Deep Learning
Transformer and Newer Architectures

Hao Chen, Dareen Alharthi

Fall 2024
Attendance: xxx

Content

- Transformer Architecture
- Transformer in Language
- Transformer in Vision
- Transformer in Audio
- Parameter Efficient Tuning
- Scaling Laws

Why Transformer?

- Almost everything today in deep learning is **Transformer**

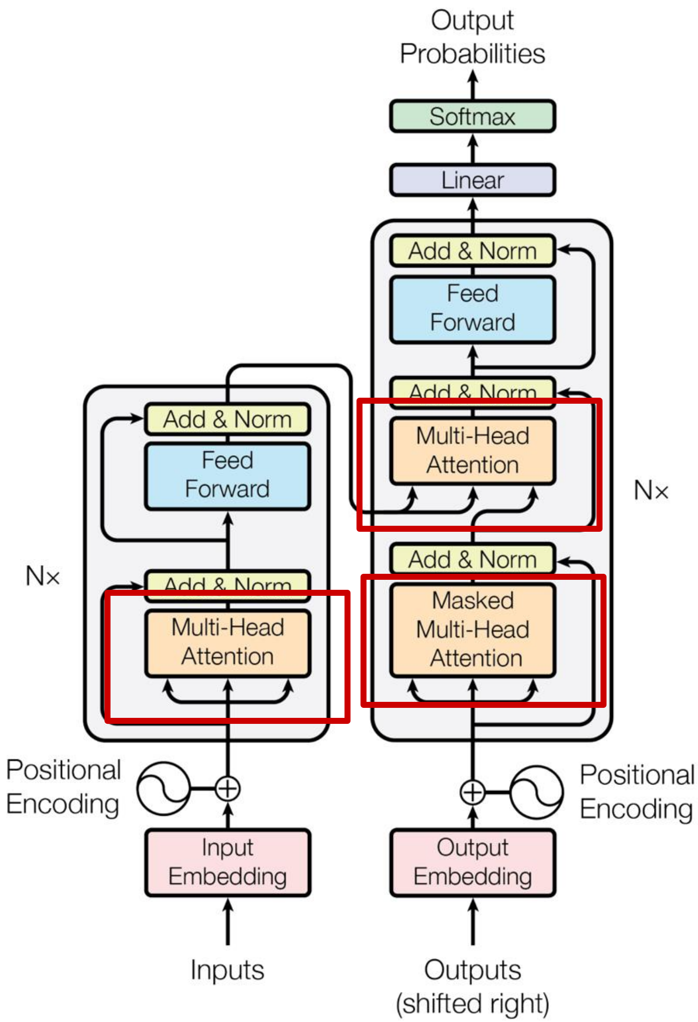
But…Why Transformer?

- Flexibility and universality of handling all modality
- Scaling with data and parameters
- “Emergent” capability and In-context Learning
- Parameter Efficient Tuning

Transformer Architecture

Transformer Architecture

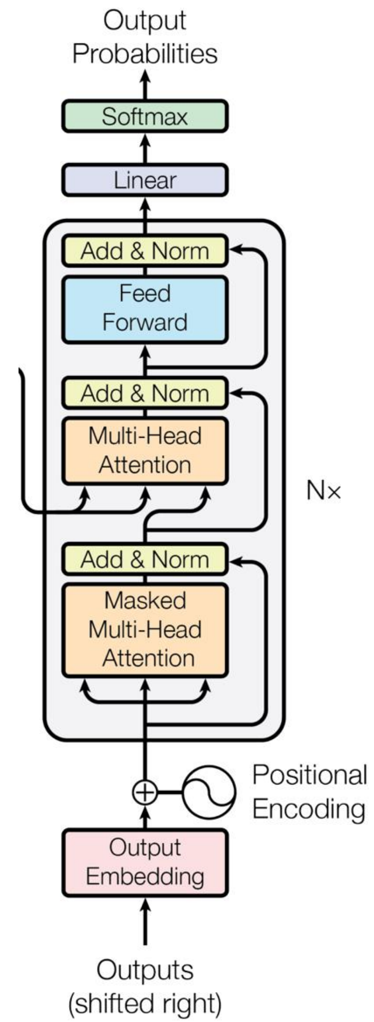
- overview



Vaswani, A. "Attention is all you need." *Advances in Neural Information Processing Systems* (2017).

Transformer Architecture

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer

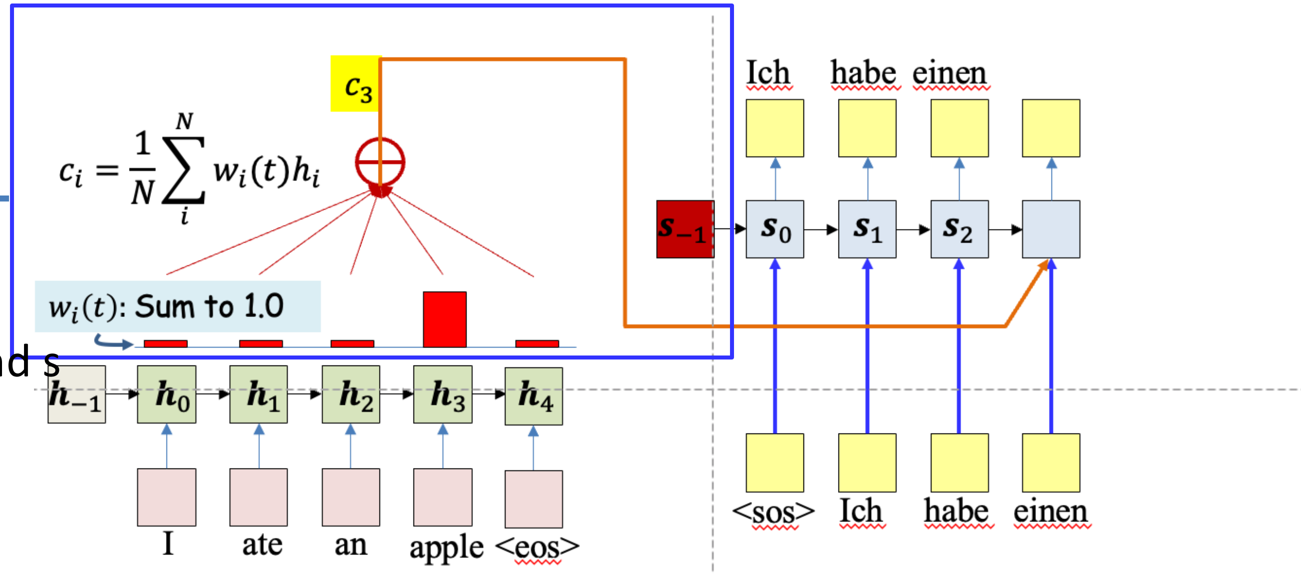


Recap: Attention in Seq2Seq Models

Attention

Weighted sum of h

Weights computed from h and s



- Typical options for $g()$ (**variables in red must learned**)

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{s}_{t-1}$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_{t-1}$$

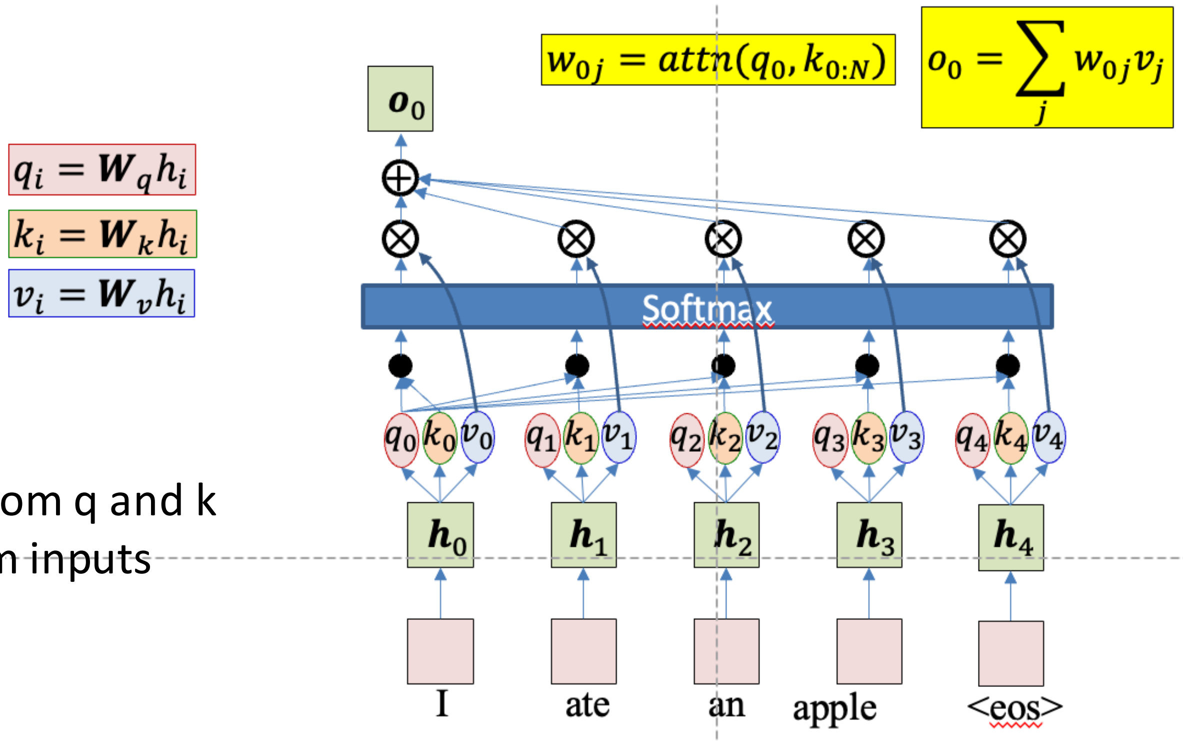
$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = \boldsymbol{v}_g^T \boldsymbol{tanh}\left(\boldsymbol{W}_g \begin{bmatrix} \boldsymbol{h}_i \\ \boldsymbol{s}_{t-1} \end{bmatrix}\right)$$

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1}) = MLP([\boldsymbol{h}_i, \boldsymbol{s}_{t-1}])$$

