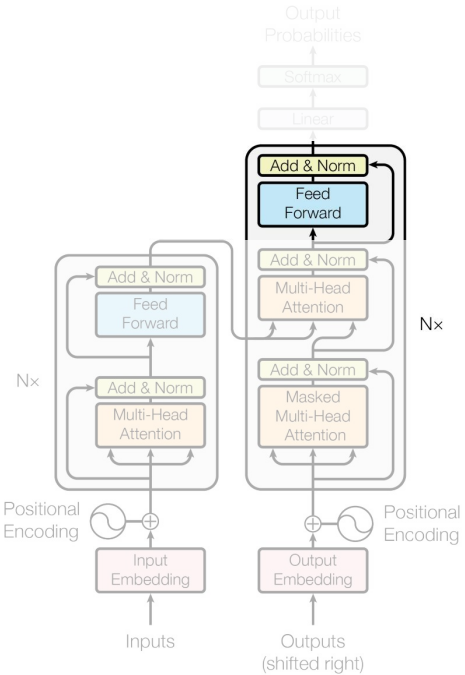
This is the same as in the 2014 paper.
This is where |x in $p(z_3|z_2,z_1,x)$ comes from.

Because self-attention is so widely used, people have started just calling it "attention".
Hence, we now often need to explicitly call this "cross attention".

Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

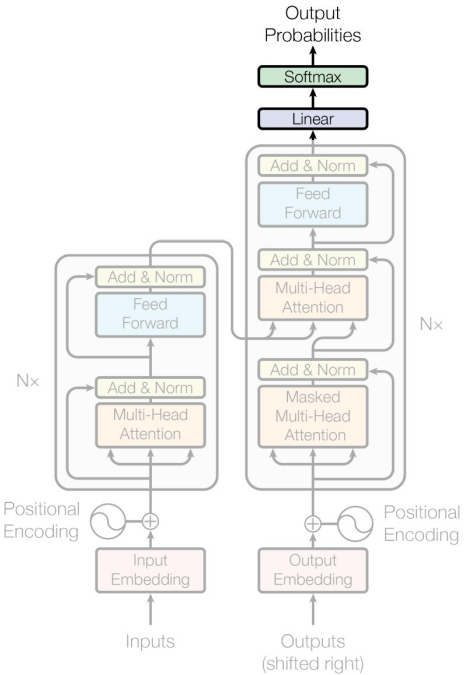
Transformer image source: "Attention Is All You Need" paper

Feedforward and stack layers.



Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper



Output layer

Assume we have already generated K tokens, generate the next one.

The decoder was used to gather all information necessary to predict a probability distribution for the next token (K), over the whole vocab.

Simple:

- linear projection of token K
- SoftMax normalization

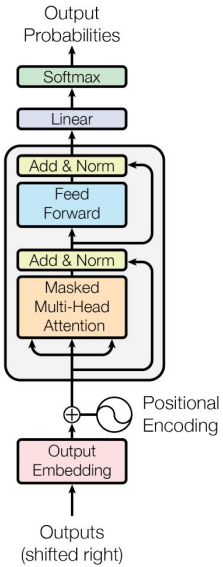
Thanks to Basil Mustafa for slide inspiration (and Lucas Beyer https://www.youtube.com/watch?v=EixI6t5oif0)

Transformer image source: "Attention Is All You Need" paper

Types of encoder-decoder combinations

Decoder (GPT)

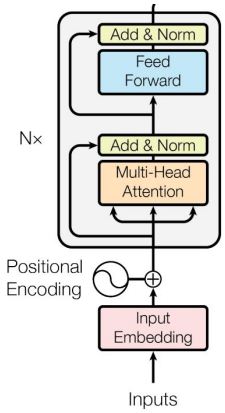
[sat_]



[START] [The_] [cat_]

Encoder (BERT, VIT)

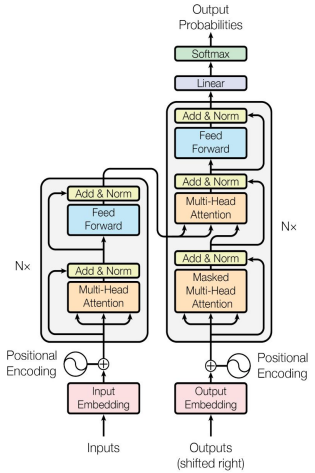
[*] [*] **[sat_]** [*] **[the_]** [*]



[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

Encoder-Decoder (T5)

Das ist gut.
A storm in Attala caused 6 victims.
This is not toxic.



Translate EN-DE: This is good.
Summarize: state authorities dispatched…
Is this toxic: You look beautiful today!

BERT

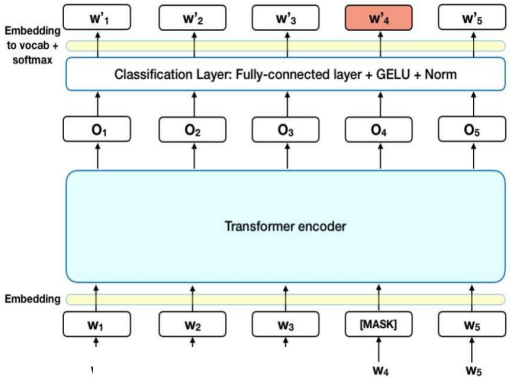
BERT = Bidirectional Encoder Representations from Transformers

Big Transformer Encoder

L – depth, H – embedding dim, A – number of attention heads

$\text{BERT}_{\text{BASE}}$: L=12, H=768, A=12, Total Parameters=110M

$\text{BERT}_{\text{LARGE}}$: L=24, H=1024, A=16, Total Parameters=340M



[Devlin et al., 2018]

RoBERTa, ELECTRA - как BERT, но лучше/быстрее

BERT was undertrained

RoBERTa: tune the hyperparameters and train longer

- Big batch
- byte-level (not character-level, unicode!) BPE
- dynamic mask generation

ELECTRA is a new pretraining approach which trains two transformer models: the generator and the discriminator.

