Transformers

DNN lecture, University of Warsaw

Acknowledgements

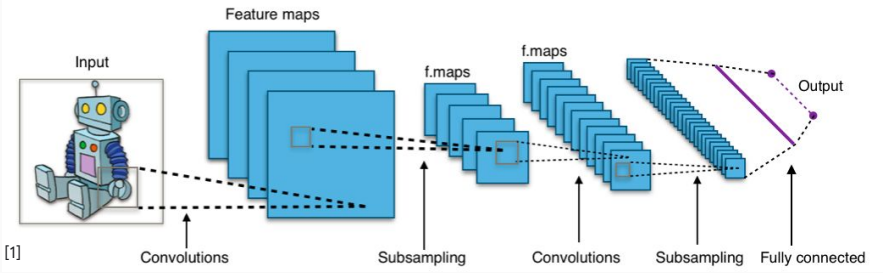
- A lot of those slides were made by Lucas Beyer from Google Research Zurich (lbeyer@google.com), who made them publicly available.

The classic landscape:

One architecture per "community"

Computer Vision

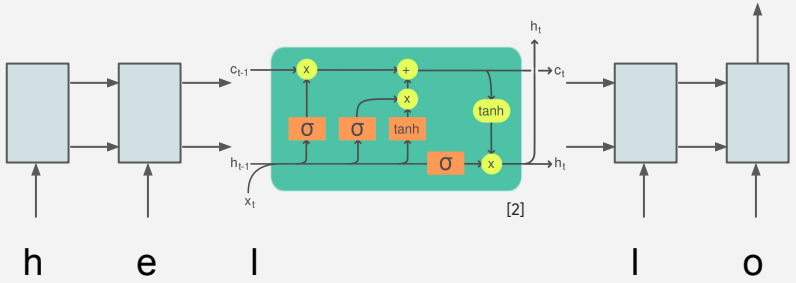
Convolutional NNs (+ResNets)



[1]

Natural Lang. Proc.

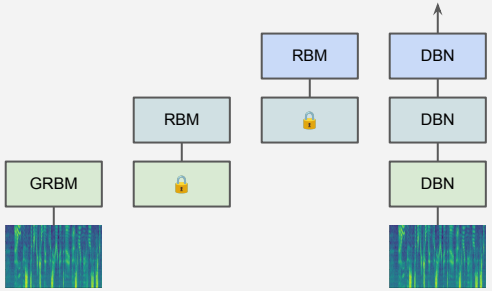
Recurrent NNs (+LSTMs)



h e l l o

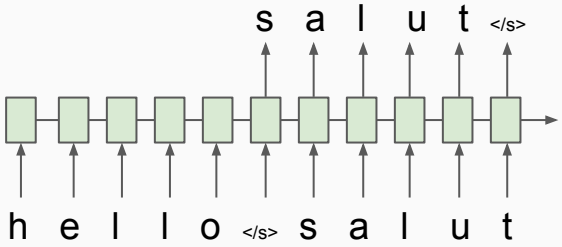
Speech

Deep Belief Nets (+non-DL)



Translation

Seq2Seq



RL

BC/GAIL

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{17}$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$$
$$\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \tag{18}$$

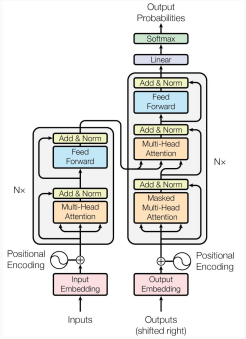
6: **end for**

[1] CNN image CC-BY-SA by Aphex34 for Wikipedia https://commons.wikimedia.org/wiki/File:Typical_cnn.png
[2] RNN image CC-BY-SA by GChe for Wikipedia https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg

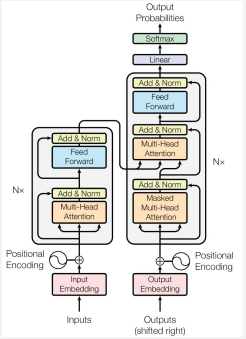
The Transformer's takeover:

One community at a time

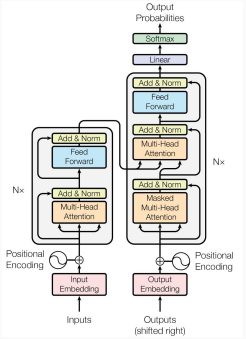
Computer Vision



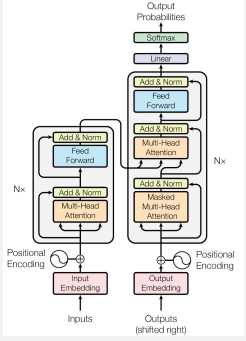
Natural Lang. Proc.