$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

Recap: Self-Attention



Weighted sum of v

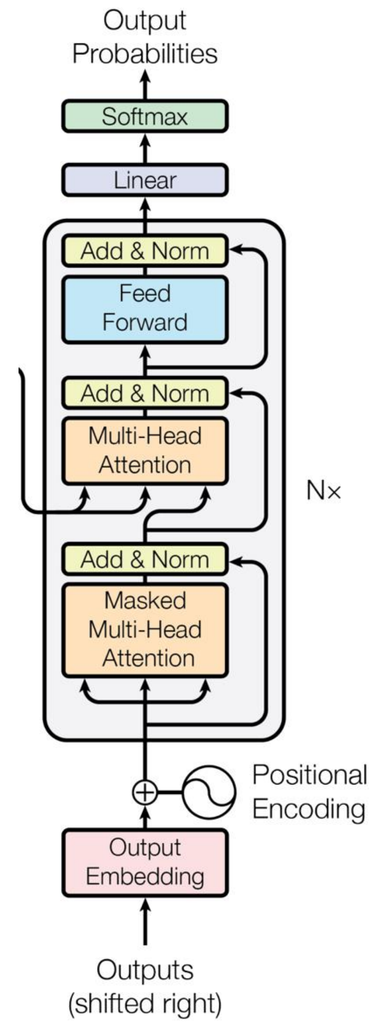
Weights computed from q and k

q, k, v computed from inputs

Transformer Architecture

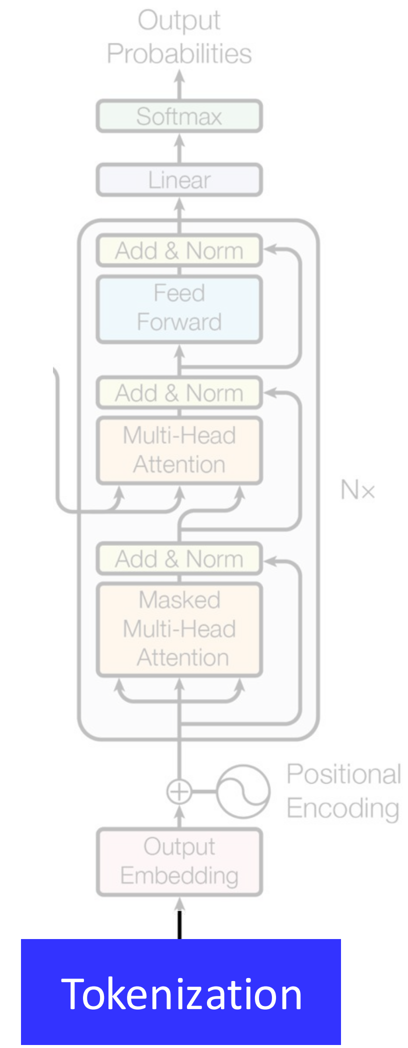
- Word Tokenization
- Word Embedding
- **(Masked) Multi-Head Attention**
- **Position Encoding**
- **Feed-Forward**
- **Add & Norm**
- Output Projection Layer

Transformer Block



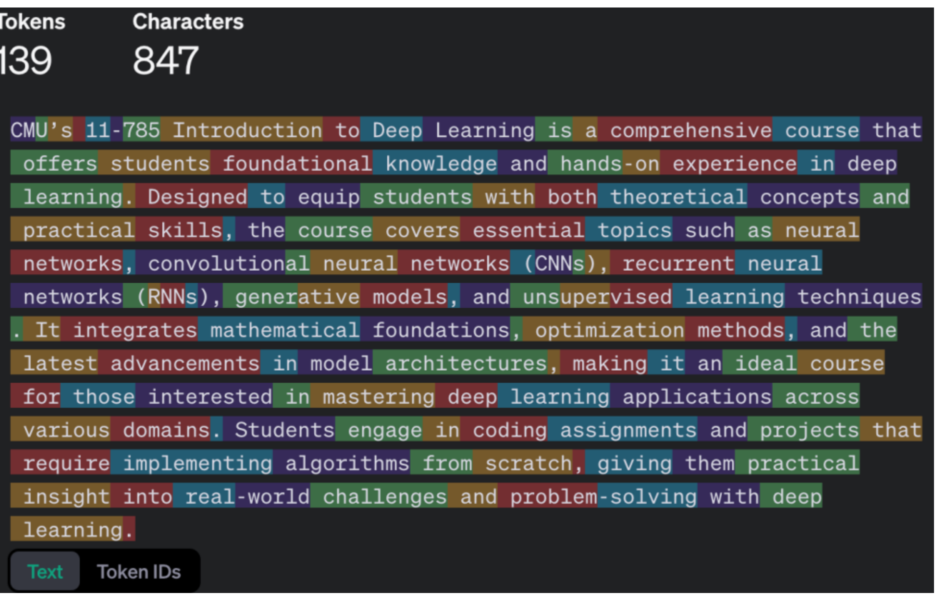
Transformer Architecture

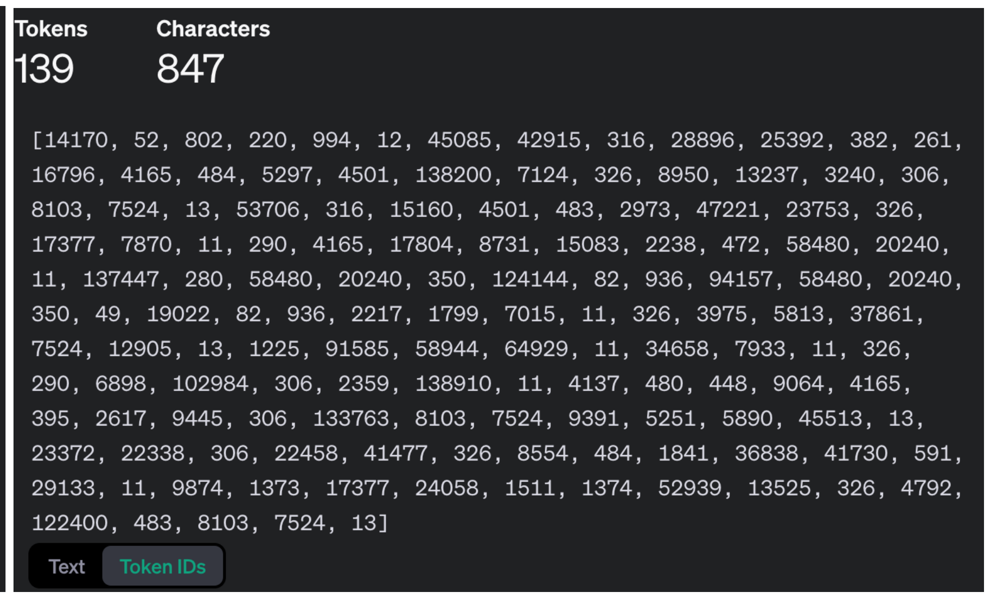
- **Word Tokenization**
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Tokenization

- Maps a word into one/multiple tokens
  - Each token represented as an index/class

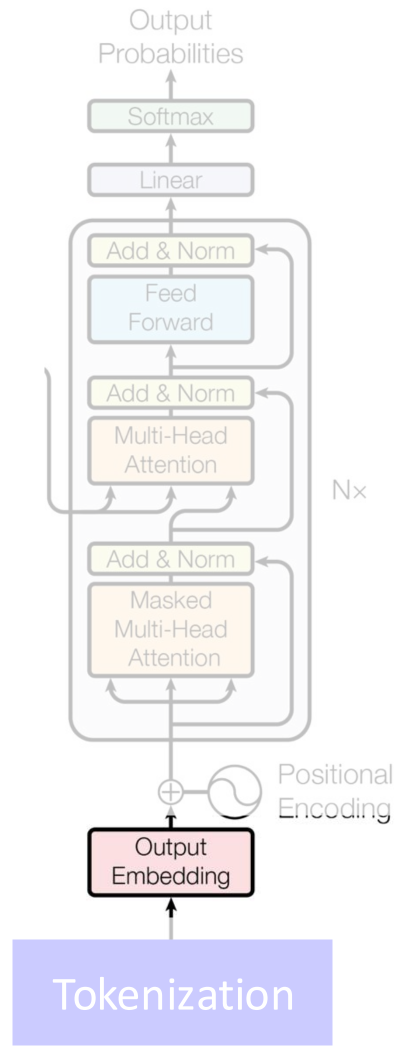




https://platform.openai.com/tokenizer

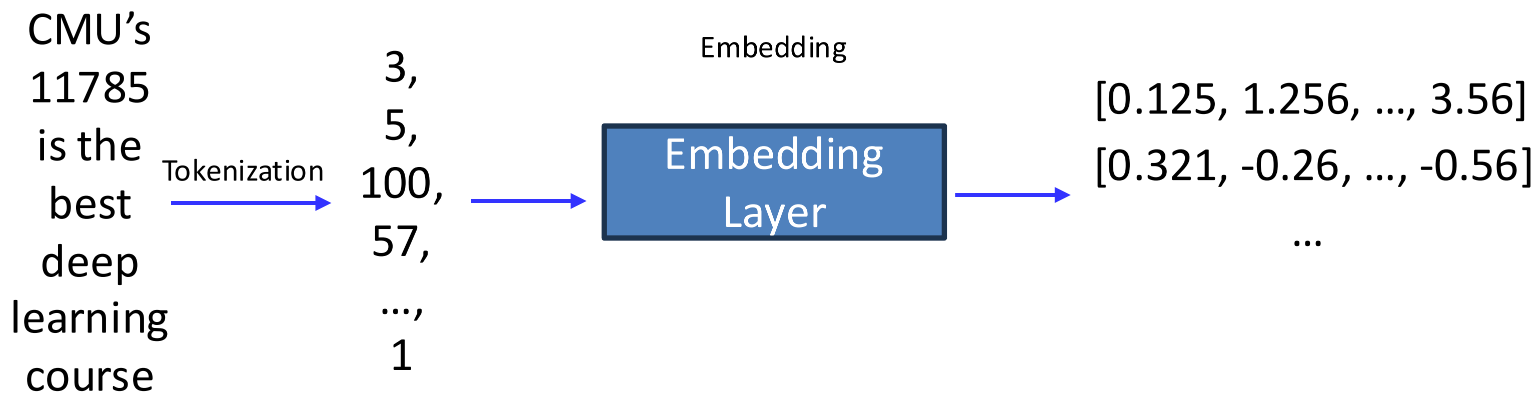
Transformer Architecture

- Word Tokenization
- **Word Embedding**
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer



Embedding

- Represents each discrete token index as continuous token embeddings



Embedding Layer

- The embedding layer is a look-up table that converts token index to continuous vectors

Token Index	Token Embedding
0	[0.235, -1.256, 3.513, …, -0.187]
1	[1.291, -2.012, 0.624, …, -1.291]
2	[0.536, 0.012, -0.024, …, 2.345]
…	…
Vocab Size $\|V\|$	[0.131, 2.102, 0.935, …, -0.125]

- In Pytorch, it is *nn.Embedding*

Embedding Layer is a Linear Layer

- *nn.Embedding* is essentially a linear layer $Y = XW$

One-Hot Vector
Token Index $X \in \mathcal{R}^{N \times |V|}$

Weight Matrix $W \in \mathcal{R}^{|V| \times D}$

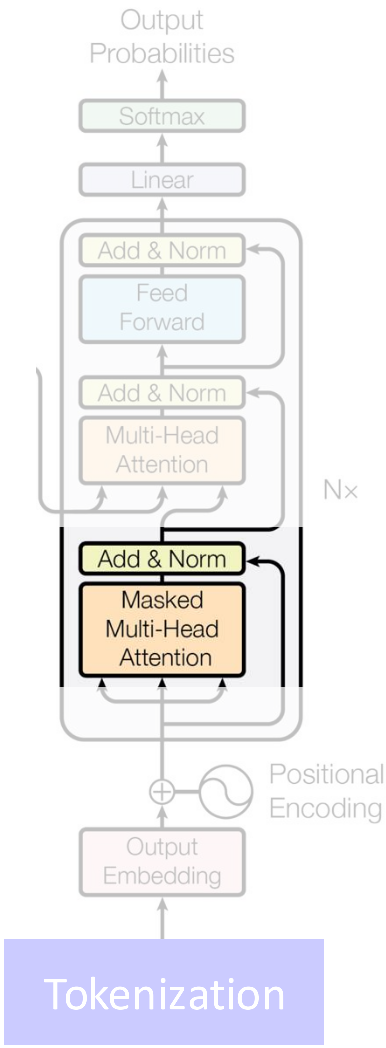
$$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \begin{bmatrix} 0.235 & -1.256 & 3.513 & \dots & -0.187 \\ 1.291 & -2.012 & 0.624 & \dots & -1.291 \\ 0.535 & 0.012 & -0.024 & \dots & 2.345 \\ \dots & \dots & \dots & \dots & \dots \\ 0.131 & 2.102 & 0.935 & \dots & -0.125 \end{bmatrix}$$

$$\begin{bmatrix} 1.291 & -2.012 & 0.624 & \dots & -1.291 \\ 0.535 & 0.012 & -0.024 & \dots & 2.345 \\ 0.131 & 2.102 & 0.935 & \dots & -0.125 \end{bmatrix}$$