- The generator's role is to replace tokens in a sequence, and is therefore trained as a masked language model.
- The discriminator, which is the model we're interested in, tries to identify which tokens were replaced by the generator in the sequence.

Smaller models

ALBERT [Lan et al., 2019]: Smaller embedding dim, shared across all layers

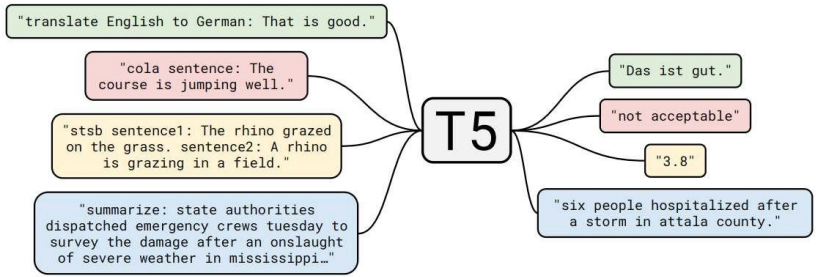
DistilBERT [Sanh et al., 2019]: distilled BERT into a smaller model

Chinchilla [Hoffmann et al., 2022]: TLDR: we need to train bigger models longer

T5

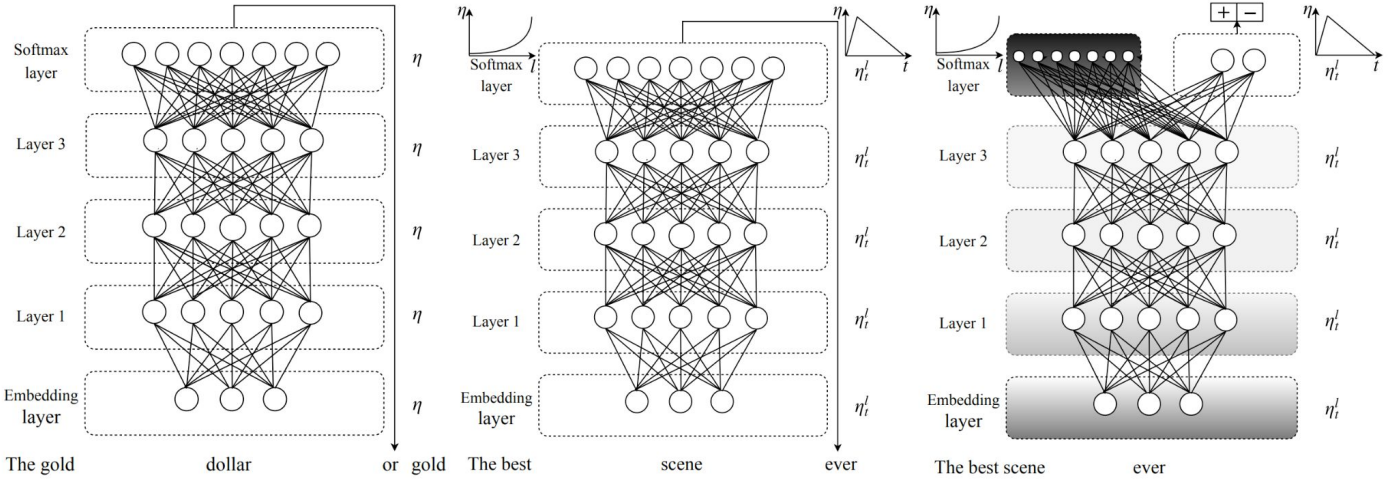
T5: Text-to-Text Transfer Transformer [Raffel et al.; 2019]

- Ablated many BERT-like unsupervised loss functions
- Formulate each supervised task as text-to-text
- More data



Generative Pretraining

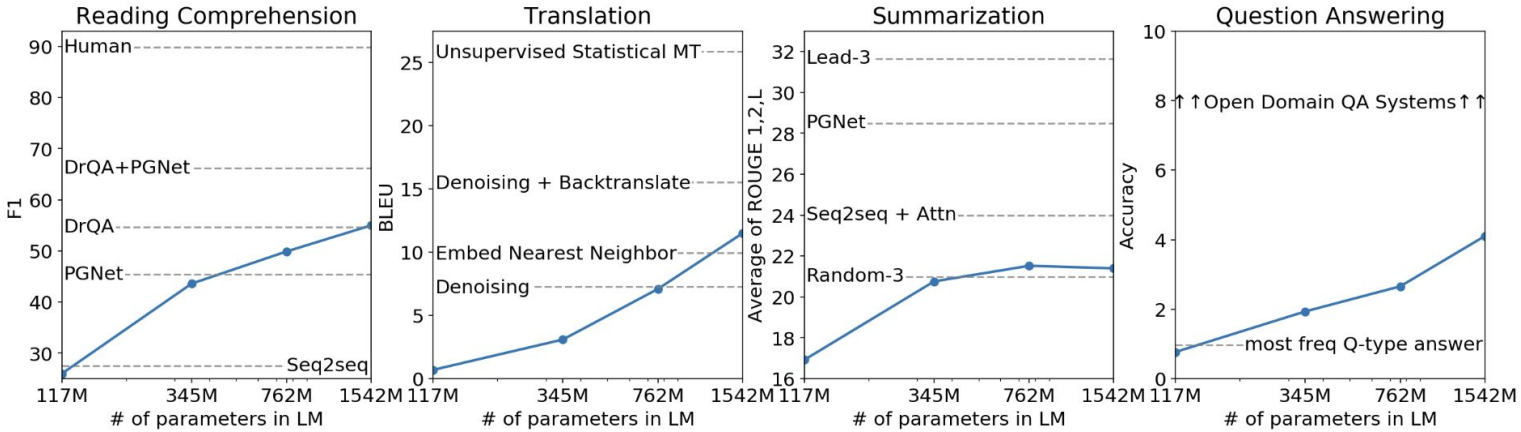
ULMFiT Introduced by Howard et al. in Universal Language Model Fine-tuning for Text Classification