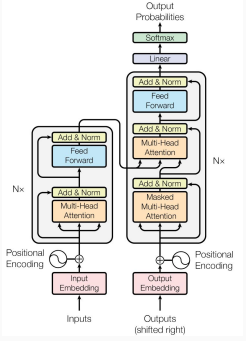
Reinf. Learning



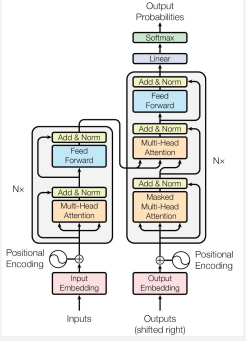
Speech



Translation

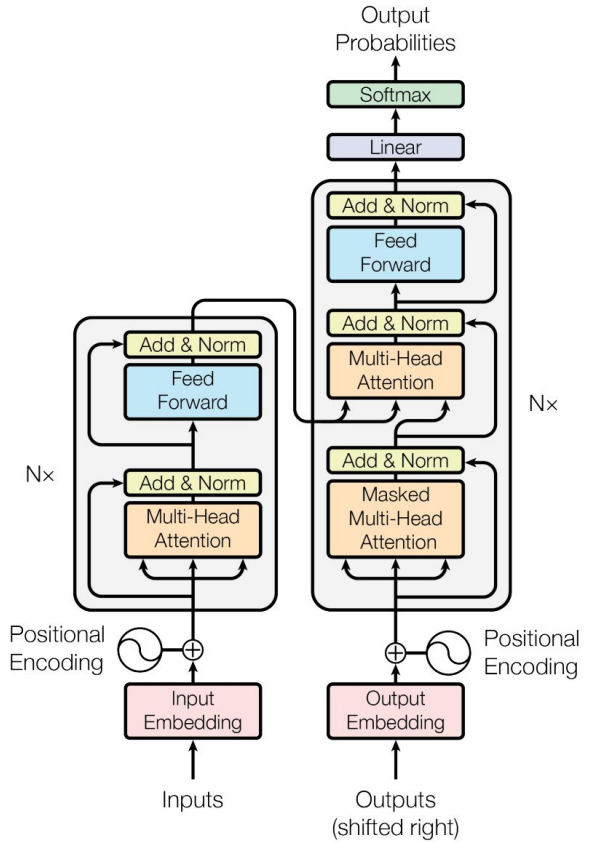


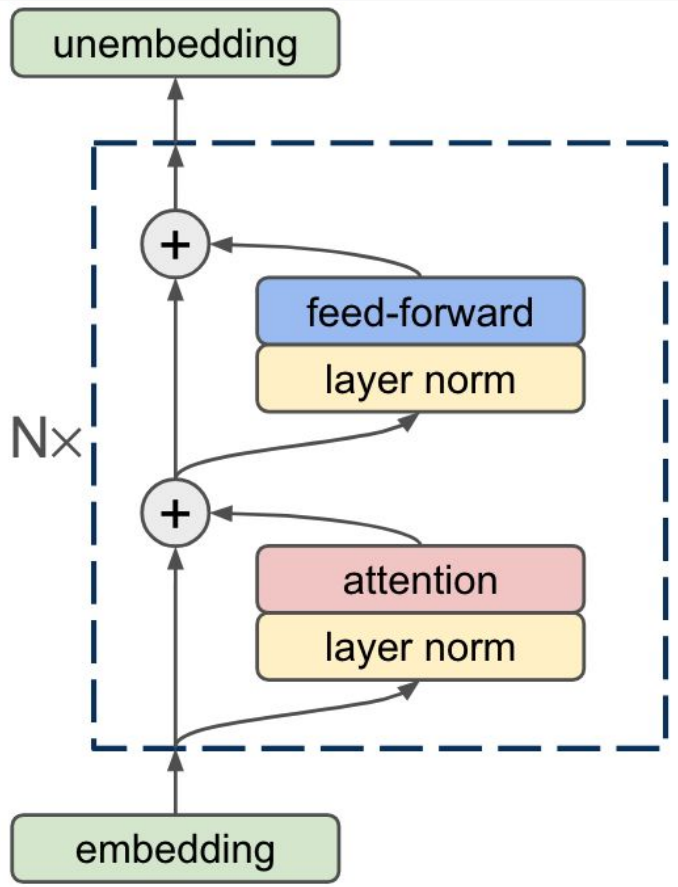
Graphs/Science



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

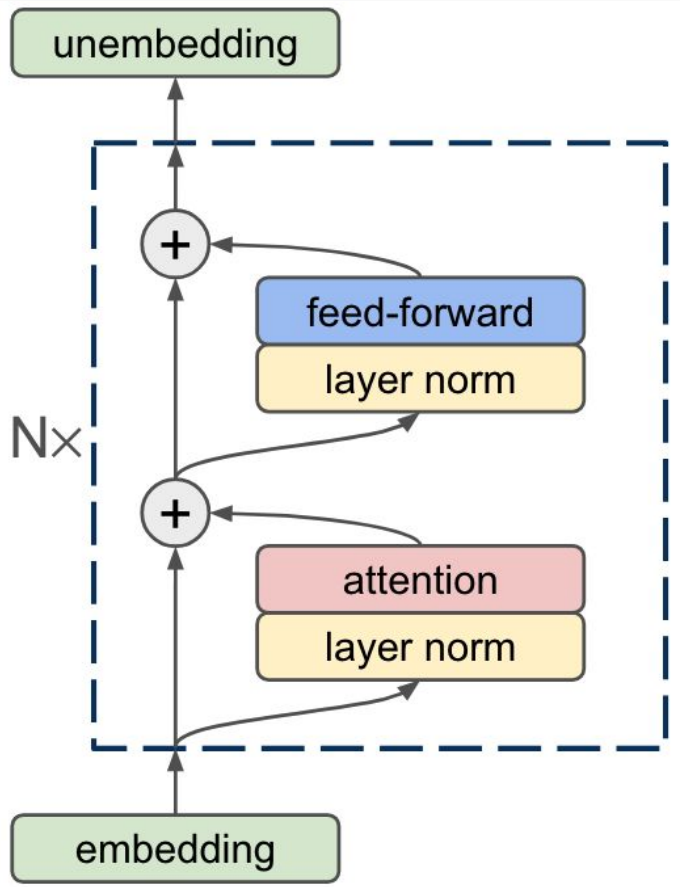
Modern Transformer





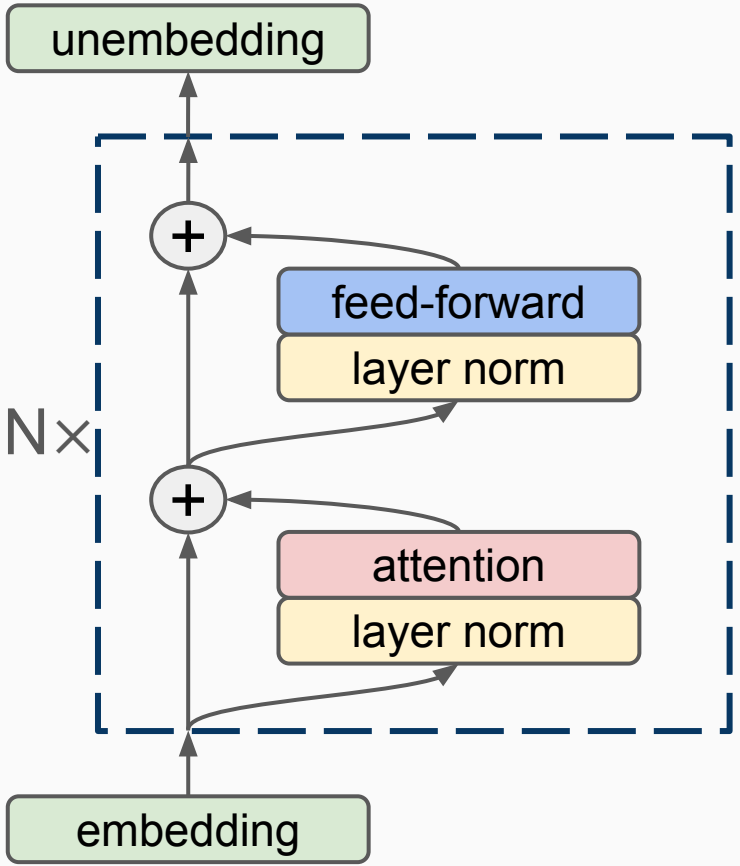
Modern Transformer

- embedding, unembedding
  - vocabulary -> representation
- residual stream
  - the representation of tokens!
  - updated with each layer
- feed-forward
  - different activations
  - most of parameters
  - often most of FLOPs
- attention
  - mixes information between tokens
- layer norm
  - for training stability



The diagram

- Task: LM, language modelling
  - that is, next-token prediction
  - sometimes different tasks, but we focus on LM
- Two modes of execution:
  - parallel (training / eval)
  - sequential (inference/generation)
- Mixed mode is possible
  - inference: prompt + generation



The origins:

Translation, learned alignment

Attention Is All You Need

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

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:

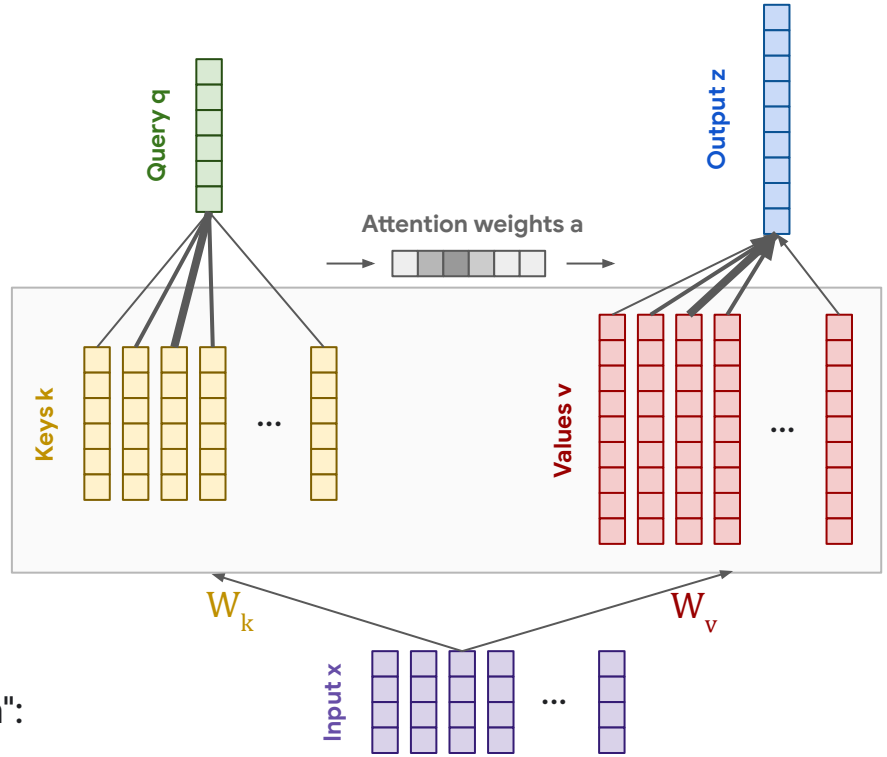
   $$k = W_k \cdot x \qquad v = W_v \cdot x$$

   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":

   $$q = W_q \cdot x$$



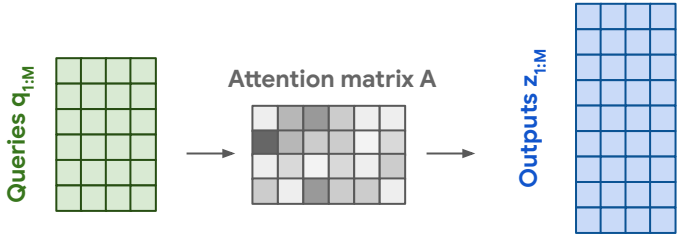
Attention Is All You Need

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

But that's not actually it! There are a few more details:

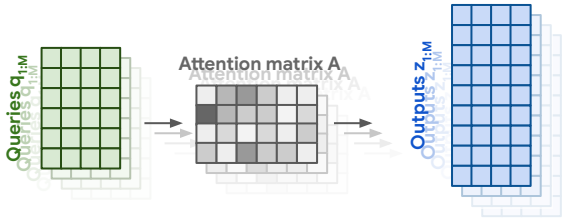
1. We usually use **many queries** $q_{1:M}$, not just one.
   Stacking them leads to the Attention matrix $\mathbf{A}_{1:N,1:M}$
   and subsequently to many outputs:

   $$z_{1:M} = \text{Attn}(q_{1:M}, x) = [\text{Attn}(q_1, x) \mid \text{Attn}(q_2, x) \mid \ldots \mid \text{Attn}(q_M, x)]$$



2. We usually use "**multi-head**" attention. This means the operation is repeated K times and the results are concatenated along the feature dimension. **W**s differ.

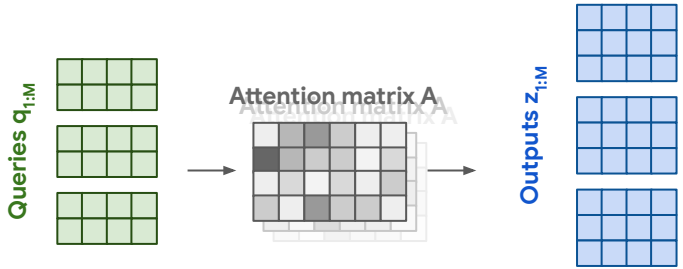
   $$z_i = \begin{pmatrix} \text{Attn}_1(q_i, x) \\ \text{Attn}_2(q_i, x) \\ \ldots. \\ \text{Attn}_K(q_i, x) \end{pmatrix}$$



3. The most commonly seen formulation:

   $$z = \text{softmax}(QK'/\sqrt{d_{key}})V$$

   Note that the complexity is $O(N^2)$, with N=#tokens



Key-Value cache

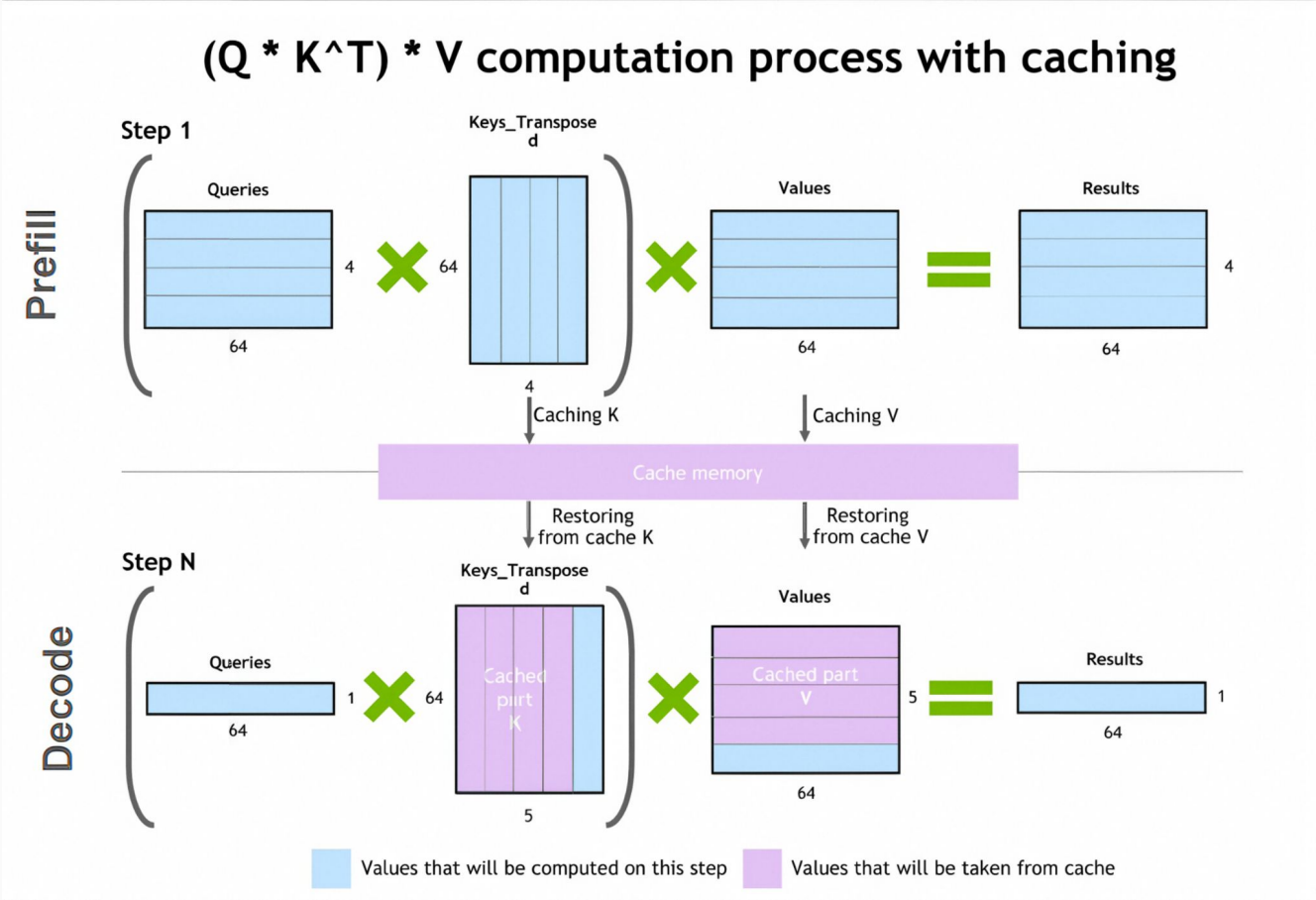
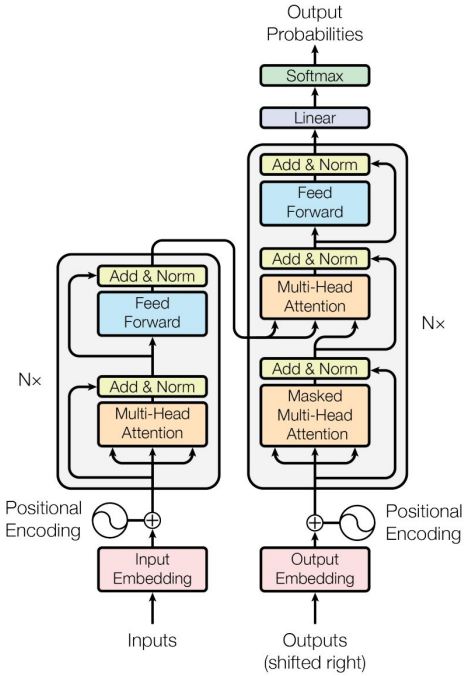