Token Embedding $Y \in \mathcal{R}^{N \times D}$

Transformer Architecture

- Word Tokenization
- Word Embedding
- **(Masked) Multi-Head Attention**
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer

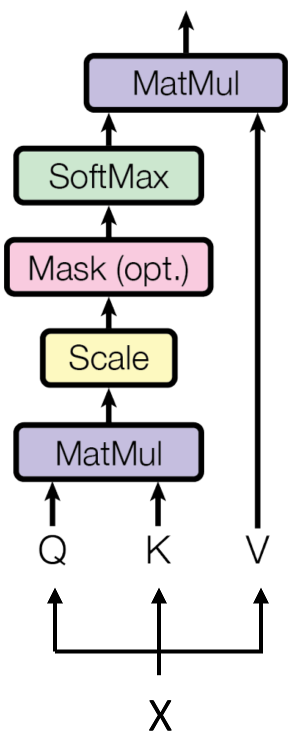


Self-Attention

- Attention Operation

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

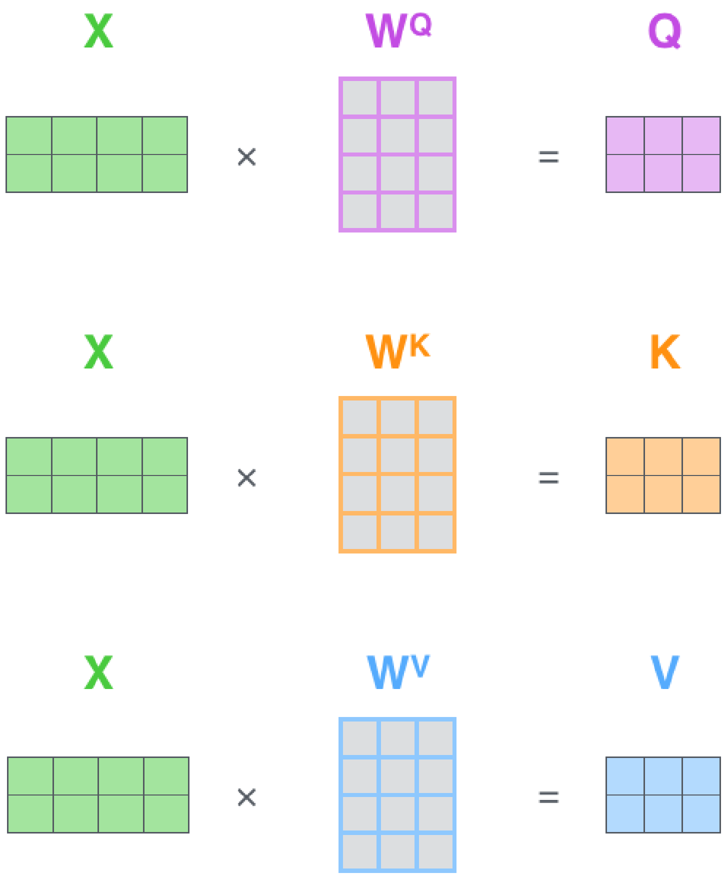
Scaled Dot-Product Attention

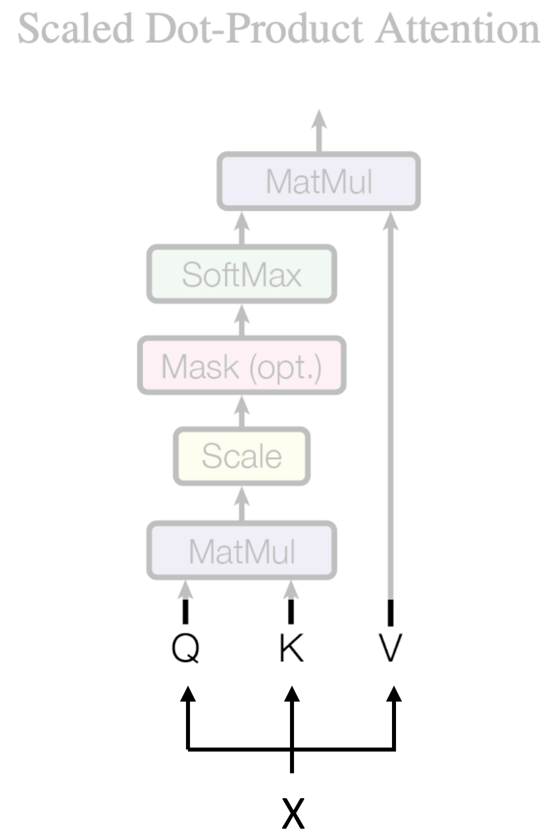


- Query-Key-Value
  - Linear affine from input X itself
- Weighted-sum of V based on similarity/correlation between Q and K
  - Each token's weights sum to one