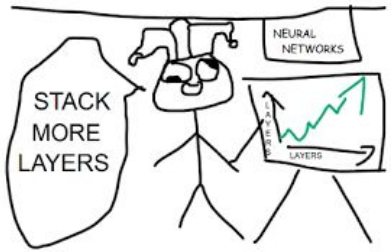


(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

Generative Pretraining

GPT-2, 3, 3.5...





Generative Pretraining

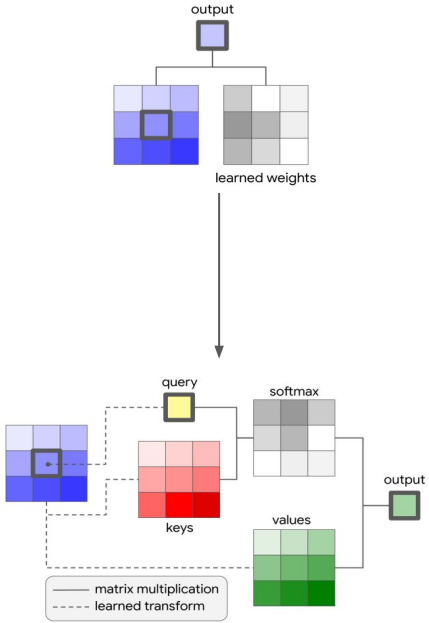
GPT-2, 3, 3.5...

More of this in the next lecture:

- prompting, finetuning
- agents
- models
- multimodal models

Vision

1. On pixels, but locally or factorized



stage	output	ResNet-50	**LR-Net-50 (7×7, m=8)**
res1	112×112	7×7 conv, 64, stride 2	**1×1, 64** **7×7 LR, 64, stride 2**
res2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		[1×1, 64 3×3 conv, 64 1×1, 256] ×3	[1×1, 100 **7×7 LR, 100** 1×1, 256] ×3
res3	28×28	[1×1, 128 3×3 conv, 128 1×1, 512] ×4	[1×1, 200 **7×7 LR, 200** 1×1, 512] ×4
res4	14×14	[1×1, 256 3×3 conv, 256 1×1, 1024] ×6	[1×1, 400 **7×7 LR, 400** 1×1, 1024] ×6
res5	7×7	[1×1, 512 3×3 conv, 512 1×1, 2048] ×3	[1×1, 800 **7×7 LR, 800** 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params		**25.5**×10^6	**23.3**×10^6
FLOPs		**4.3**×10^9	**4.3**×10^9

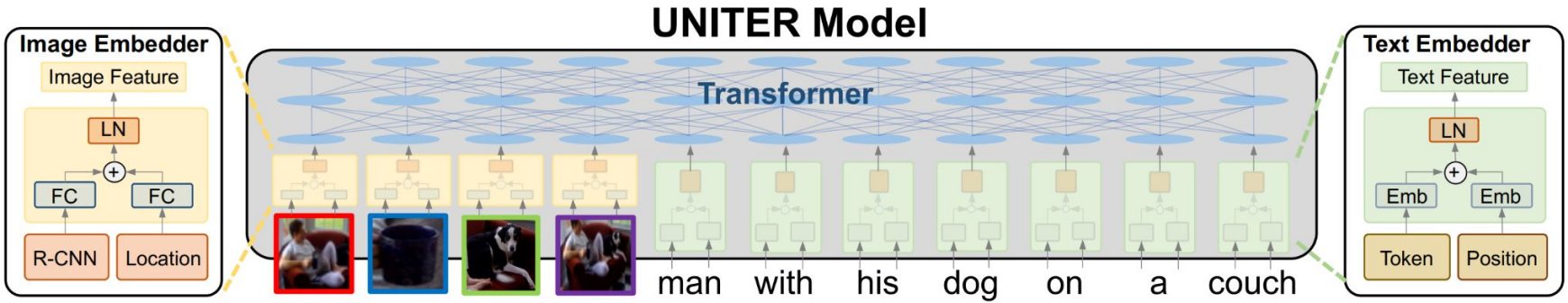
Many prior works attempted to introduce self-attention at the pixel level.

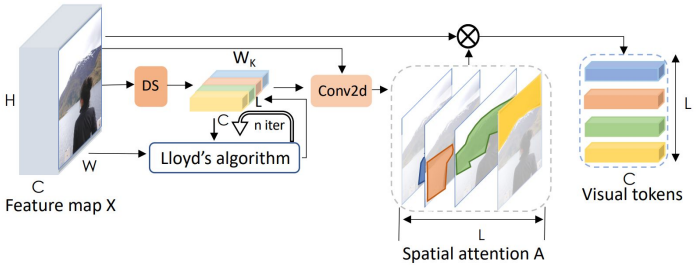
For 224px², that's 50k sequence length, too much!