image from Shashank Verma and Neal Vaidyam, Nvidia, https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/, upscaled with Nero AI

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

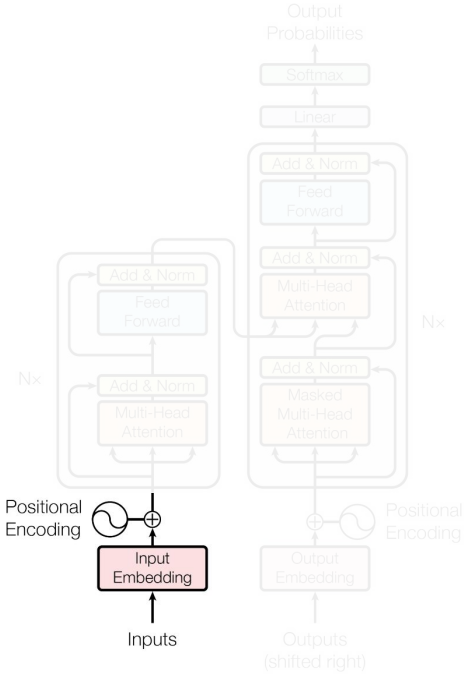


Thanks to Basil Mustafa for slide inspiration

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

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



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!

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

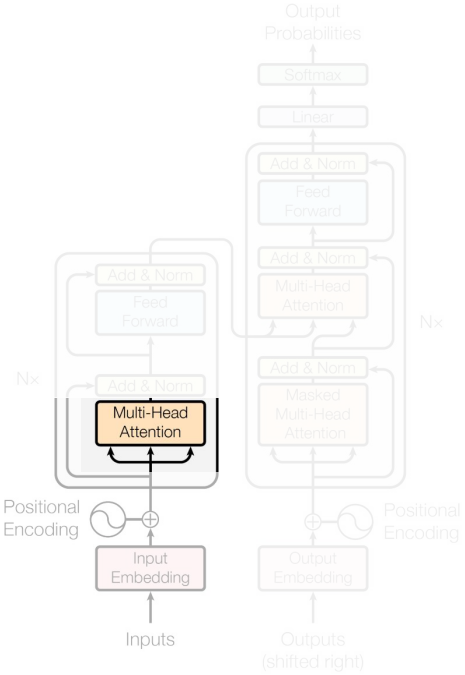
Think `[The_] + 10` `[detective_] + 20` `[invest] + 30` ... but smarter.

Thanks to Basil Mustafa for slide inspiration

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

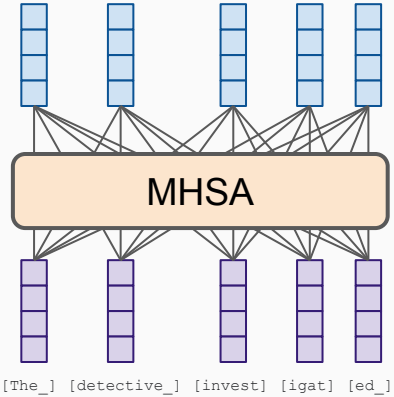
Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



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.

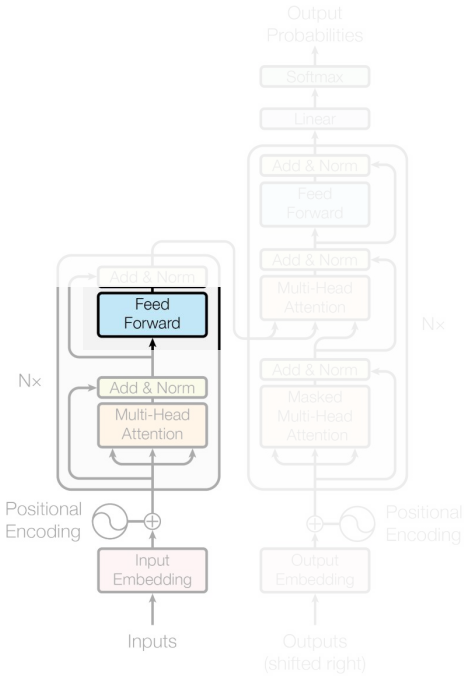


Thanks to Basil Mustafa for slide inspiration

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

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



Point-wise MLP

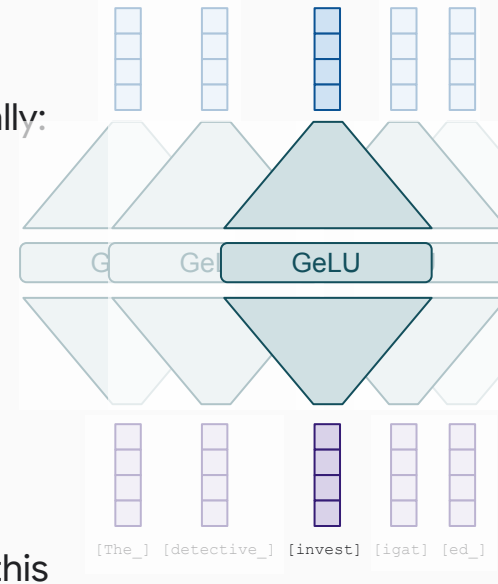
A simple MLP applied to each token individually:

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

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.

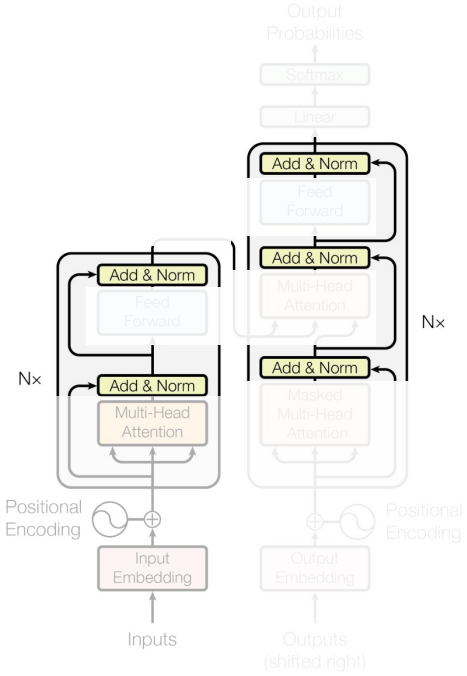


Thanks to Basil Mustafa for slide inspiration

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

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



Residual connections

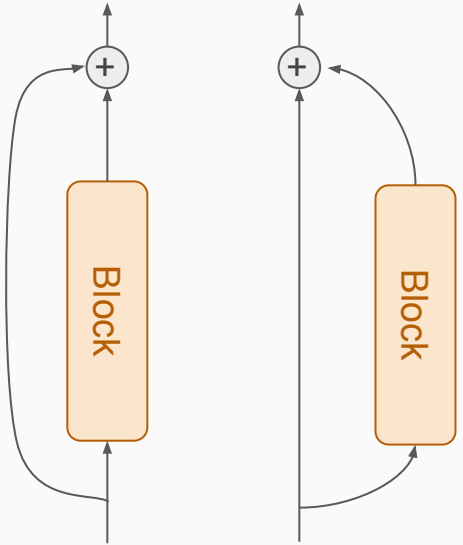
Each module's output has the exact same shape as its input.

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.

"Skip connection" "Residual block"



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)

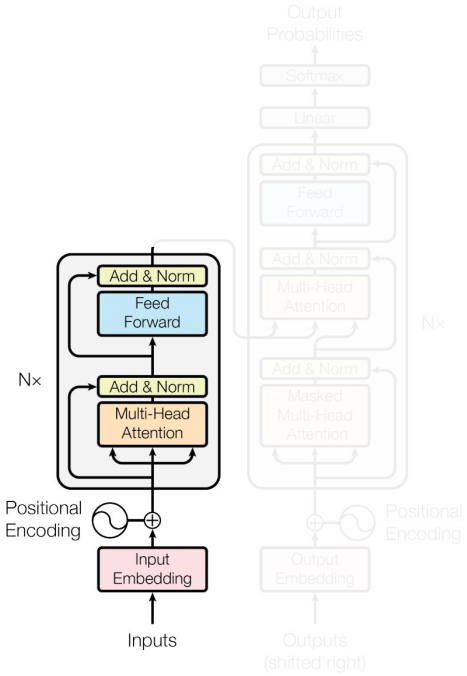
$$z_i = \text{Module}(\text{LN}(x_i)) + x_i$$

Thanks to Basil Mustafa for slide inspiration

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

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



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.

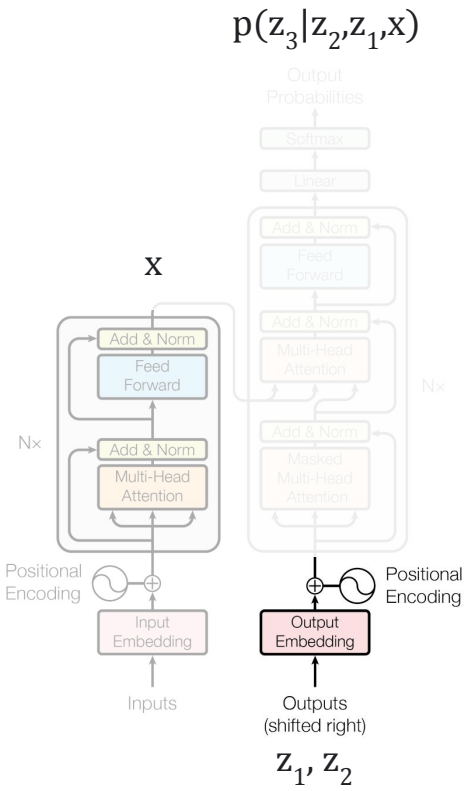
This has nothing to do with the requested output yet (think: translation). That comes with the decoder.