Self-Attention

- Query-Key-Value from Three Linear Affine of X



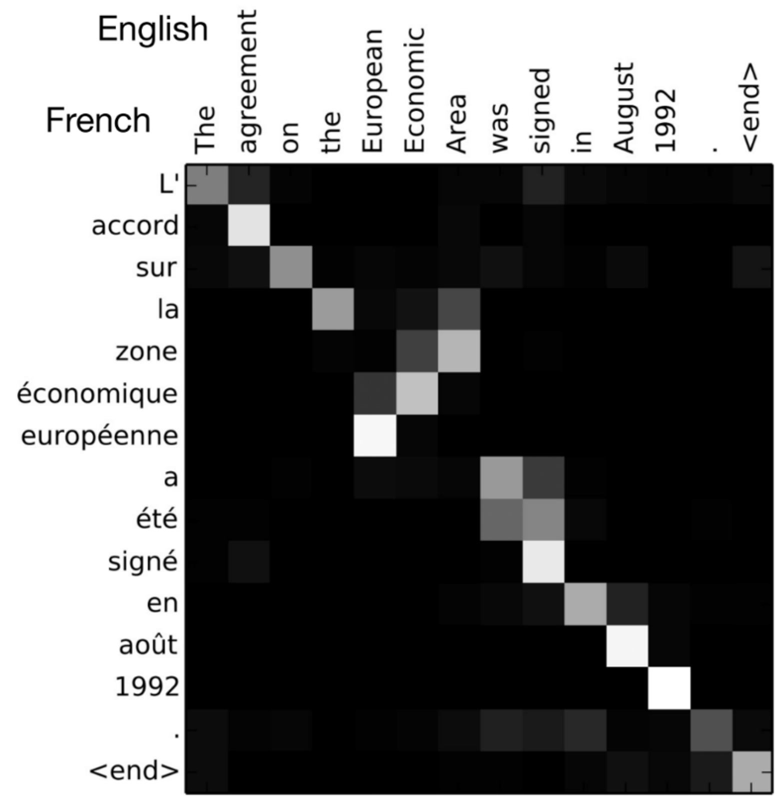


The Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Self-Attention

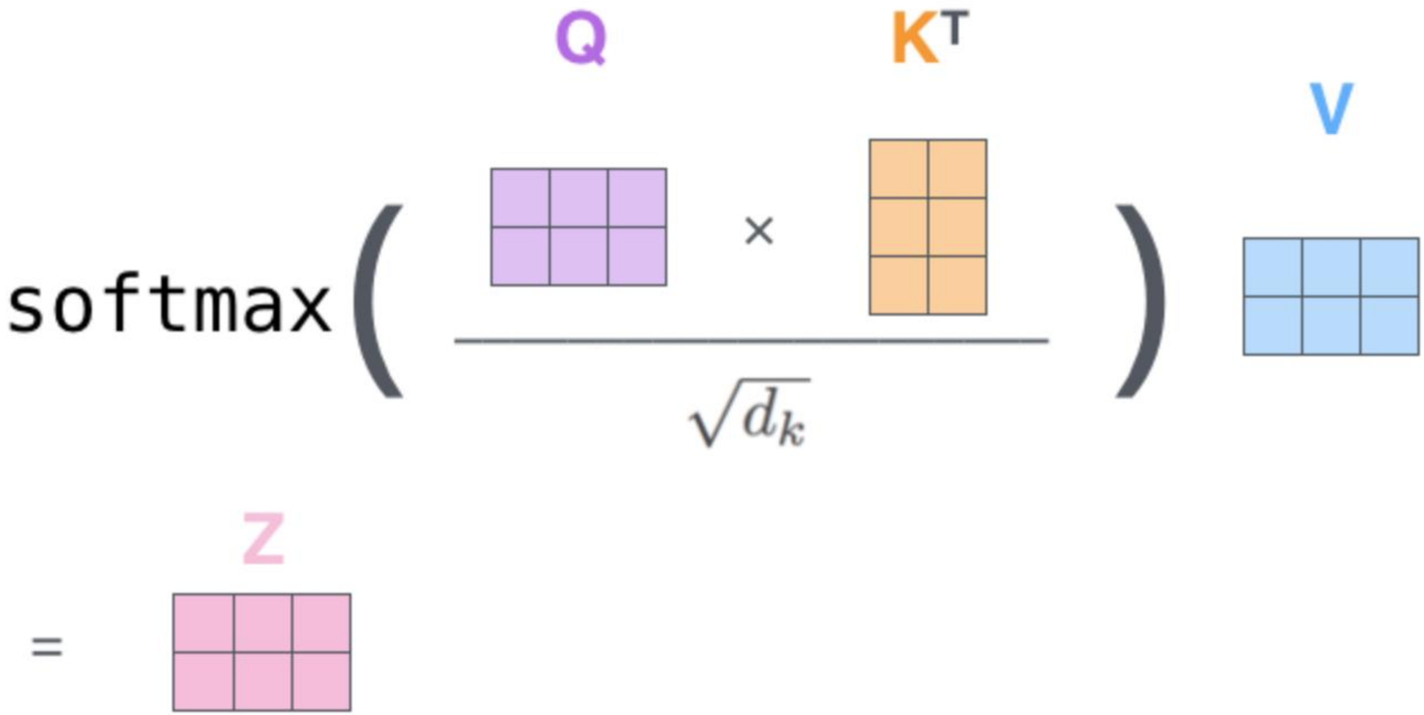
- Attention weights

$$\text{softmax}\left(\frac{Q \times K^{T}}{\sqrt{d_k}}\right)$$



Self-Attention

- Output

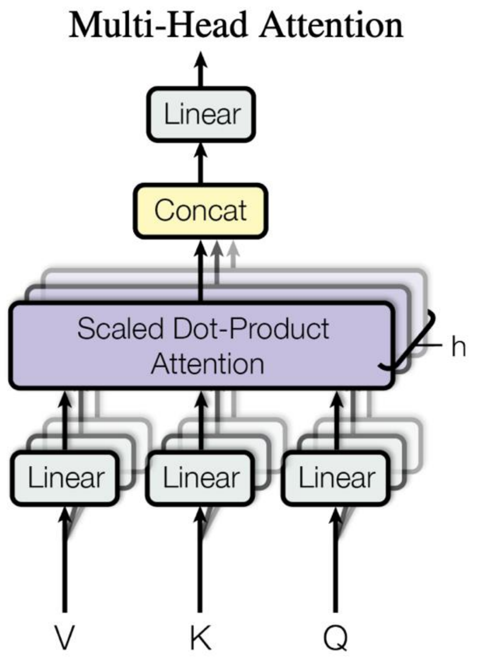


Multi-Head Self-Attention

- Multiple self-attention operations over the channel dimension

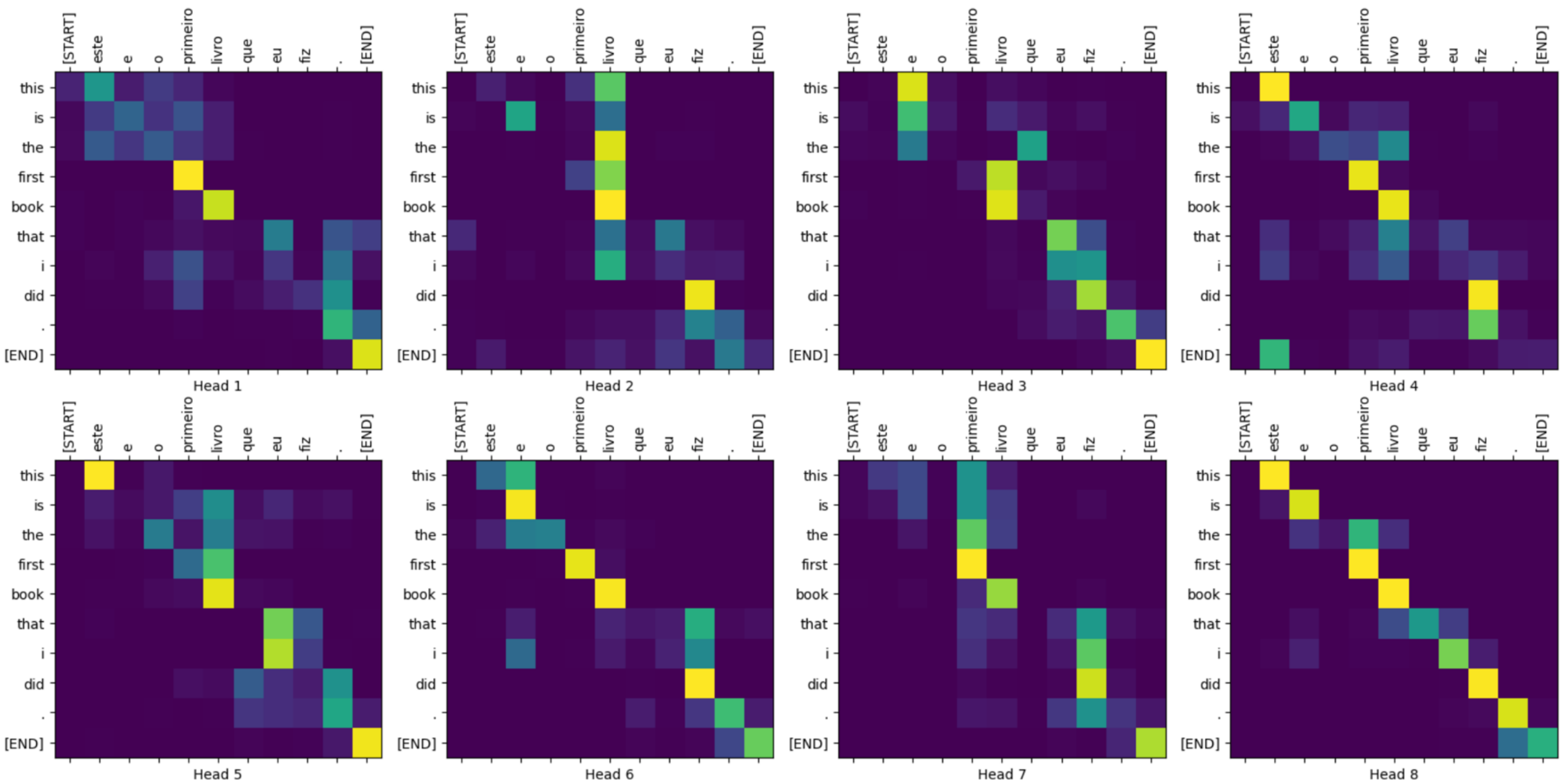
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}\left(\mathrm{head}_1, \ldots, \ \mathrm{head}_{\mathrm{h}}\right)W^{Q}$$
$$\text{where head} = \mathrm{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