Results:
Are usually "meh", nothing to call home about
Do not justify increased complexity
Do not justify slowdown over convolutions

Vision

2. Globally, after/inside a full-blown CNN, or even detector/segmenter!





Cons:

result is highly complex, often multi-stage trained architecture.

not from pixels, i.e. transformer can't "learn to fix" the (often frozen!) CNN's mistakes.

Examples:

DETR (Carion, Massa et.al. 2020)

UNITER (Chen, Li, Yu et.al. 2019)

VisualBERT (Li et.al. 20190)

Visual Transformers (Wu et.al. 2020)

ViLBERT (Lu et.al. 2019)

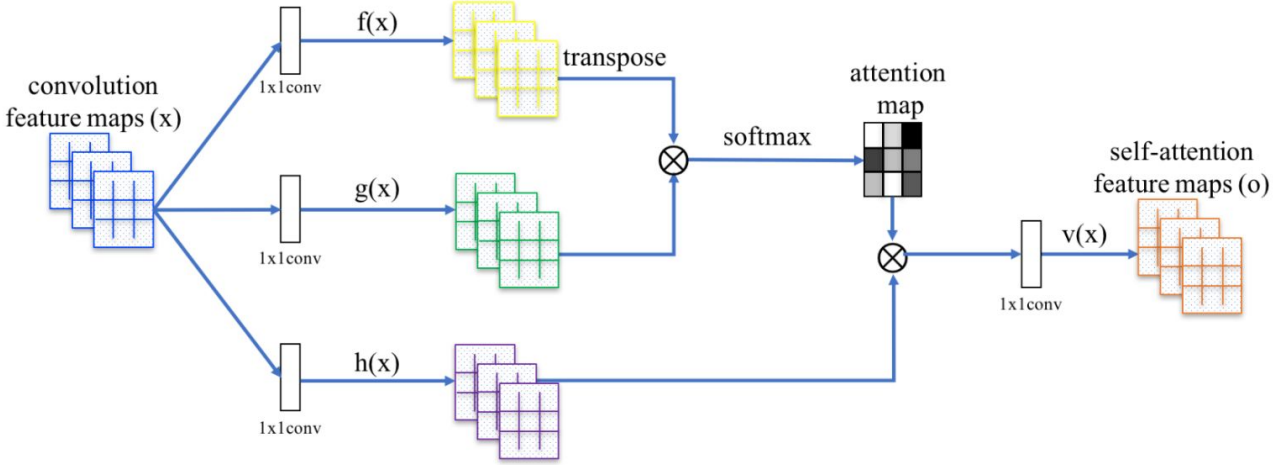
etc...

Vision

2.5. Self-Attention GAN

Model	Inception Score	Intra FID	FID
AC-GAN (Odena et al., 2017)	28.5	260.0	/
SNGAN-projection (Miyato & Koyama, 2018)	36.8	92.4	27.62*
SAGAN	**52.52**	**83.7**	**18.65**

Table 2. Comparison of the proposed SAGAN with state-of-the-art GAN models (Odena et al., 2017; Miyato & Koyama, 2018) for class conditional image generation on ImageNet. FID of SNGAN-projection is calculated from officially released weights.



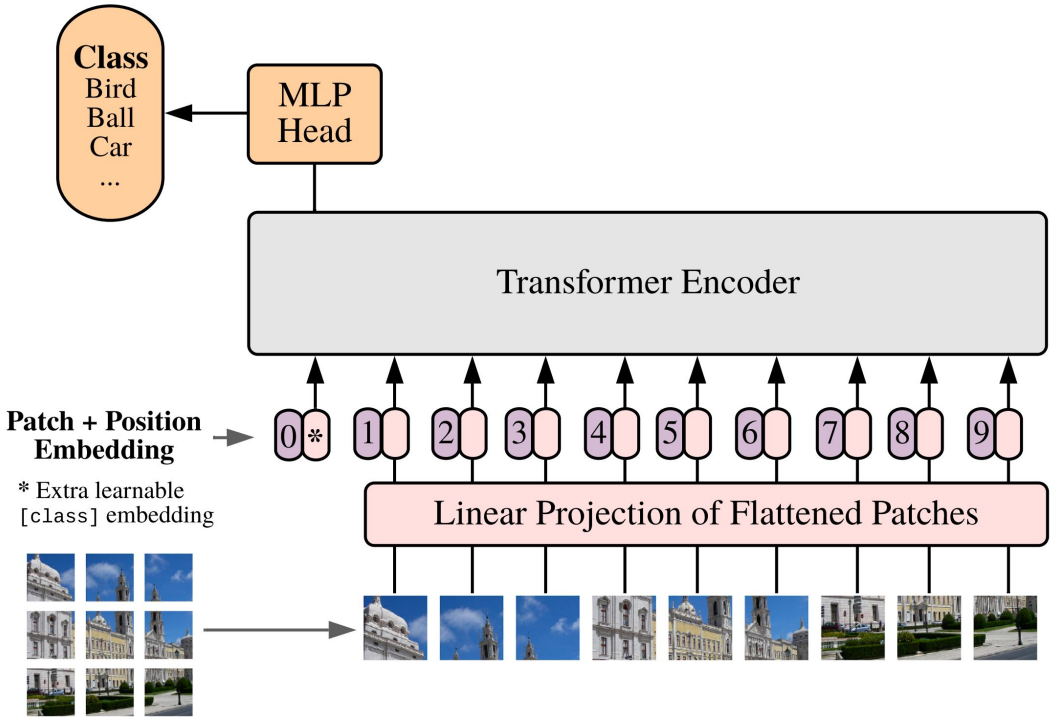
VIT

Many prior works attempted to introduce self-attention at the pixel level.