Thanks to Basil Mustafa for slide inspiration

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

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



Thanks to Basil Mustafa for slide inspiration

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: $p(z \mid \text{"the detective investigated"})$ **∀z**

Seems impossible at first, but 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 $p(z_3|z_2,z_1,x)$:

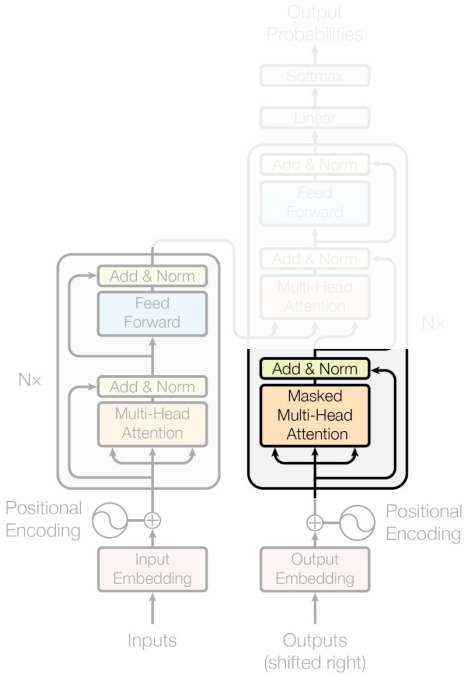
x comes from the encoder,

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, ...

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



Masked self-attention

This is regular self-attention as in the encoder, to process what's been decoded so far, 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

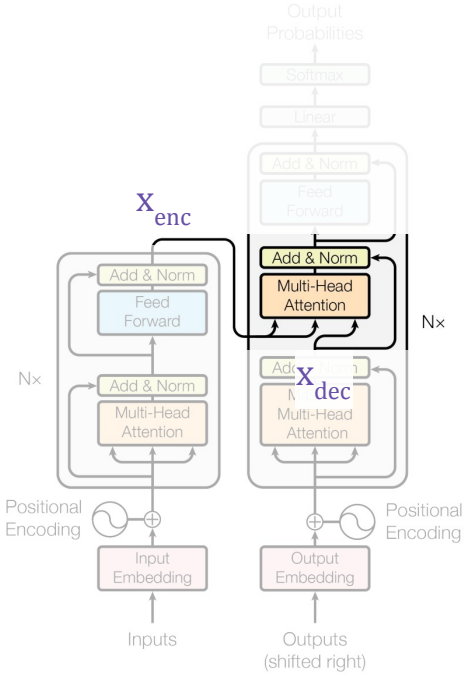
There is no such trick. We need to generate one z_i at a time. This is why autoregressive decoding is extremely slow.

Thanks to Basil Mustafa for slide inspiration

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

Attention Is All You Need - The Transformer architecture

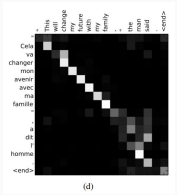
2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



"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.

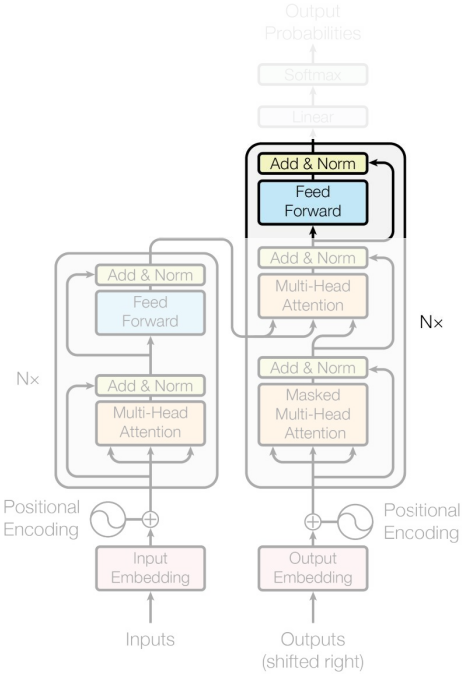
Because self-attention is so widely used, people have started just calling it "attention".
Hence, we now often need to explicitly call this "cross attention".

Thanks to Basil Mustafa for slide inspiration

Transformer image source: "Attention Is All You Need" paper ; heatmap image source: "Neural Machine Translation by Jointly Learning to Align and Translate" paper

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



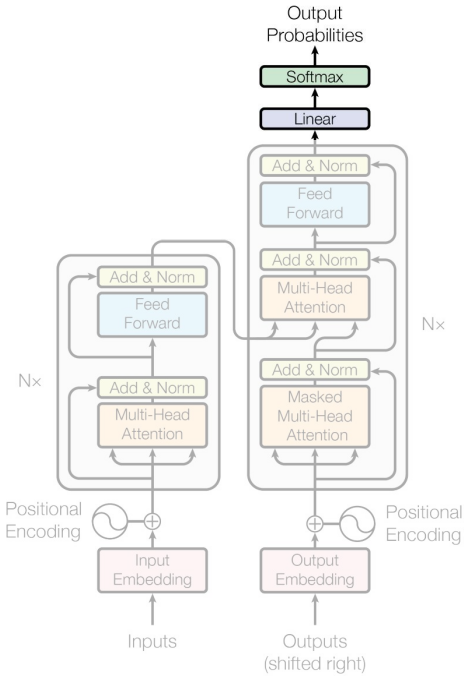
Feedforward and stack layers.

Thanks to Basil Mustafa for slide inspiration

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

Attention Is All You Need - The Transformer architecture

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



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

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

Attention Is All You Need - Summary and results

2017, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin

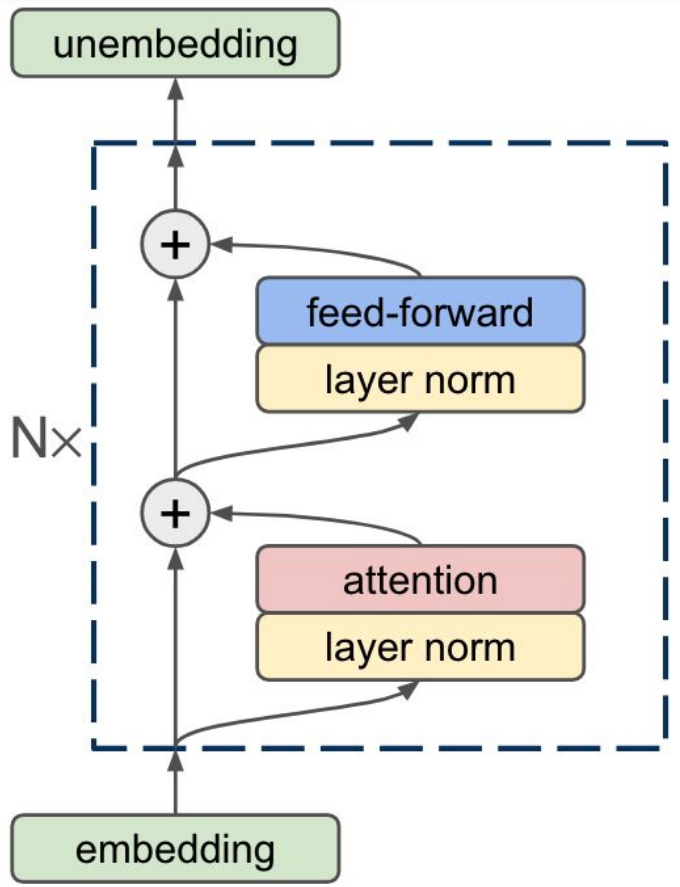
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Animation source: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html
Table source: "Attention Is All You Need" paper

Modern Transformer

- embedding, unembedding
  - vocabulary -> representation
- residual stream
  - the representation of tokens!
  - updated with each layer
- feed-forward
  - different activations
  - most of parameters
  - often most of FLOPs
- attention
  - mixes information between tokens
- layer norm
  - for training stability



Number of Parameters

Caveats

- All numbers are in number of floats. Multiply by 2 or 4 to get bytes.
- Everything is estimated, back-of-envelope calculation.
    - 2x or 3x mistake is fine. First we want to develop mental model of what is happening, precision is less important for overview.

Simple case - linear layer

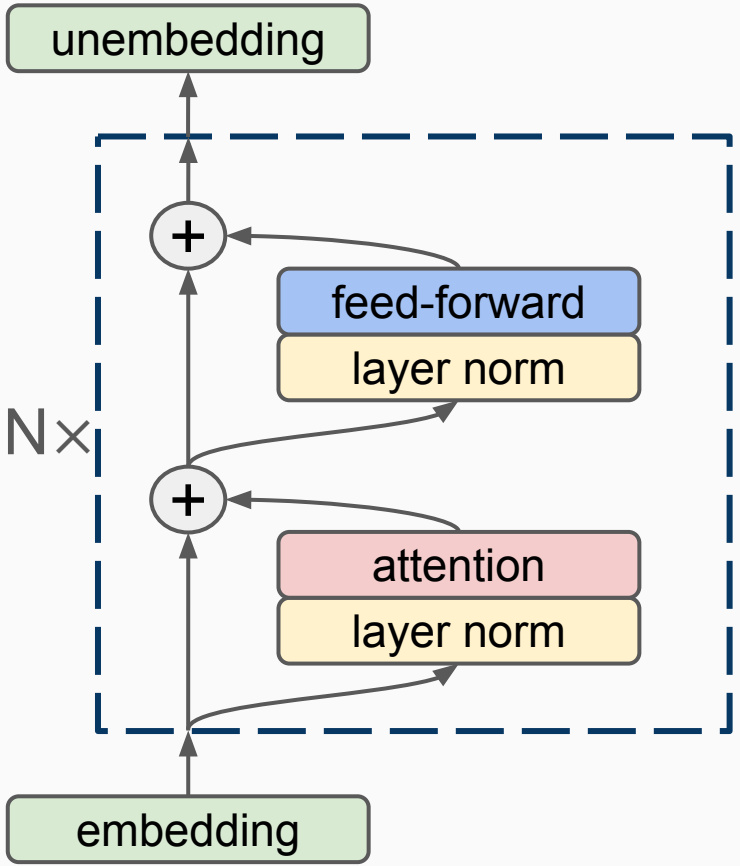
For a linear projection, f(x) = A * x + b.
A: <d_input, d_output>
b: <d_input>

Usually d_input and d_output is one of:

- d_model (GPT3: 12k; Llama-8B: 4k)
- expansion_rate * d_model (between 3 and 6)
- vocab_size (usually 30k - 120k)

The Transformer

- embedding
  - vocab_size * d_model
- unembedding
  - vocab_size * d_model
- residual stream
  - basically none
- feed-forward
  - 2 * d_model * (4 * d_model)
- attention
  - 4 * (d_model * d_model)
  - Key, Query, Value, Output
- layer norm
  - negligible, like biases; O(d_model)



FLOPs

FLOPs - Floating Point Operations

- The number of operations, most of them being simple multiplications and additions.
- peak performance on GPUs is only achievable during matmul
- remember to count both multiplication and addition (reduce_sum)!