- Different attention maps capture different relationships



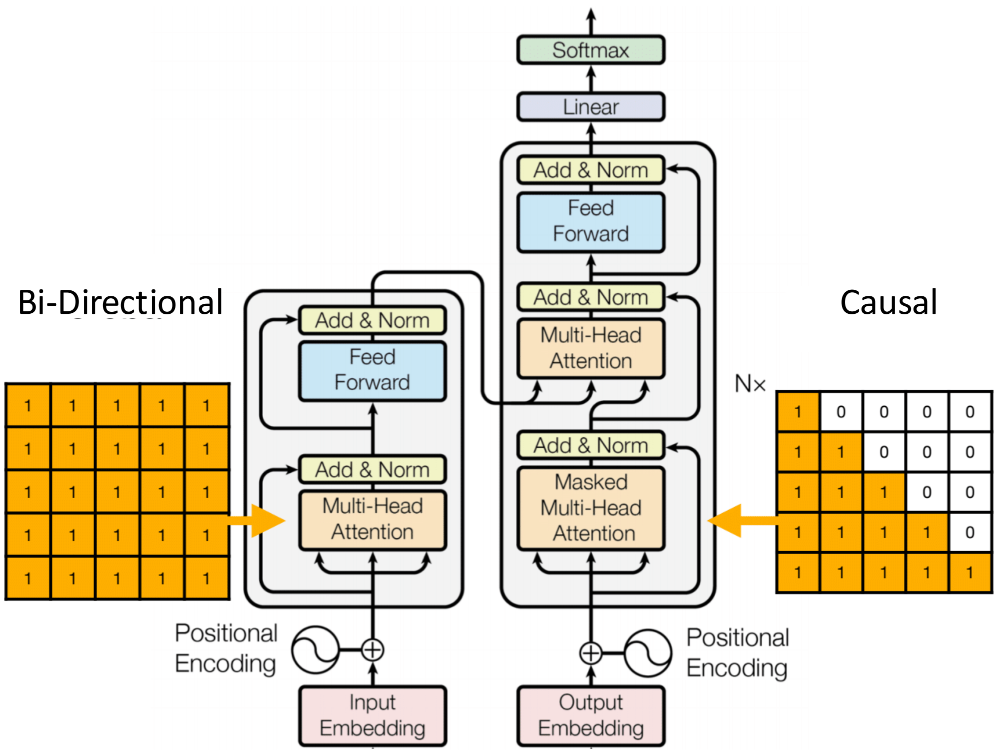
Multi-Head Attention

- Each head captures different semantics

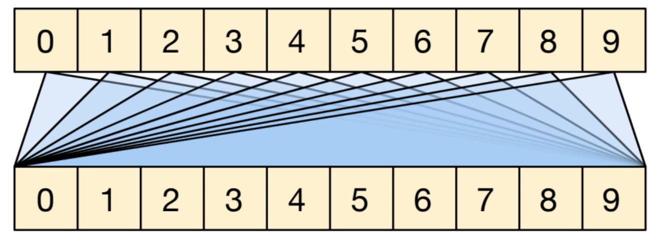


https://www.tensorflow.org/text/tutorials/transformer

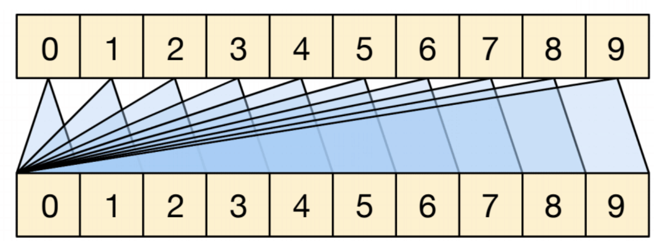
Attention Masking



Bi-Directional

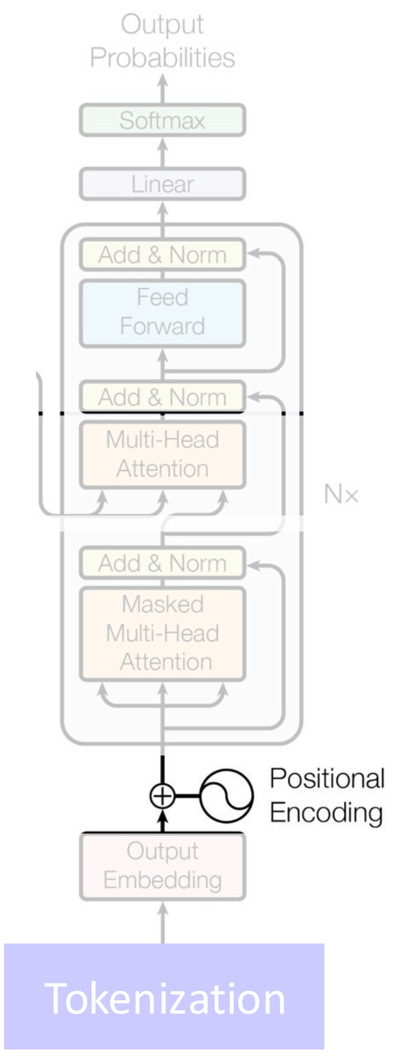


Causal



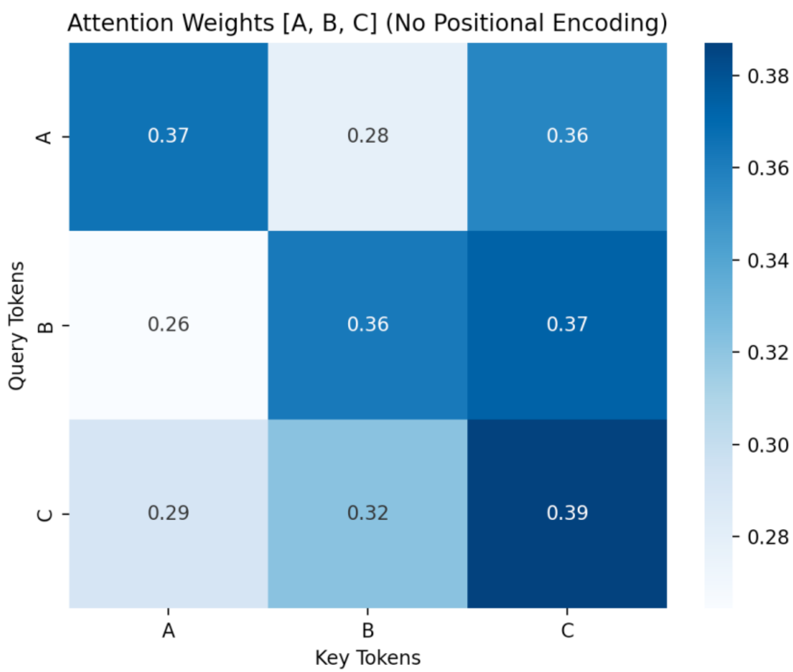
Transformer Architecture

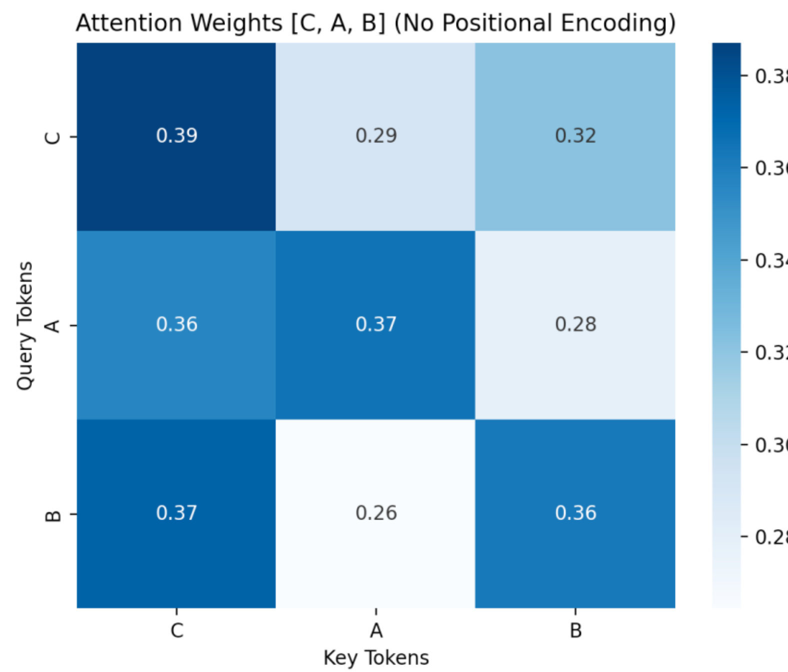
- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- **Position Encoding**
- Feed-Forward
- Add & Norm
- Output Projection Layer



Position Encoding

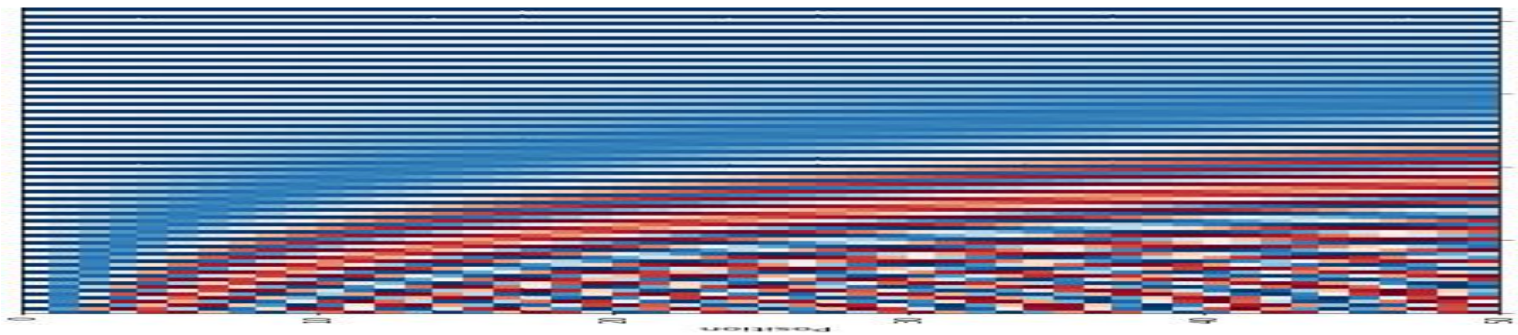
- Why do we need them?
  - Self-attention is permutation-invariant!
- Considering a sequence of
  - [A, B, C] vs. [C, A, B]





Position Encoding

- Captures the abs./relative distance between tokens



$$P_t = \begin{bmatrix} \sin \omega_1 t \\ \cos \omega_1 t \\ \sin \omega_2 t \\ \cos \omega_2 t \\ \vdots \\ \sin \omega_{d/2} t \\ \sin \omega_{d/2} t \end{bmatrix}$$

$$\omega_l = \frac{1}{10000^{2l/d}}$$

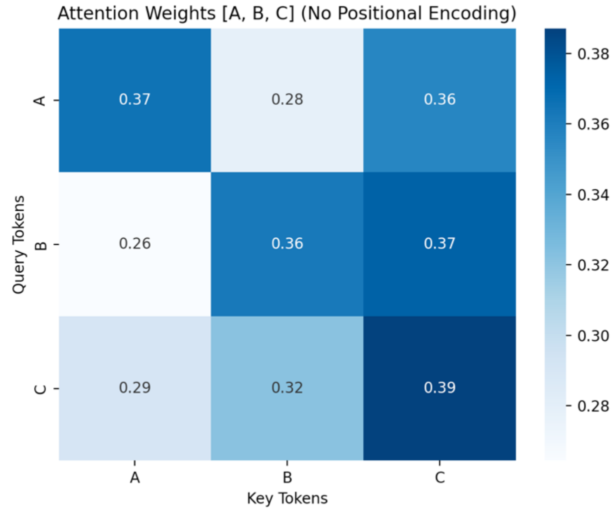
$$P_{t+\tau} = M_\tau P_t$$

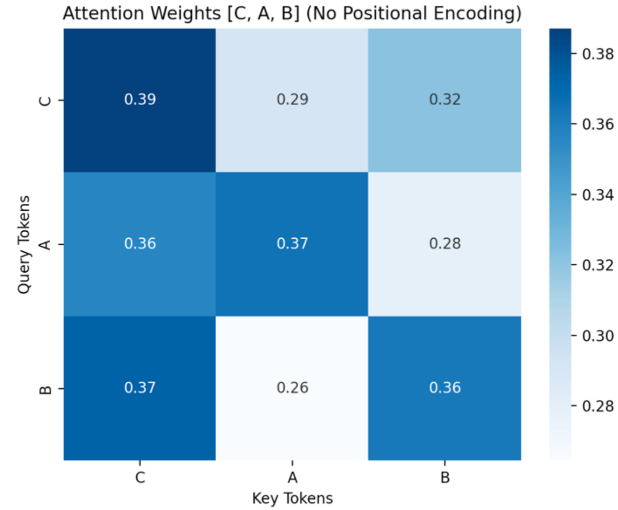
$$M_\tau = diag\left(\begin{bmatrix} \cos \omega_l \tau & \sin \omega_l \tau \\ -\sin \omega_l \tau & \cos \omega_l \tau \end{bmatrix}, l = 1 \ldots d/2\right)$$

  - A vector of sines and cosines of a harmonic series of frequencies
  - Never Repeats

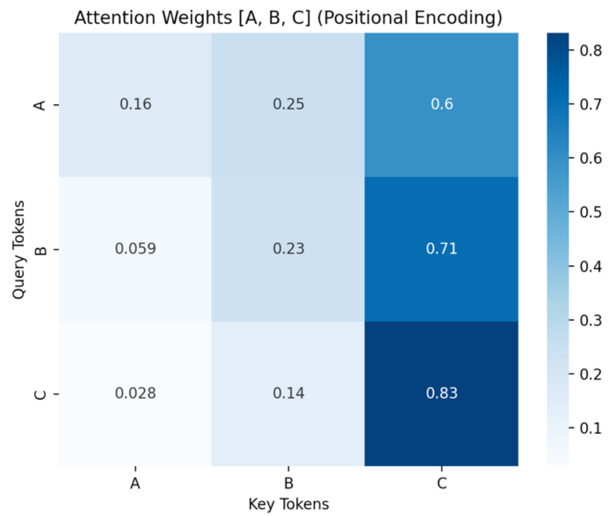
Position Encoding

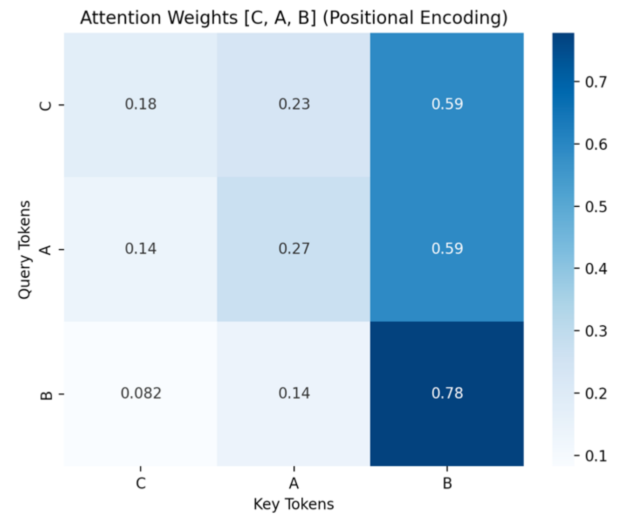
No Position Info.





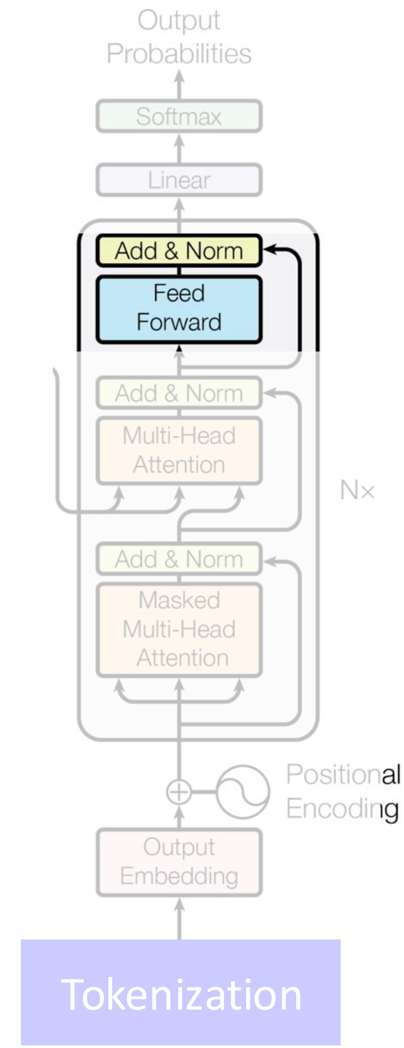
With Position Info.