For 224px², that's 50k sequence length, too much!

Thus, most works restrict attention to local pixel neighborhoods, or as high-level mechanism on top of detections.

The **key breakthrough** in using the full Transformer architecture, standalone, was to **"tokenize" the image** by **cutting it into patches** of 16px², and treating each patch as a token, e.g. embedding it into input space.



Tabular

Encode continuous features as vectors by scaling an embedding
Use a regular Transformer afterwards

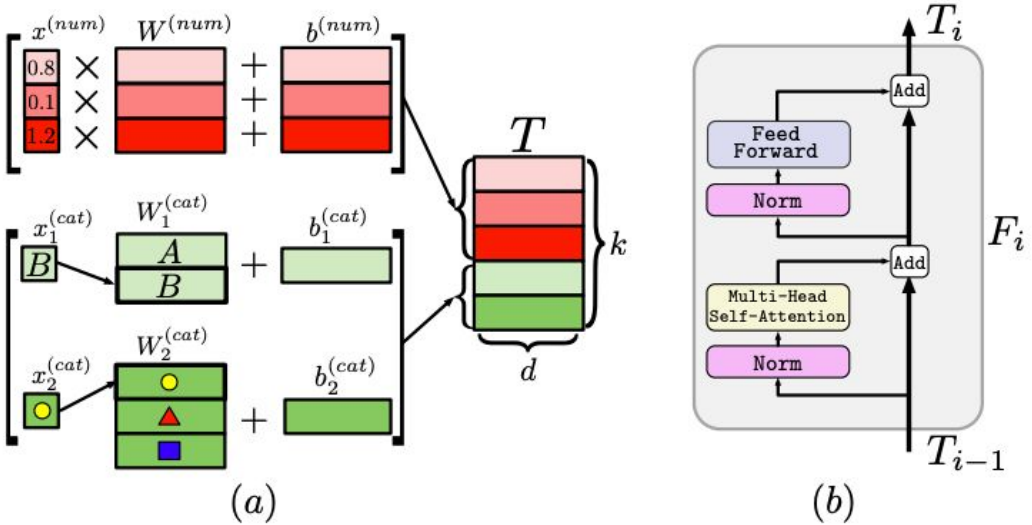


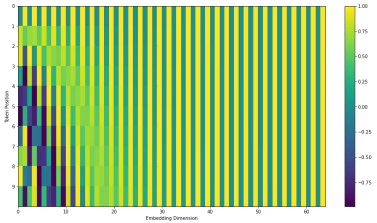
Figure 2: (a) Feature Tokenizer; in the example, there are three numerical and two categorical features; (b) One Transformer layer.

⚠️ Beware ⚠️

We are now in a research territory.

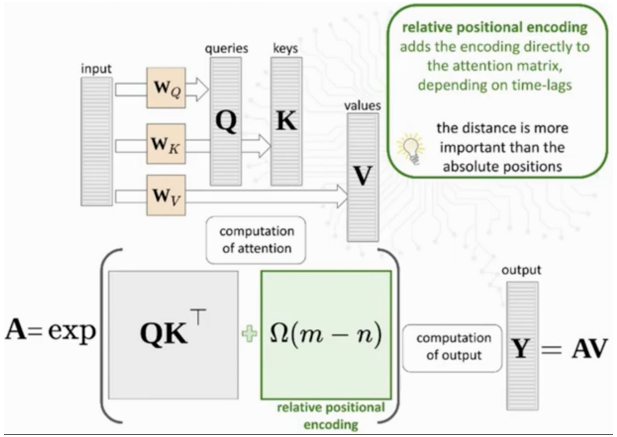
Encoding positional information (and extrapolation to longer sequences)

Sinusoidal embedding



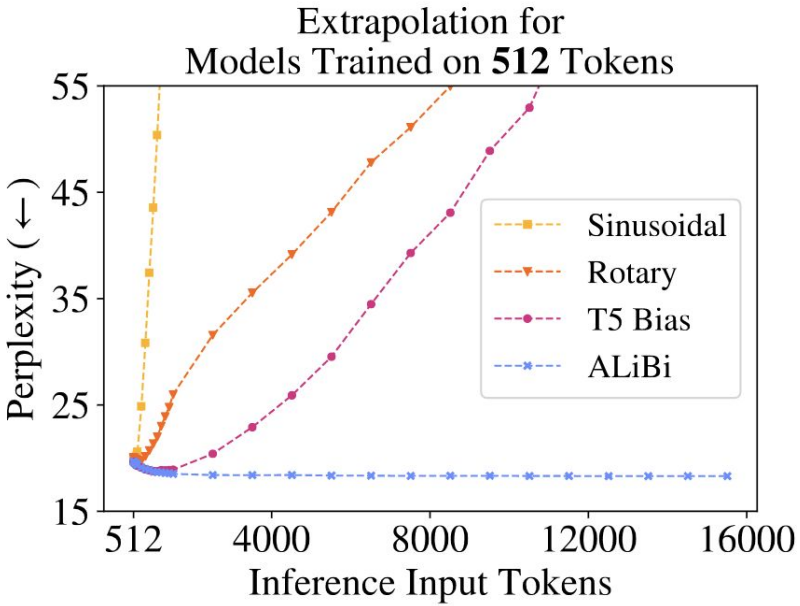
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