FLOPs - linear layer

For a linear projection, f(x) = A * x (no biases)

A: <d_input, d_output>

b: <n_tokens, d_input>

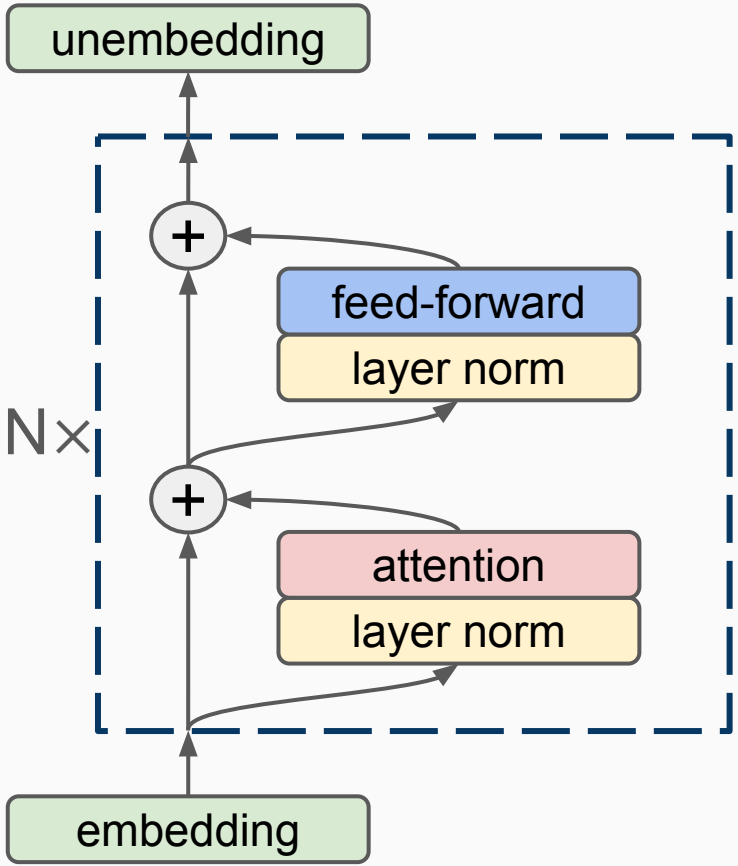
output: <n_tokens, d_output>

FLOPs = 2 * n_tokens * d_input * d_output

FLOPs - The Transformer

Flops **PER TOKEN**.

- embedding
  - d_model (index select)
- unembedding
  - 2* d_model * vocab_size
- residual stream / layer norm
  - basically none, O(d_ model)
- feed-forward
  - **2 * 2 * d_model * (4 * d_model)**
- attention
  - KQVO: 2 * 4 * (d_model * d_model)
  - mechanism: **2*2*n_tokens*d_model**



FLOPs in training

FLOPs in training

For each "visible" matmul in forward pass, we need to do three matmuls during training.

- one for computing forward pass
- one for computing gradients wrt parameters
- one for computing gradients wrt input

If we are doing activation checkpointing:

- one additional matmul for re-computing forward pass

FLOPs in training - linear layer

For a linear projection, f(x) = A * x (no biases)

FLOPs eval/inference: 2 * n_tokens * d_input * d_output

FLOPs training: 3 * 2 * n_tokens * d_input * d_output

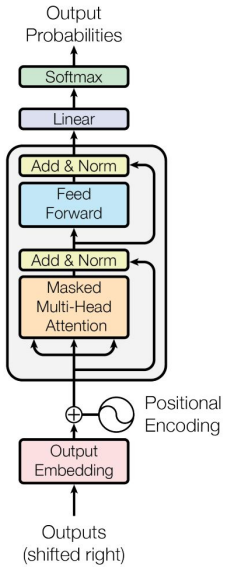
FLOPs training + activation checkpointing: 4 * 2 * n_tokens * d_input * d_output

The first big takeover:

Language Modeling / NLP

Decoder-only GPT

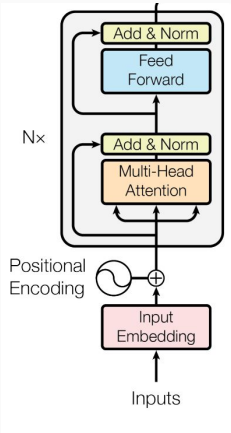
[sat_]



[START] [The_] [cat_]

Encoder-only BERT

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



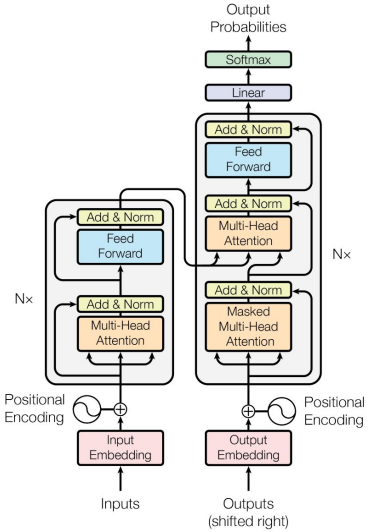
[The_] [cat_] **[MASK]** [on_] **[MASK]** [mat_]

Enc-Dec 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!

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

The second big takeover:

Computer Vision

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

2020, A Dosovitskiy, L Beyer, A Kolesnikov, D Weissenborn, X Zhai, T Unterthiner, M Dehghani, M Minderer, G Heigold, S Gelly, J Uszkoreit, N Houlsby

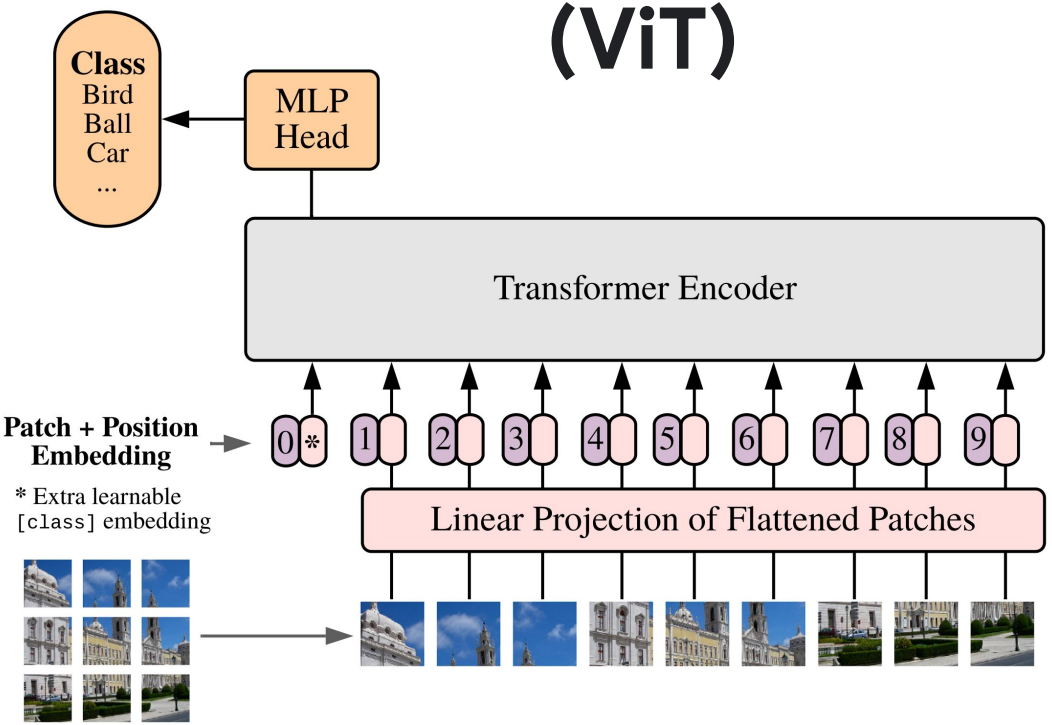
Many prior works attempted to introduce self-attention at the pixel level.

For 224px^2, 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 16px^2, and treating each patch as a token, e.g. embedding it into input space.

Vision Transformer (ViT)



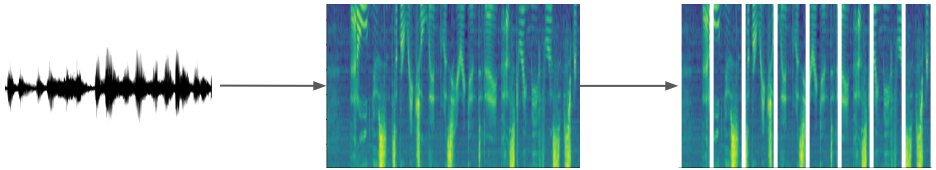
The third big takeover:

Speech

Conformer: Convolution-augmented Transformer for Speech Recognition

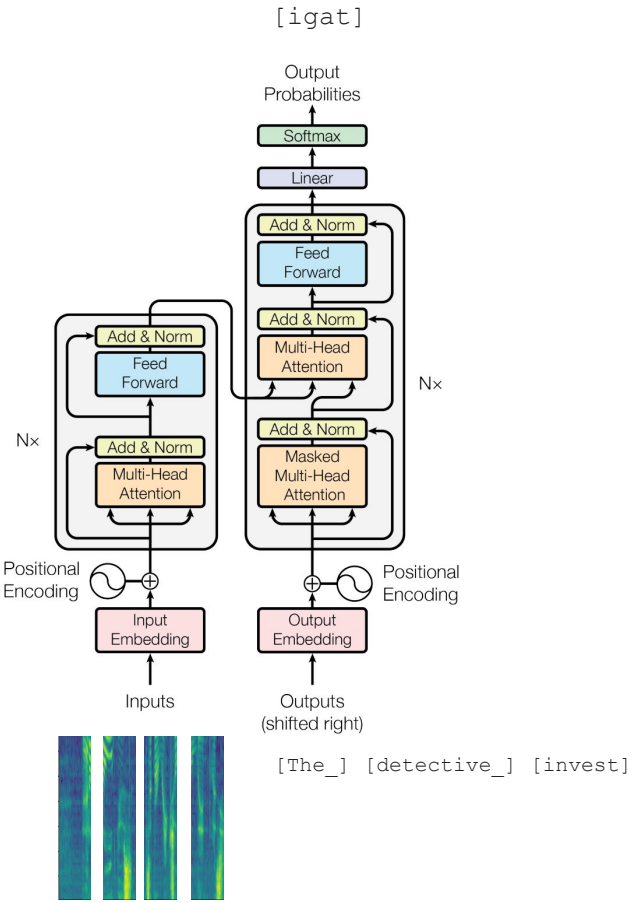
2020, A Gulati, J Qin, C-C Chiu, N Parmar, Y Zhang, J Yu, W Han, S Wang, Z Zhang, Y Wu, R Pang

Largely the same story as in computer vision.
But with spectrograms instead of images.