Transformer Architecture

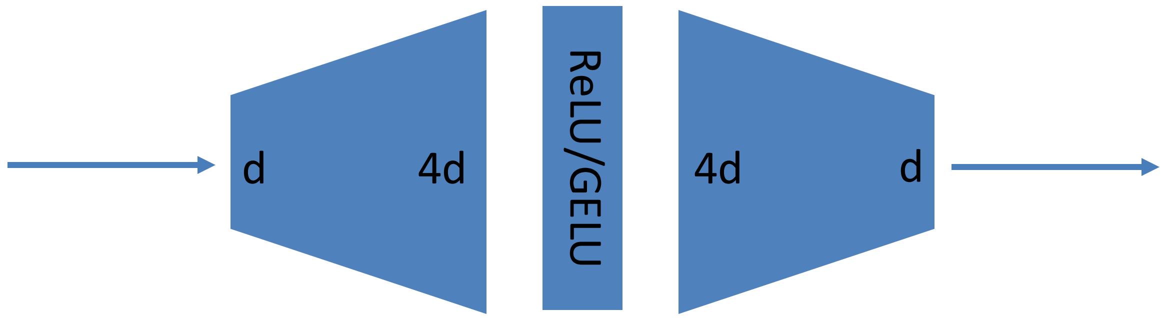
- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- **Feed-Forward**
- Add & Norm
- Position Encoding
- Output Projection Layer



Feed-Forward Block

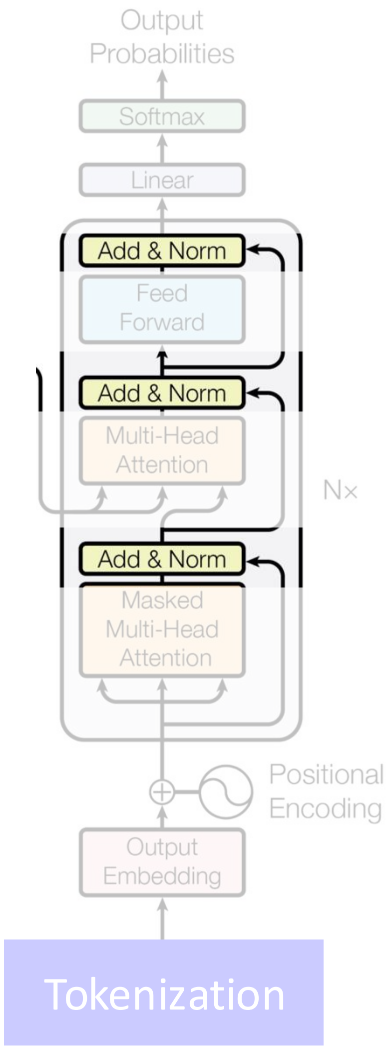
- Just a MLP!

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



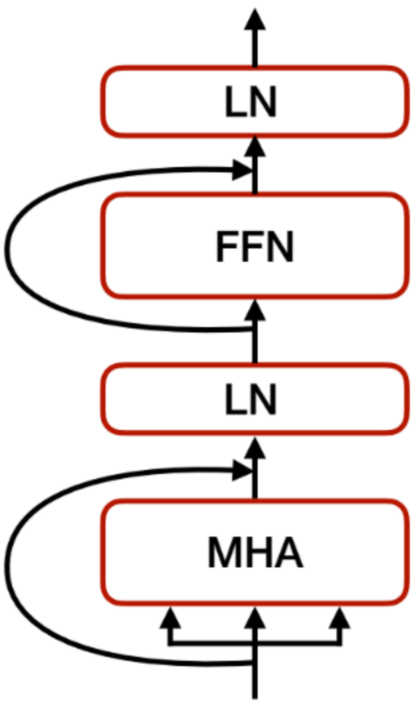
Transformer Architecture

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Feed-Forward
- **Add & Norm**
- Position Encoding
- Output Projection Layer



Residual and Normalization

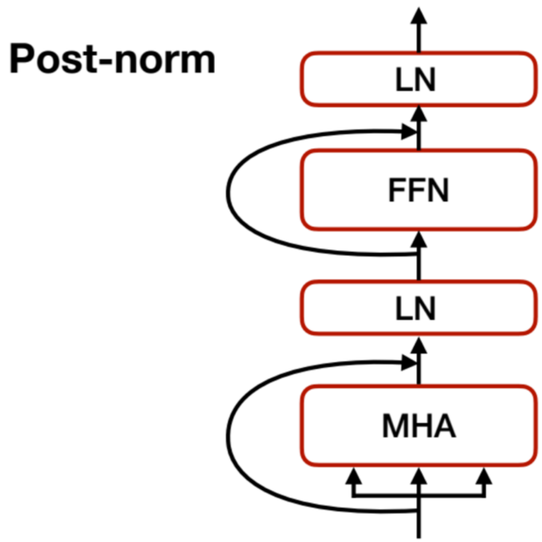
- Each layer in Transformer has:
  - A residual connection
  - A normalization layer
- Layer Norm. normalize each token by its embedding size dimension
  - For more stable training



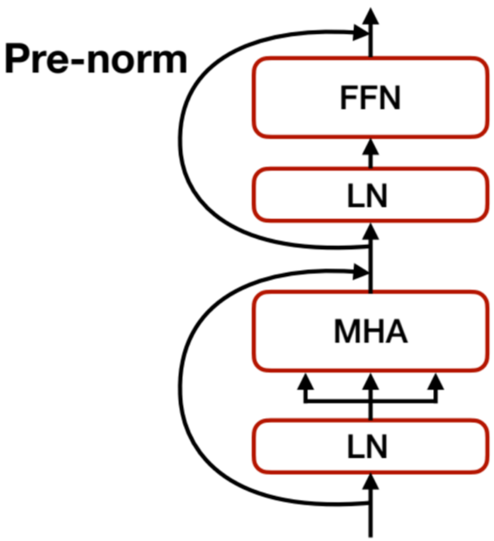
Position of Normalization

- Post-Norm vs Pre-Norm

$$\boldsymbol{x}_{t+1} = \mathrm{Norm}\left(\boldsymbol{x}_t + F_t\left(\boldsymbol{x}_t\right)\right)$$



$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + F_t\left(\mathrm{Norm}\left(\boldsymbol{x}_t\right)\right)$$

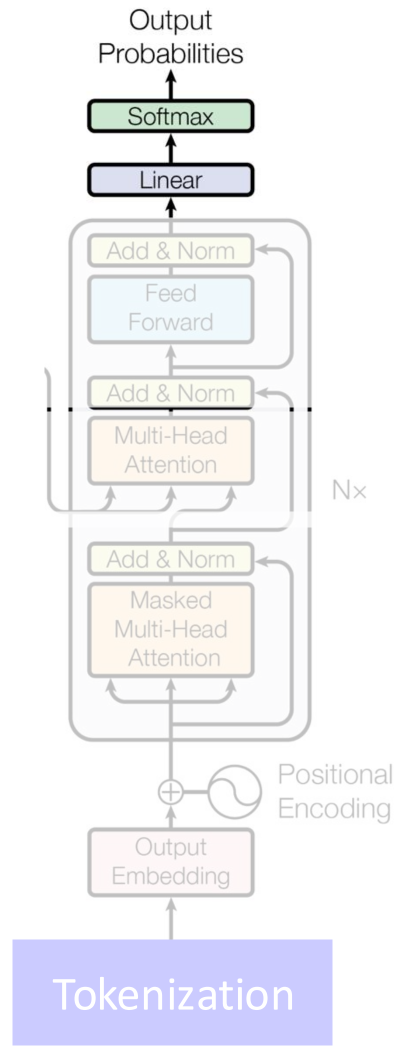


- Pre-Norm is easier and more stable to train
- Post-Norm tends to present better performance if properly trained

Credit to: Han Song. TinyML. Lecture 12 Transformers.

Transformer Architecture

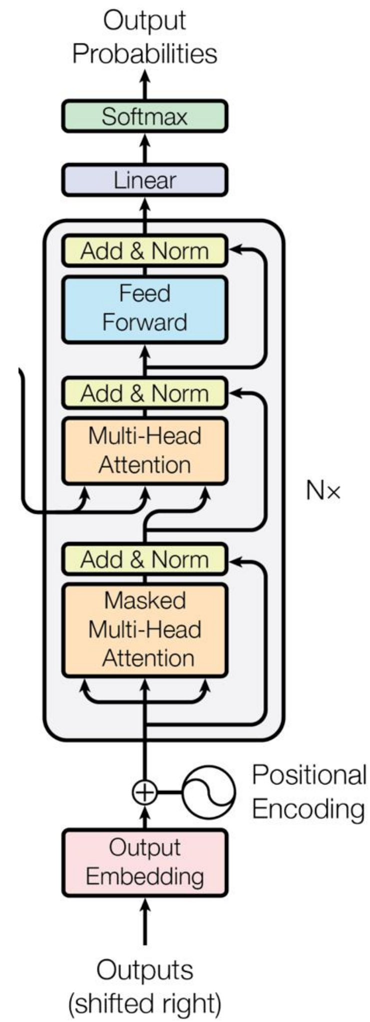
- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Feed-Forward
- Add & Norm
- Position Encoding
- **Output Projection Layer**
  - **Just a linear layer**



Putting Them Together - Transformer

- Word Tokenization
- Word Embedding
- (Masked) Multi-Head Attention
- Position Encoding
- Feed-Forward
- Add & Norm
- Output Projection Layer

Transformer Block



Poll

Transformer in NLP

NLP Tasks with Transformer

- Question Answering.
- Machine Translation.
- Summarization.
- Code Generation.
- Text Completion.
- Sentiment Analysis.
- Dialogue Generation and Conversational AI.
- Semantic Search.
- Text Anonymization.
- …….

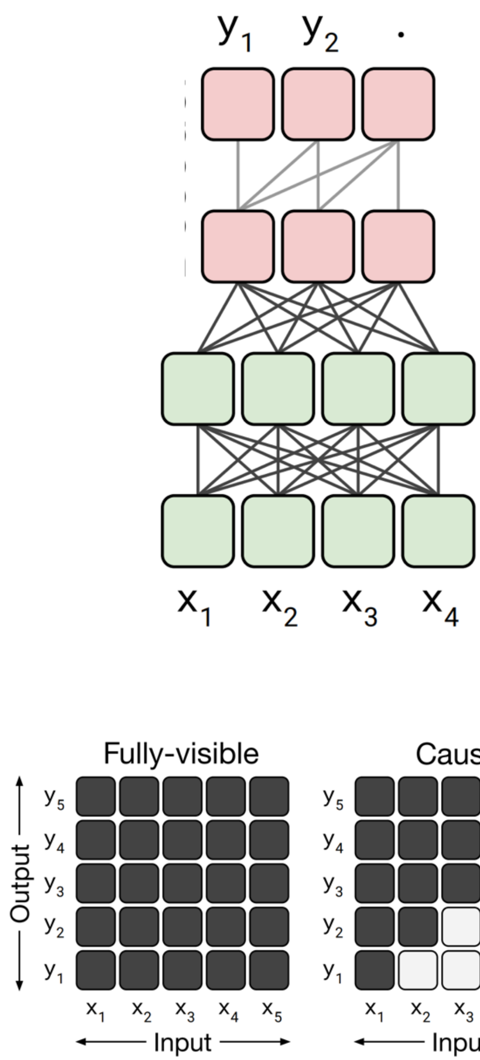
Overview

- Architecture
  - Encoder-Decoder
  - Encoder-Only
  - Decoder-Only
- Position Encoding
  - Relative Position Encoding
  - Rotary Position Encoding
- Efficient Attention Mechanism