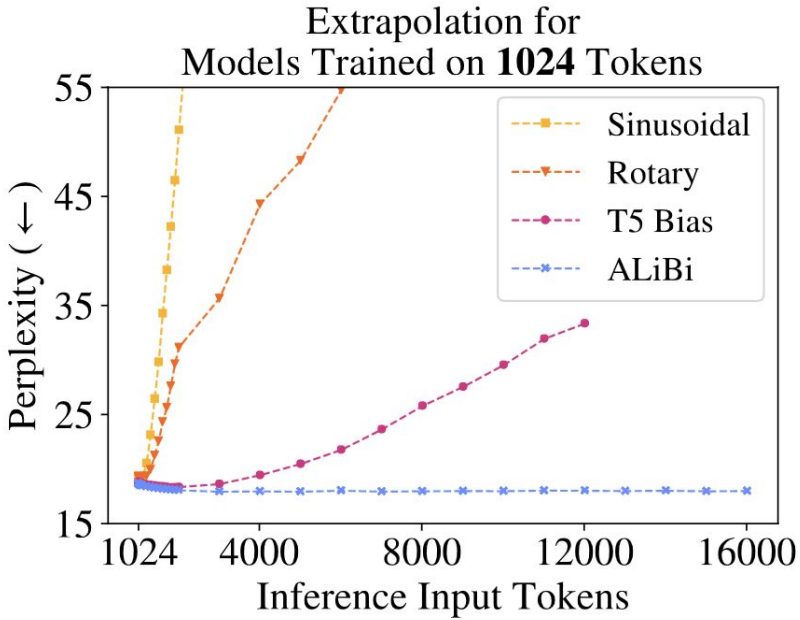
Relative embedding



$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$
$$z_i = \sum_{j=1}^{n} \alpha_{ij} (x_j W^V + a_{ij}^V)$$

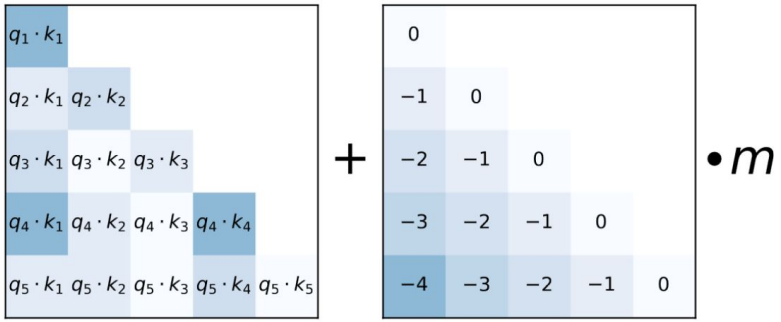
RoPE, ALiBi

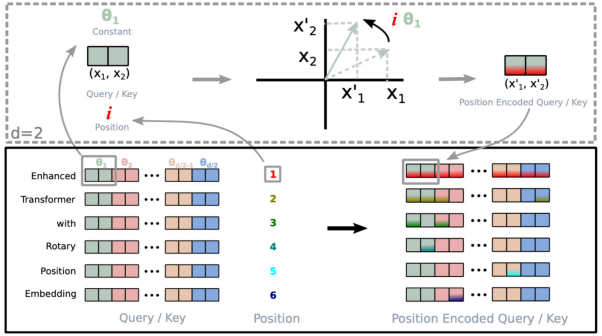

Extrapolation for
Models Trained on **512** Tokens
Perplexity (←)
55
45
35
25
15
512
4000
8000
12000
16000
Inference Input Tokens
Sinusoidal
Rotary
T5 Bias
ALiBi


Extrapolation for
Models Trained on **1024** Tokens
Perplexity (←)
55
45
35
25
15
1024
4000
8000
12000
16000
Inference Input Tokens
Sinusoidal
Rotary
T5 Bias
ALiBi

RoPE, ALiBi

ALiBi:





RoPE

$$\mathbf{f}(\mathbf{x}, m) = [(x_0 + \mathrm{i}x_1)e^{\mathrm{i}m\theta_0}, (x_2 + \mathrm{i}x_3)e^{\mathrm{i}m\theta_1}, \ldots, (x_{d-2} + \mathrm{i}x_{d-1})e^{\mathrm{i}m\theta_{d/2-1}}]^\top \quad (1)$$

where $\mathrm{i} := \sqrt{-1}$ is the imaginary unit and $\theta_j = 10000^{-2j/d}$. Using RoPE, the self-attention score

$$\begin{aligned} a(m,n) &= \mathrm{Re}\langle \mathbf{f}(\mathbf{q}, m), \mathbf{f}(\mathbf{k}, n)\rangle \\ &= \mathrm{Re}\left[\sum_{j=0}^{d/2-1} (q_{2j} + \mathrm{i}q_{2j+1})(k_{2j} - \mathrm{i}k_{2j+1})e^{\mathrm{i}(m-n)\theta_j}\right] \\ &= \sum_{j=0}^{d/2-1} (q_{2j}k_{2j} + q_{2j+1}k_{2j+1})\cos((m-n)\theta_j) + (q_{2j}k_{2j+1} - q_{2j+1}k_{2j})\sin((m-n)\theta_j) \\ &=: a(m-n) \end{aligned} \quad (2)$$

is only dependent on relative position $m - n$ through trigonometric functions. Here $\mathbf{q}$ and $\mathbf{k}$ are the query and key vector for a specific attention head. At each layer, RoPE is applied on both query and key embeddings for computing attention scores.

https://lilianweng.github.io/posts/2023-01-27-the-transformer-family-v2/#rotary-position-embedding