Add a third type of block using convolutions, and slightly reorder blocks, but overall very transformer-like.

Exists as encoder-decoder variant, or as encoder-only variant with CTC loss.



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

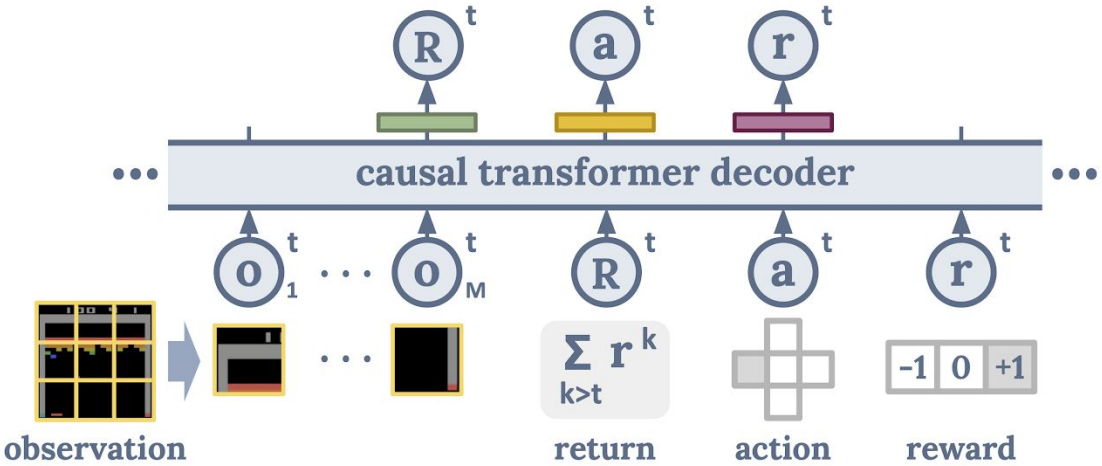
The fourth big takeover:

Reinforcement Learning

Decision Transformer: Reinforcement Learning via Sequence Modeling

2021, L Chen, K Lu, A Rajeswaran, K Lee, A Grover, M Laskin, P Abbeel, A Srinivas, I Mordatch

Cast the (supervised/offline) RL problem into a sequence ("language") modeling task:



Can generate/decode sequences of actions with desired return (eg skill)

The trick is prompting: `"The following is a trajectory of an expert player: [obs] ..."`

Slide credit: Igor Mordatch

The Transformer's

Unification of communities

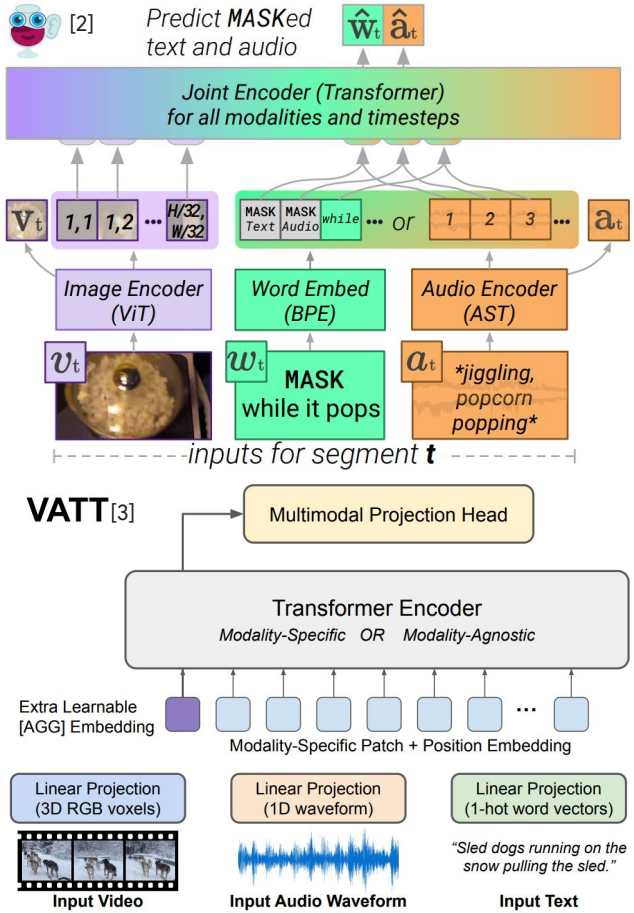
Anything you can tokenize, you can feed to Transformer

ca 2021 and onwards

Tokenize different modalities each in their own way (some kind of "patching"), and send them all jointly into a Transformer...

Seems to just work...
Currently an explosion of works doing this!

[1]



Images from:
[1] LIMoE by B Mustafa, C Riquelme, J Puigcerver, R Jenatton, N Houlsby
[2] MERLOT Reserve by R Zellers, J Lu, X Lu, Y Yu, Y Zhao, M Salehi, A Kusupati, J Hessel, A Farhadi, Y Choi
[3] VATT by H Akbari, L Yuan, R Qian, W-H Chuang, S-F Chang, Y Cui, B Gong

Speeding up Feed Forward layer: Conditional Computation

Intuitive explanation of conditional computing

Neurons in neural network encode information.

All information (knowledge of the model) is encoded in neurons.

A given neuron, probably, contains information regarding specific task.

In Transformer, every token is processed by all the neurons!

Even if information encoded in those neurons is irrelevant!

Conditional computation approaches

We can look at the problem from two perspectives:

- Increase model size while keeping computation budget.
- Decrease computation budget while keeping model size.

Switch Transformers:
Scaling to Trillion Parameter Models

Switch Transformer

Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity

William Fedus*
LIAMFEDUS@GOOGLE.COM

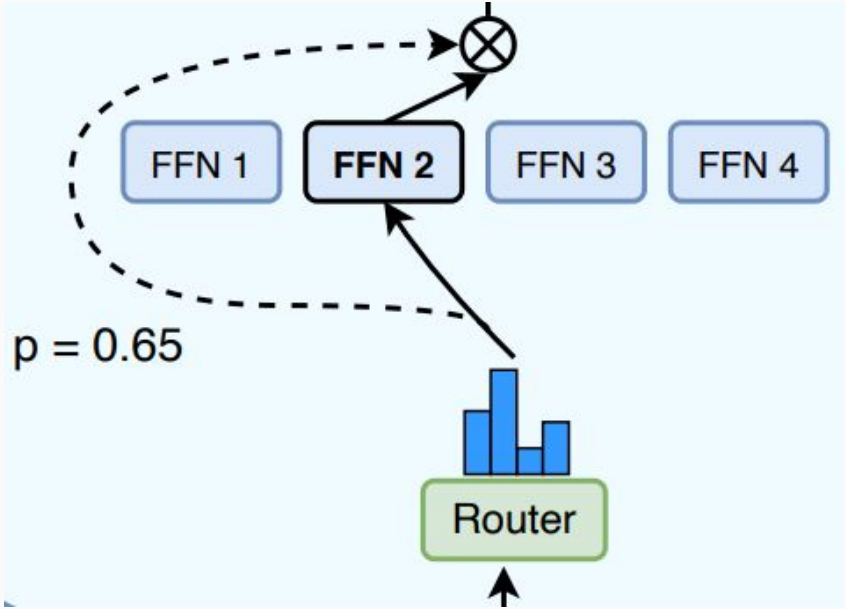
Barret Zoph*
BARRETZOPH@GOOGLE.COM

Noam Shazeer
NOAM@GOOGLE.COM
Google, Mountain View, CA 94043, USA

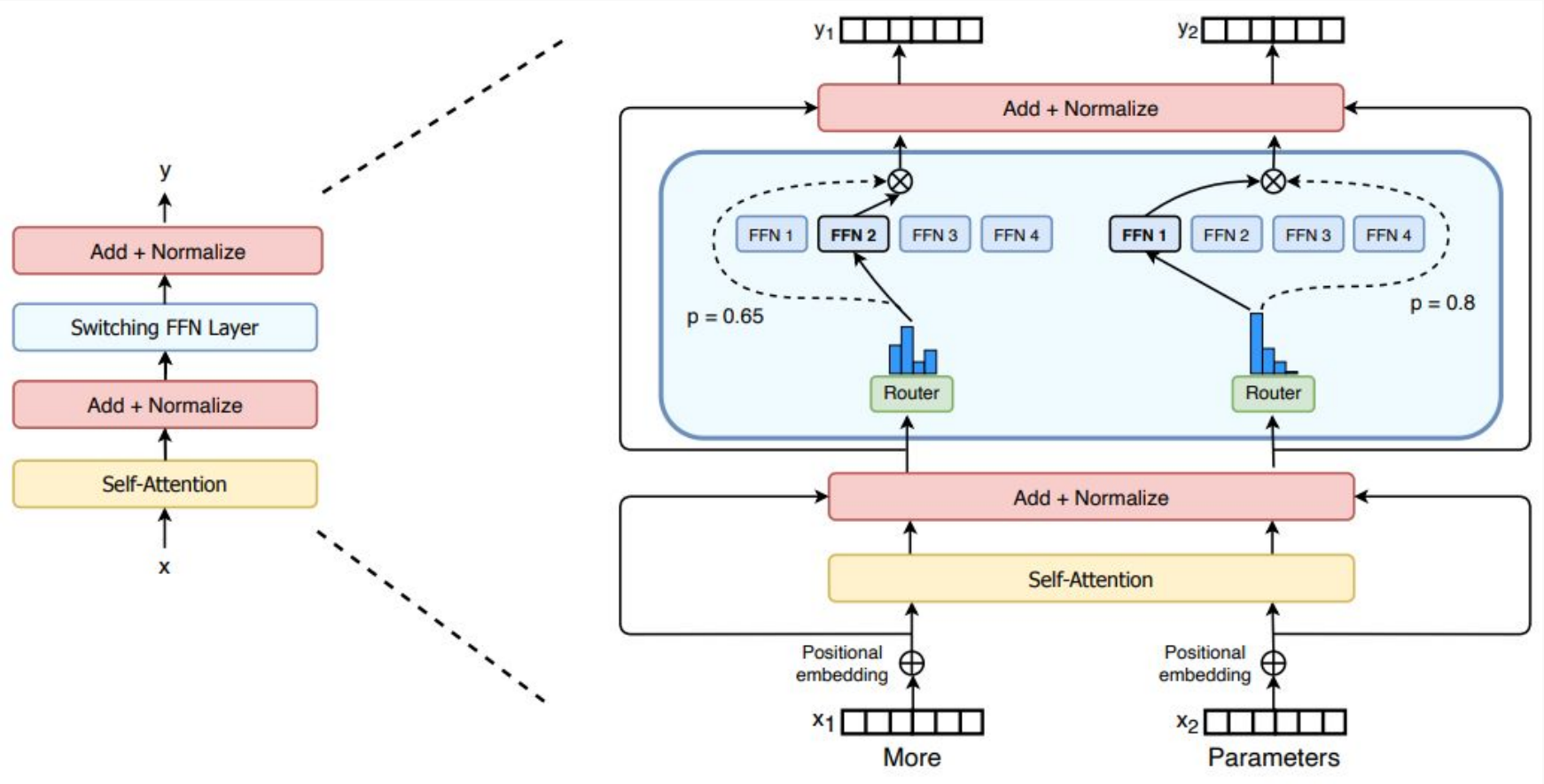
Switch Transformer

We duplicate Feed Forward layer N times, getting N experts.