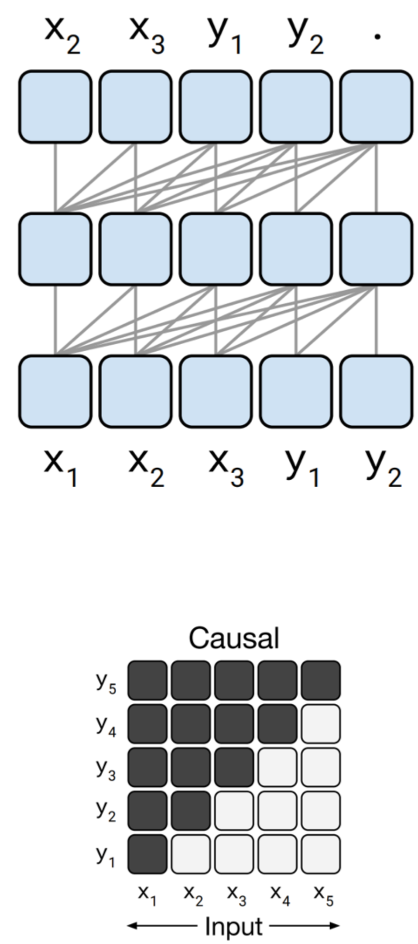
Overview

- Architecture
  - Encoder-Decoder
  - Encoder-Only
  - Decoder-Only
- Position Encoding
  - Relative Position Encoding
  - Rotary Position Encoding
- Efficient Attention Mechanism

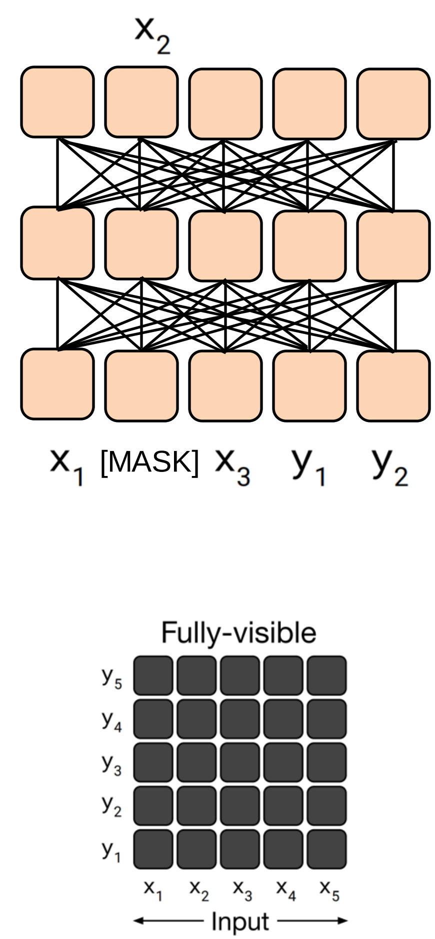
Encoder-Decoder



Decoder-Only

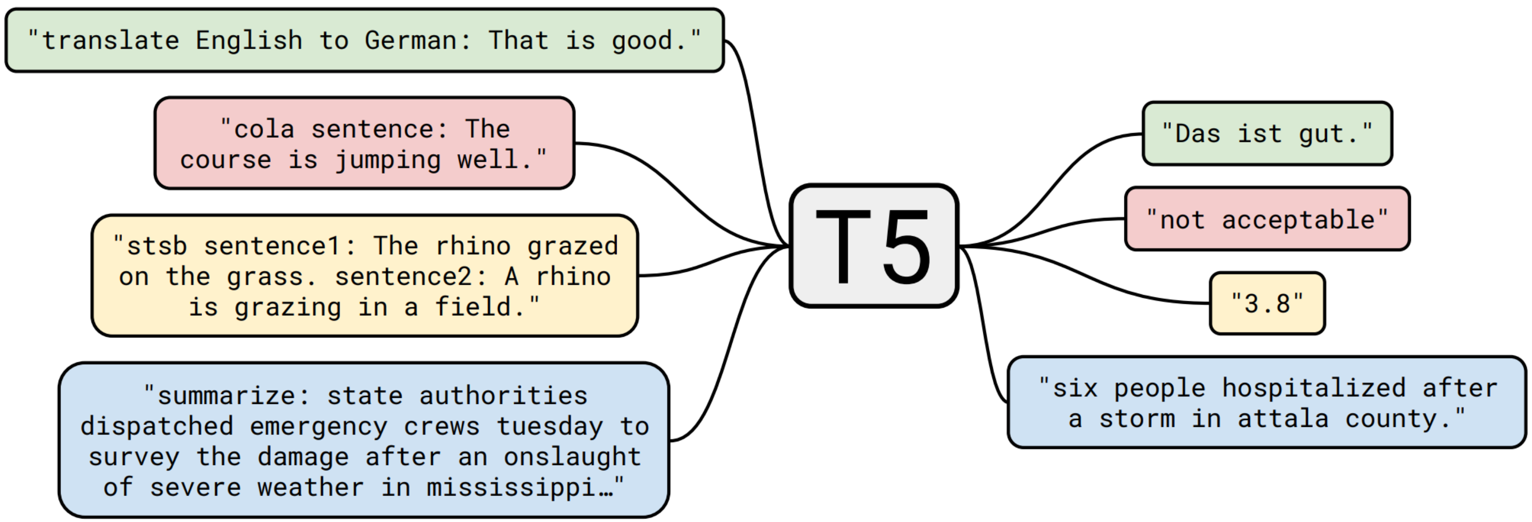


Encoder-Only



Encoder-Decoder - T5

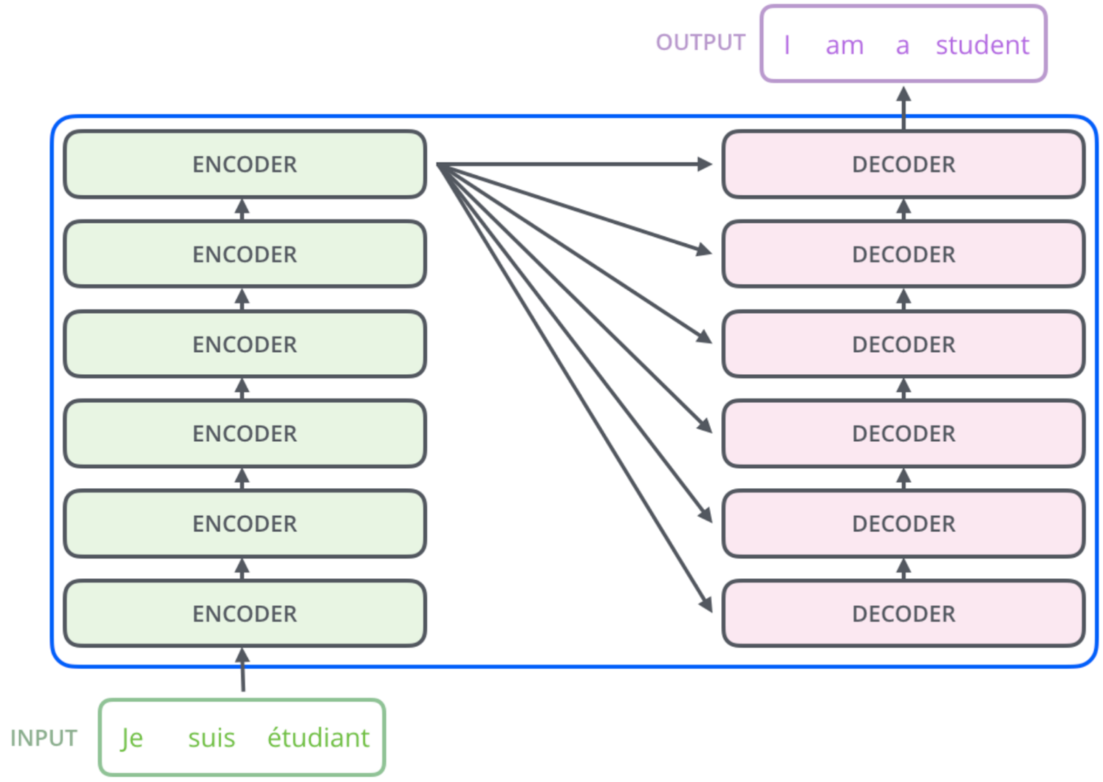
- Encoder-Decoder architecture as in the original transformer paper
- A text-to-text model on various NLP tasks



Raffel et. al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. 2019

Encoder-Decoder - T5

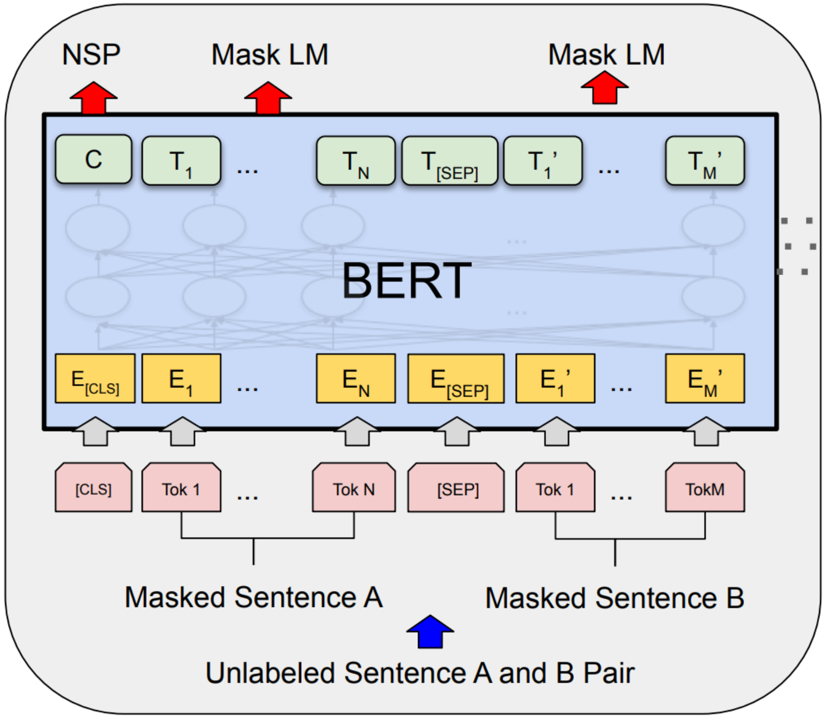
- The prompt is fed into encoder, and the decoder generates answer



Illustrated Transformer. https://jalammar.github.io/illustrated-transformer/

Encoder-Only - BERT

- Bidirectional Encoder Representations from Transformers (BERT)
  - Encoder-only arch.