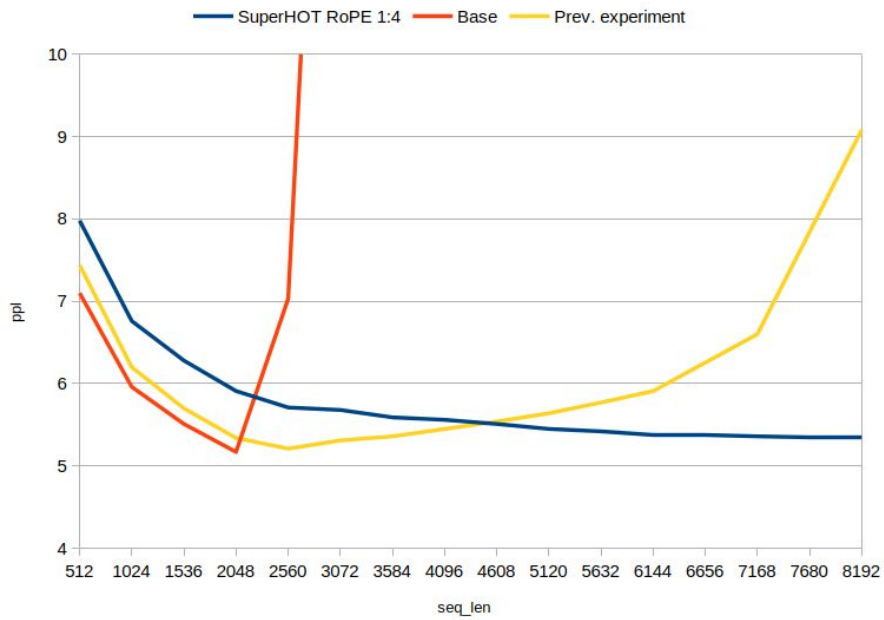
RoPE, Alibi

SuperHot RoPE

https://kaiokendev.github.io/til#extending-context-to-8k

RoPE, Alibi

SuperHot RoPE (June 2023)

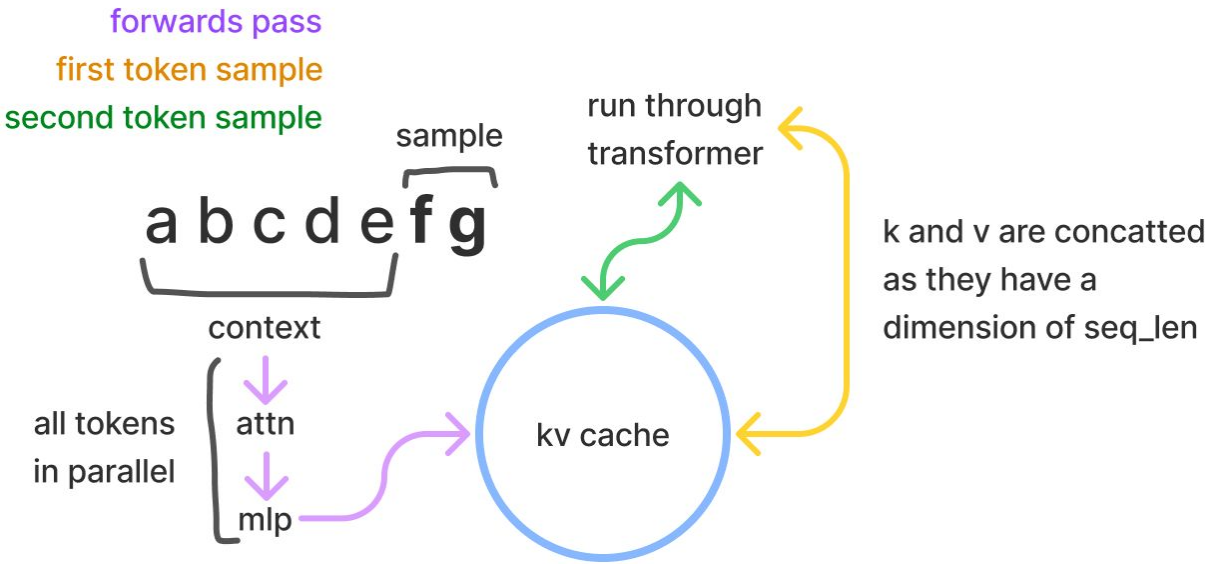


DINO VIT (April 2021)

```python
def interpolate_pos_encoding(self, x, w, h):
    npatch = x.shape[1] - 1
    N = self.pos_embed.shape[1] - 1
    if npatch == N and w == h:
        return self.pos_embed
    class_pos_embed = self.pos_embed[:, 0]
    patch_pos_embed = self.pos_embed[:, 1:]
    dim = x.shape[-1]
    w0 = w // self.patch_embed.patch_size
    h0 = h // self.patch_embed.patch_size
    # we add a small number to avoid floating point error in the interpolation
    # see discussion at https://github.com/facebookresearch/dino/issues/8
    w0, h0 = w0 + 0.1, h0 + 0.1
    patch_pos_embed = nn.functional.interpolate(
        patch_pos_embed.reshape(1, int(math.sqrt(N)), int(math.sqrt(N)), dim).permute(0, 3, 1, 2),
        scale_factor=(w0 / math.sqrt(N), h0 / math.sqrt(N)),
        mode='bicubic',
    )
    assert int(w0) == patch_pos_embed.shape[-2] and int(h0) == patch_pos_embed.shape[-1]
    patch_pos_embed = patch_pos_embed.permute(0, 2, 3, 1).view(1, -1, dim)
    return torch.cat((class_pos_embed.unsqueeze(0), patch_pos_embed), dim=1)
```