We insert a router, which decides which version of the layer (which expert) to use.



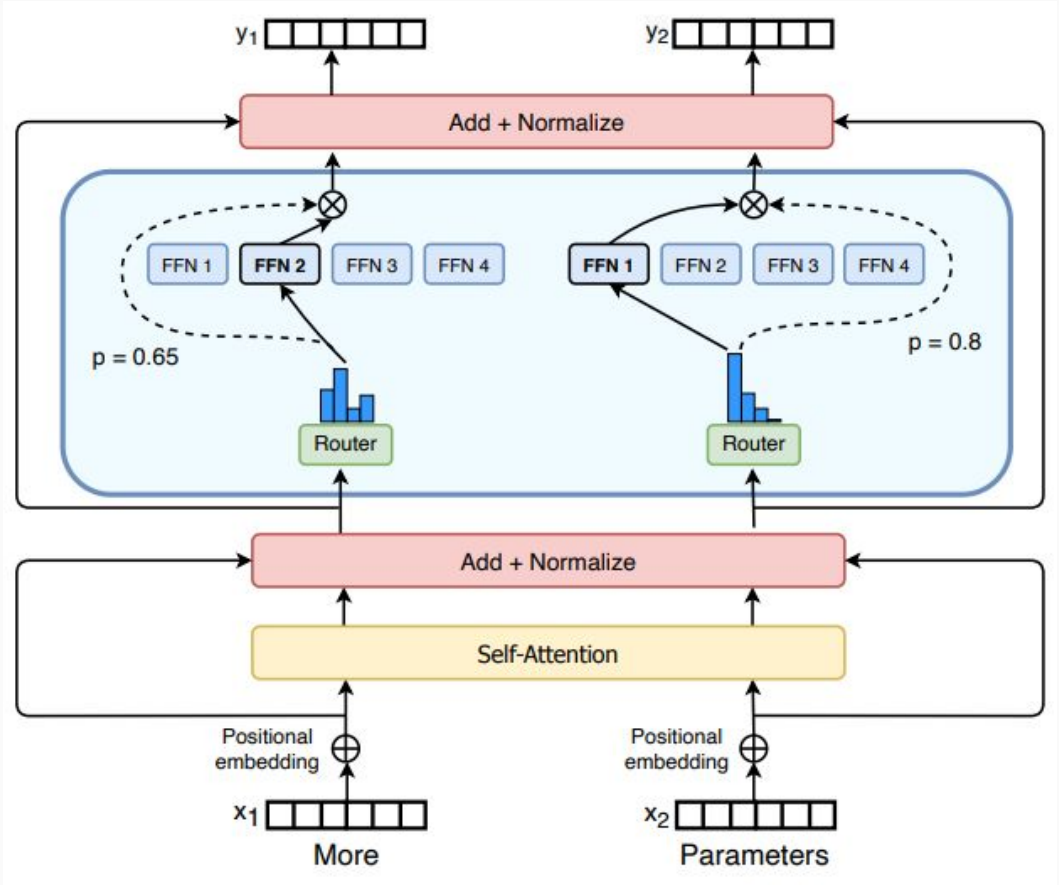
Switch Transformer



Switch Transformer

We can see on the picture that router chooses the expert for each token independently…

almost independently at least.

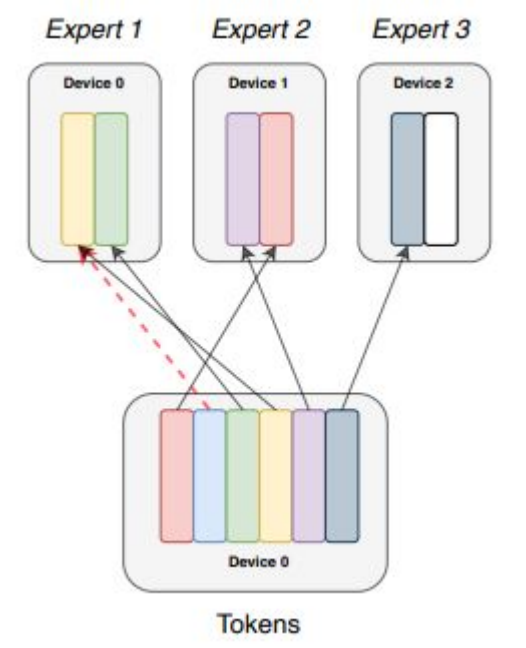


Switch Transformer

For technical reasons, to speed-up training as well, each expert receives some limited amount of tokens.

Often those different experts can be located on different devices!

If an expert receives too many tokens, some are not processed.

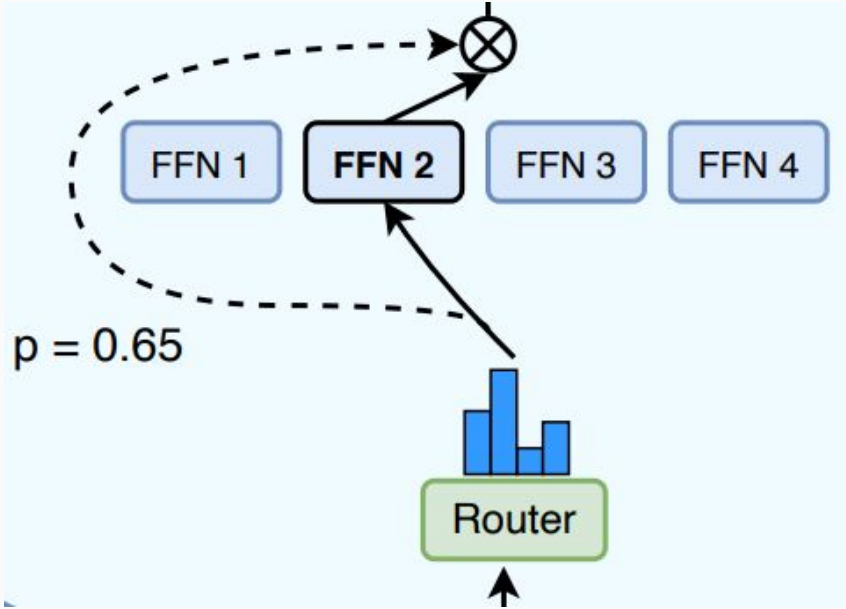


Switch Transformer

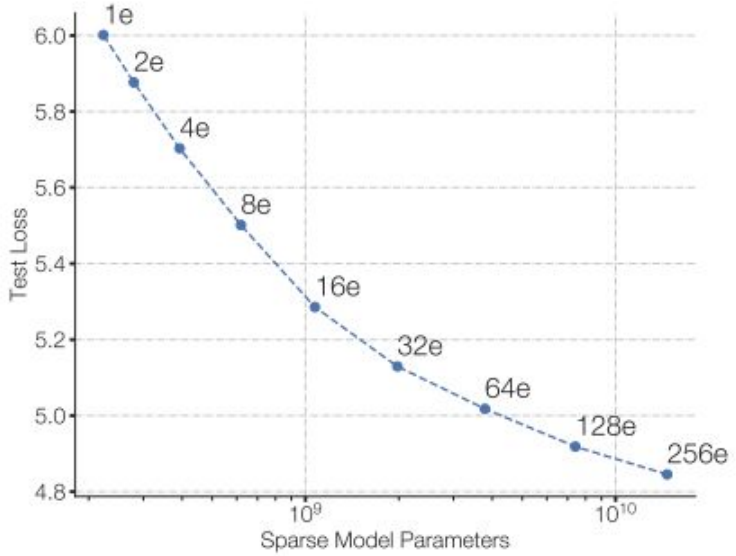
How can the training work?

We just get the probability of choosing a particular expert (0.65 on the picture), and multiply it by expert's output.

This allows for gradients to pass to the router!



Switch Transformer - results



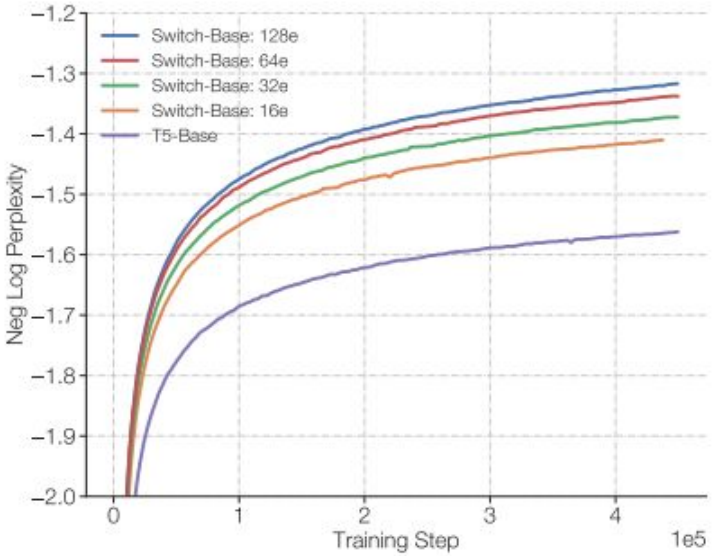
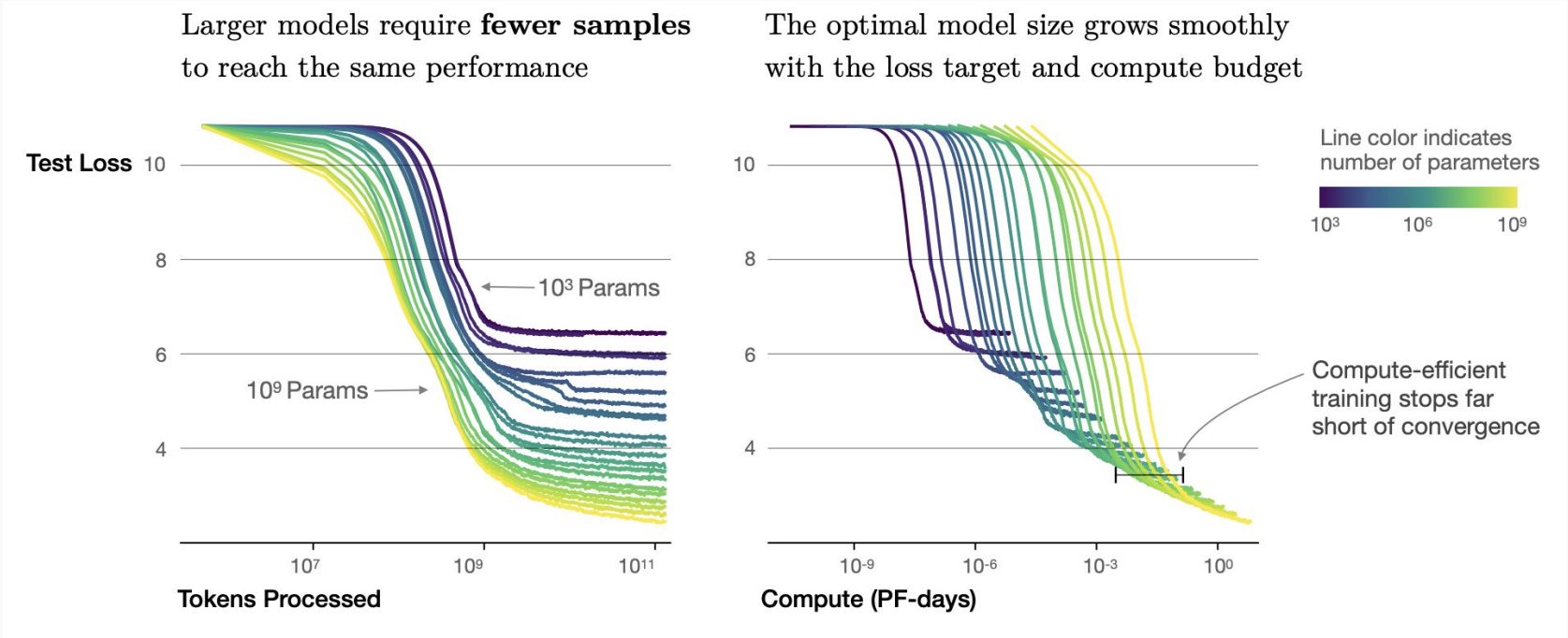


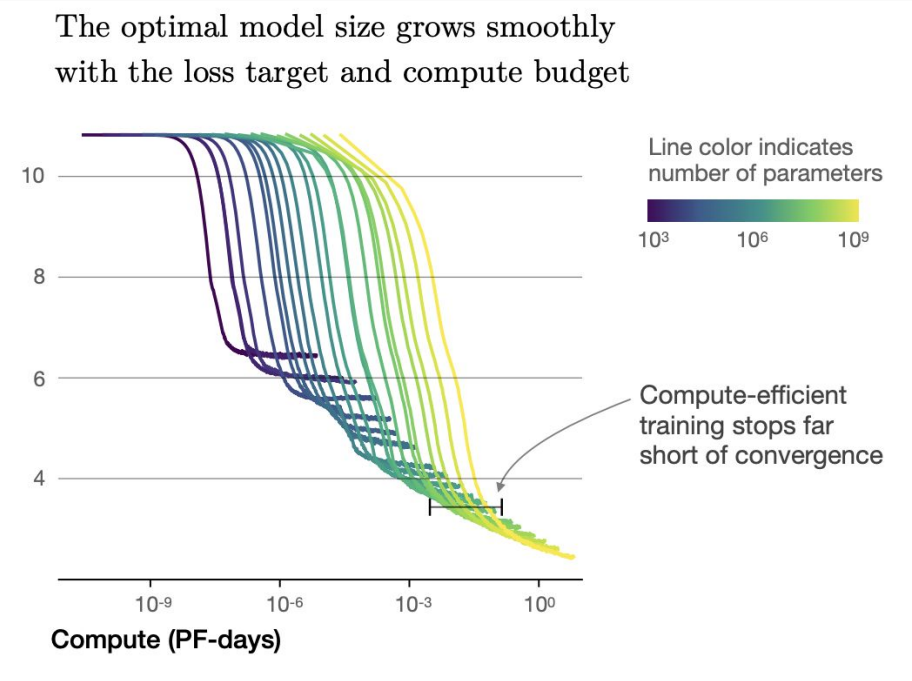
Figure 1: Scaling and sample efficiency of Switch Transformers. Left Plot: Scaling properties for increasingly sparse (more experts) Switch Transformers. Right Plot: Negative log perplexity comparing Switch Transformers to T5 (Raffel et al., 2019) models using the same compute budget.

Scaling Laws

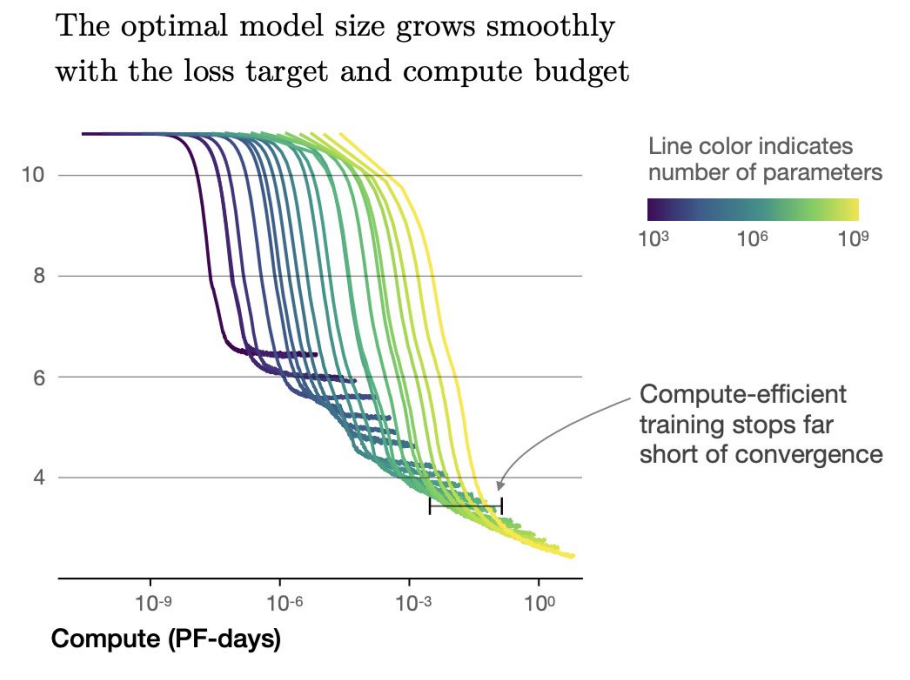
Scaling Laws: Kaplan et al.

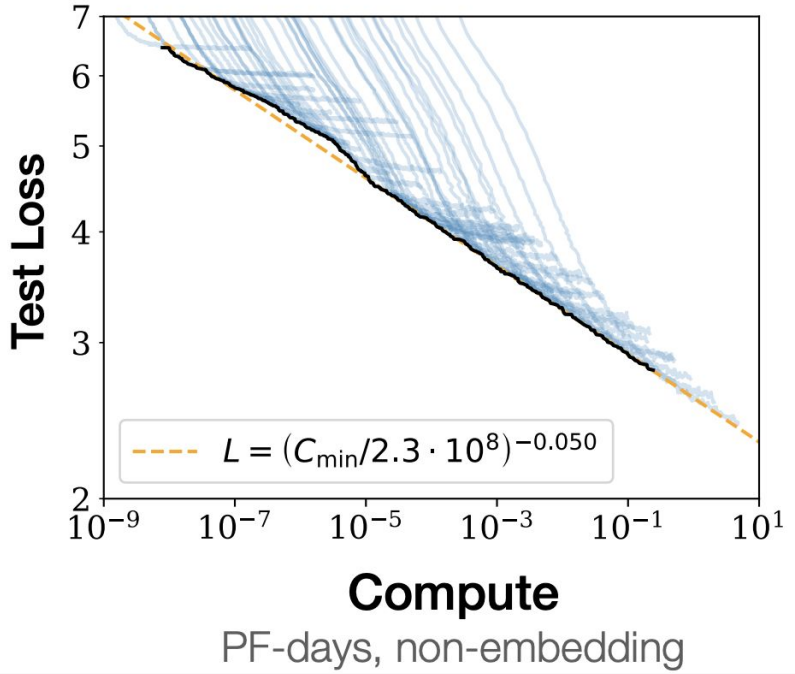


Scaling Laws: Kaplan et al. - Compute Optimal Models



Scaling Laws: Kaplan et al. - Compute Optimal Models





Scaling Laws: Kaplan et al.

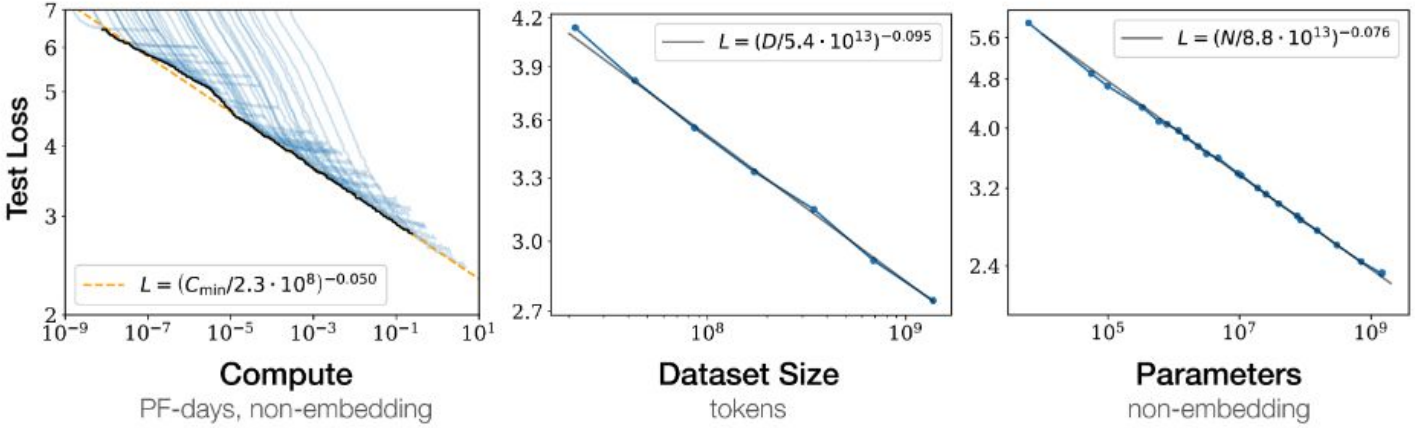


Figure 1 from Kaplan et al., 2020, *Scaling Laws for Neural Language Models*

Scaling Laws Research

Previous slides I presented figures from Kaplan et al. (2020, then-OpenAI)

Later paper by Hoffmann et al. (2022, then-DeepMind) refined Scaling Laws.

Feedback is a gift

http://tinyurl.com/dnn-2025-12-10