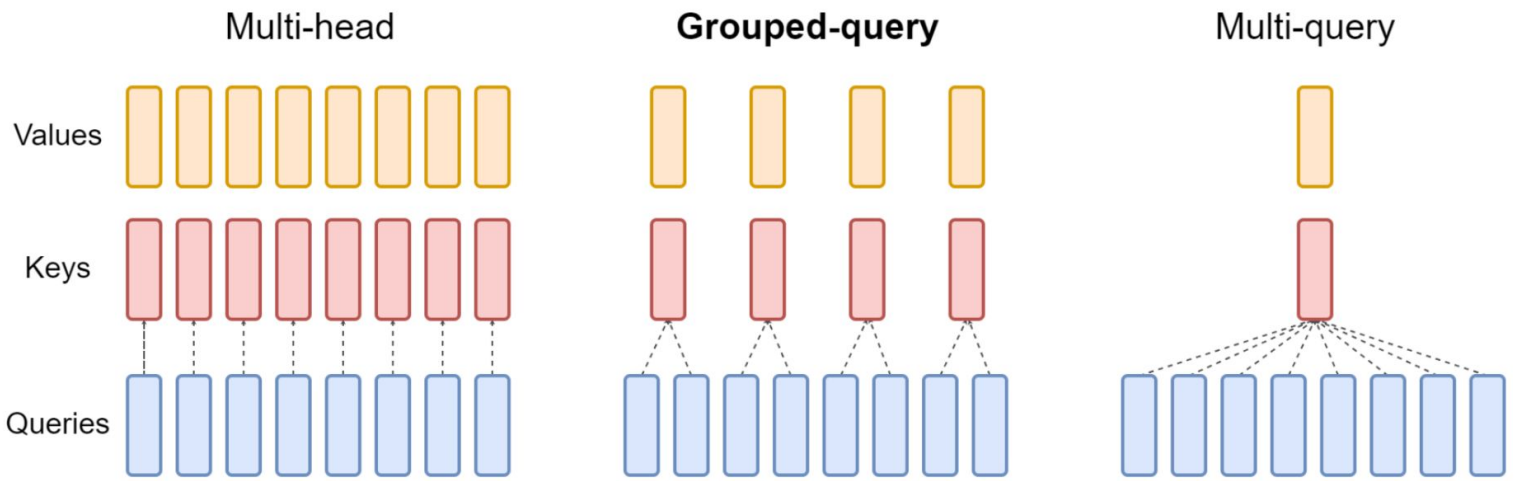
https://kaiokendev.github.io/til#extending-context-to-8k

Group Query Attention

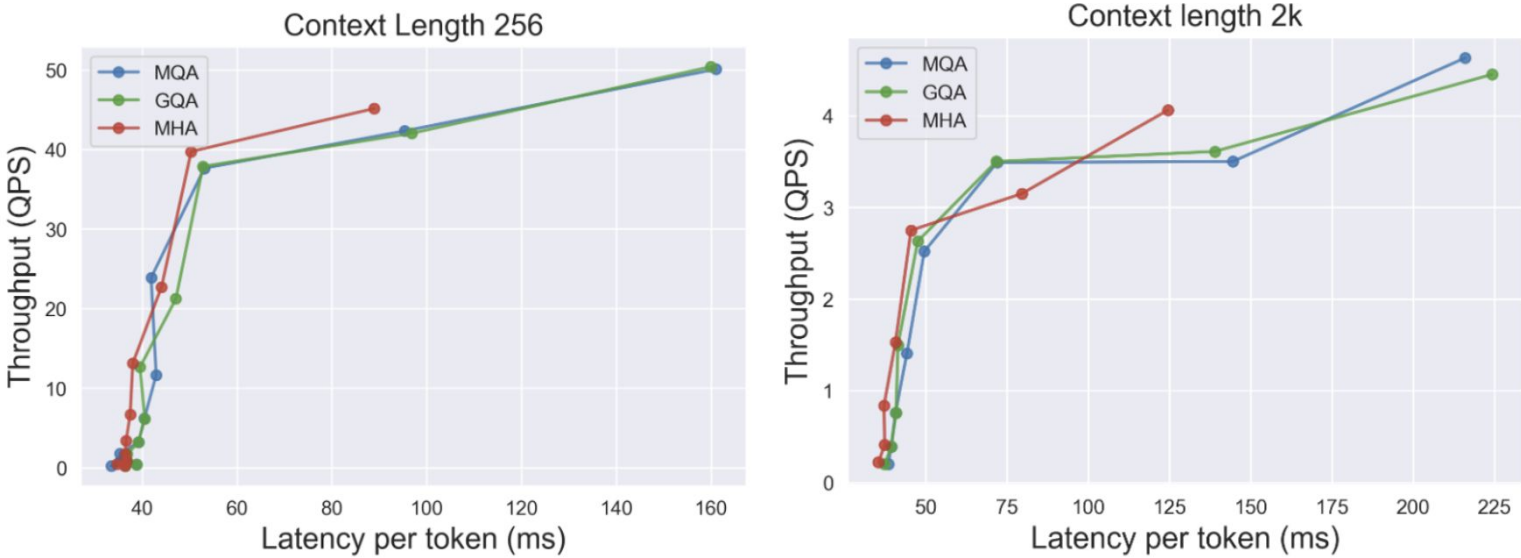


For more in-depth discussion, see this great post: https://kipp.ly/transformer-inference-arithmetic

Grouped Query Attention



Grouped Query Attention



Thanks to the

- https://www.stateof.ai (for the template)
- Lucas Beyer (https://www.youtube.com/watch?v=EixI6t5oif0) for the slides
- Illustrated transformer http://jalammar.github.io/illustrated-transformer/
- https://kipp.ly/transformer-taxonomy

Next time we'll talk about LLMs