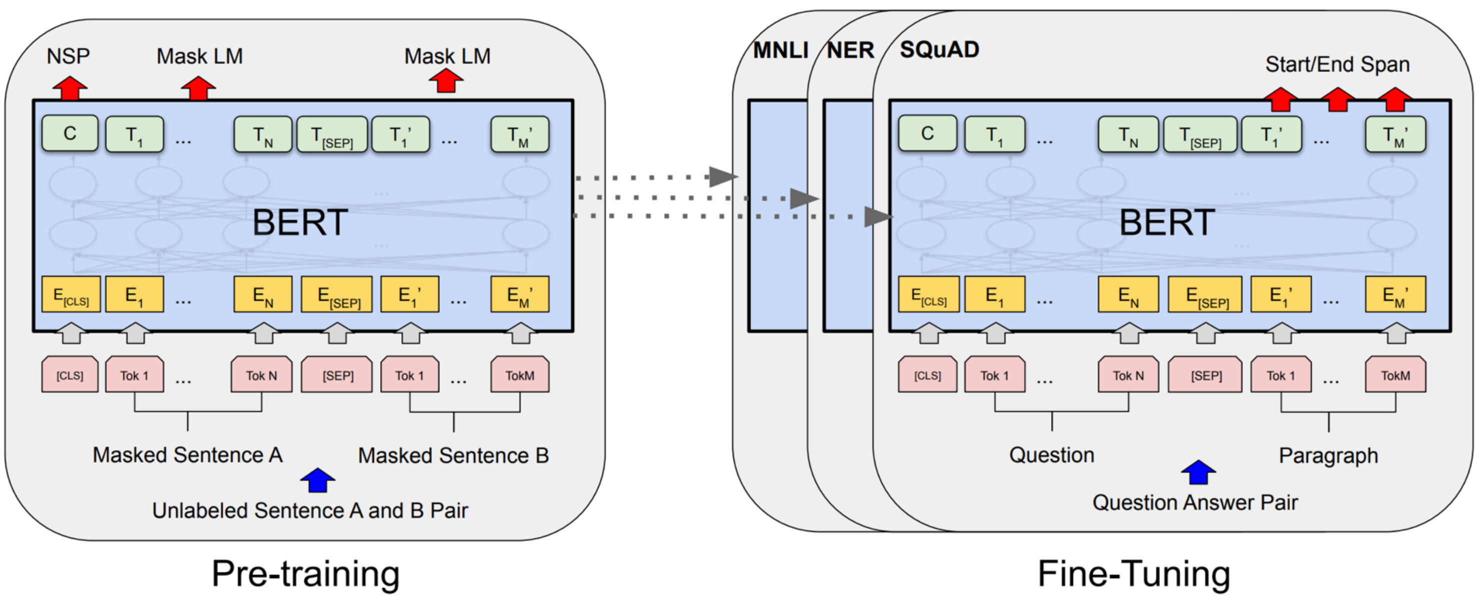


- Trained with
  - Mask token prediction
  - Next sentence prediction

Pre-training and then Fine-Tuning

Pre-training on a proxy task
- Masked token prediction
- Next sentence prediction

Fine-tuning on specific downstream tasks
- Machine translation
- Question answering

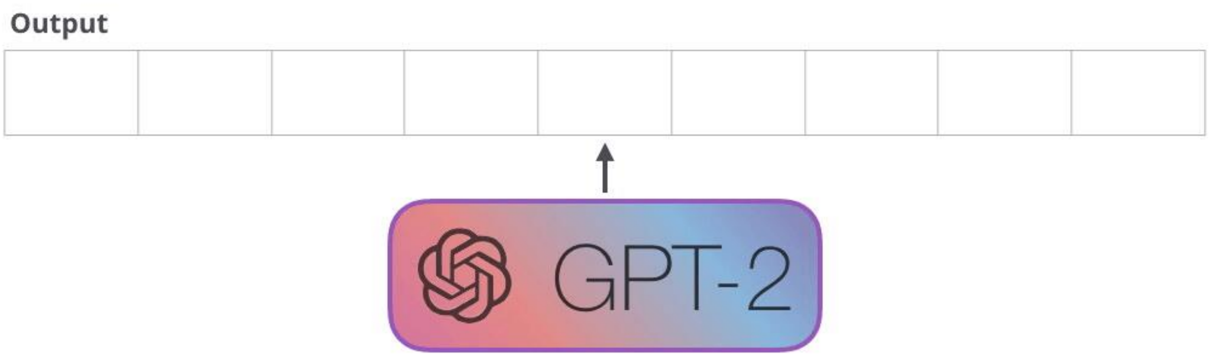


Jacob et. al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019

Decoder-Only - GPT

- Generative Pre-training (GPT)
  - Decoder-only
- Trained with next token prediction
  - A language model!

$$L_1(\mathcal{U}) = \sum_i \log P(u_i \mid u_{i-k}, \ldots, u_{i-1}; \Theta)$$



Large Language Model

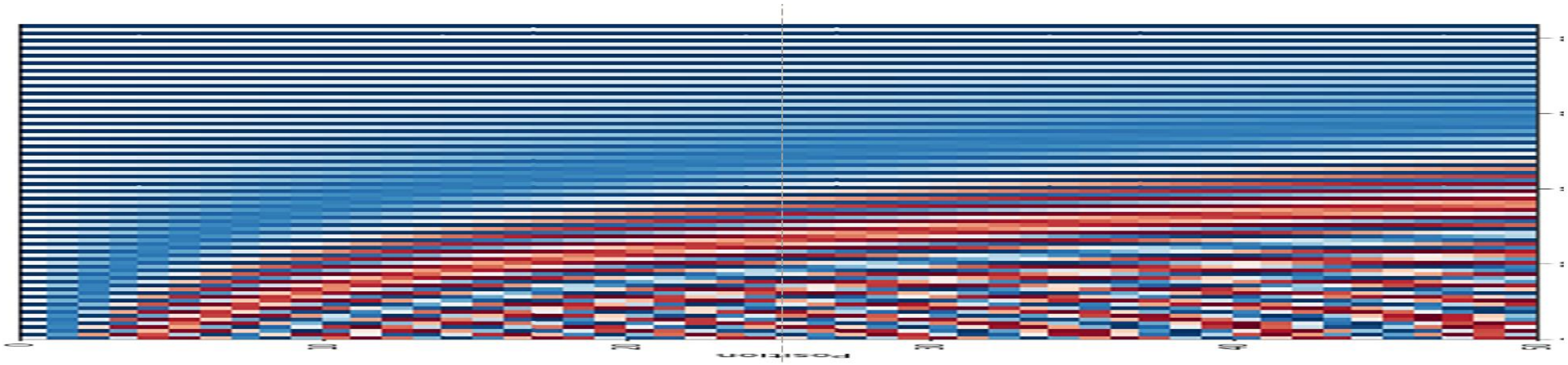
- GPT-2
  - Pre-training and fine-tuning on specific tasks
- GPT-3
  - zero-shot capability
  - in-context learning
  - ChatGPT!
- GPT-4

Overview

- Architecture
  - Encoder-Decoder
  - Encoder-Only
  - Decoder-Only
- Position Encoding
  - Relative Position Encoding
  - Rotary Position Encoding
- Efficient Attention Mechanism

Absolute Position Encoding

- Absolute position embedding fuses the position information into input embeddings

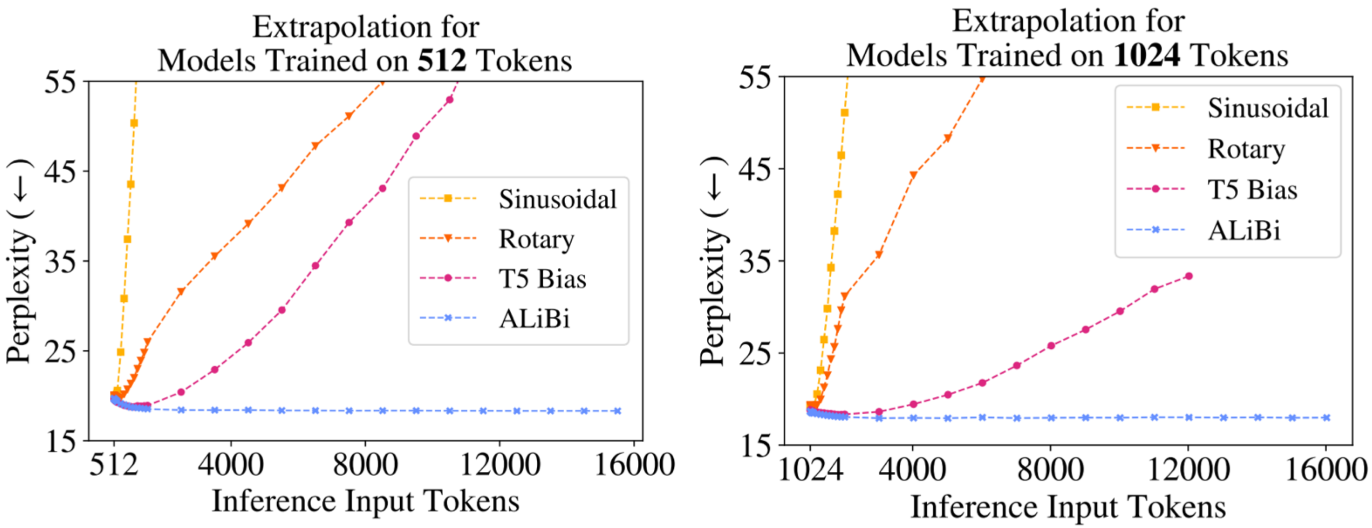


- Fixed length! Not generalize to longer input sequence

Relative Position Encoding

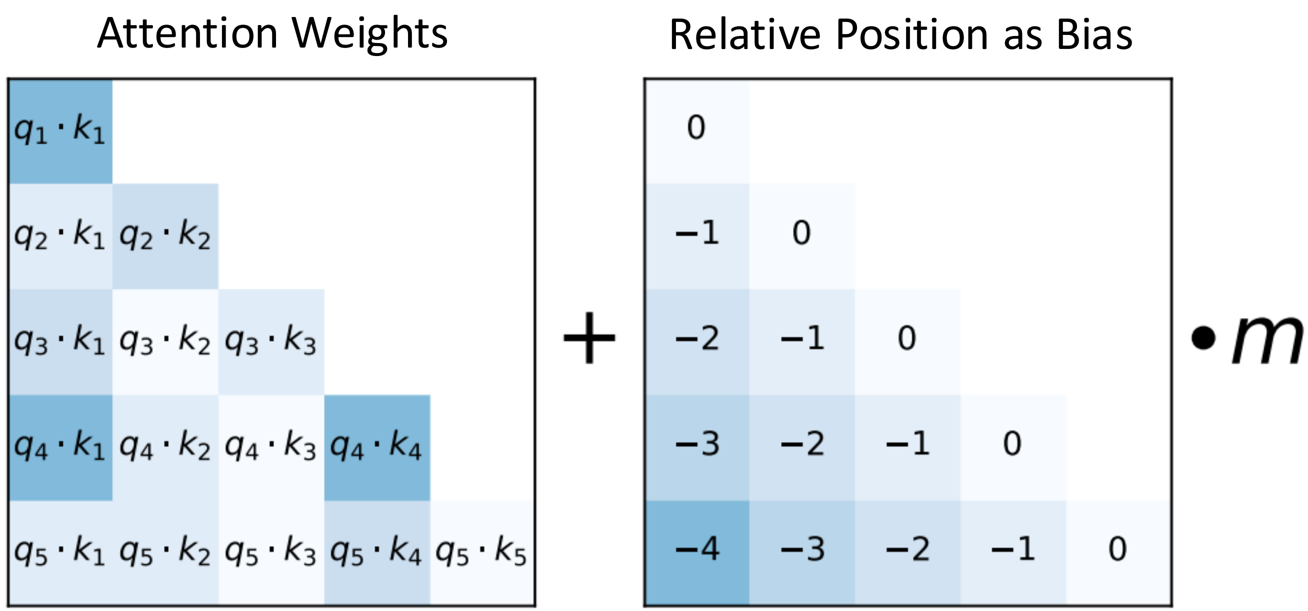
- Relative position embedding fuses position information into attention matrices

- Attention with linear bias
  - Input length extrapolation!



Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. 2021.

Relative Position Encoding

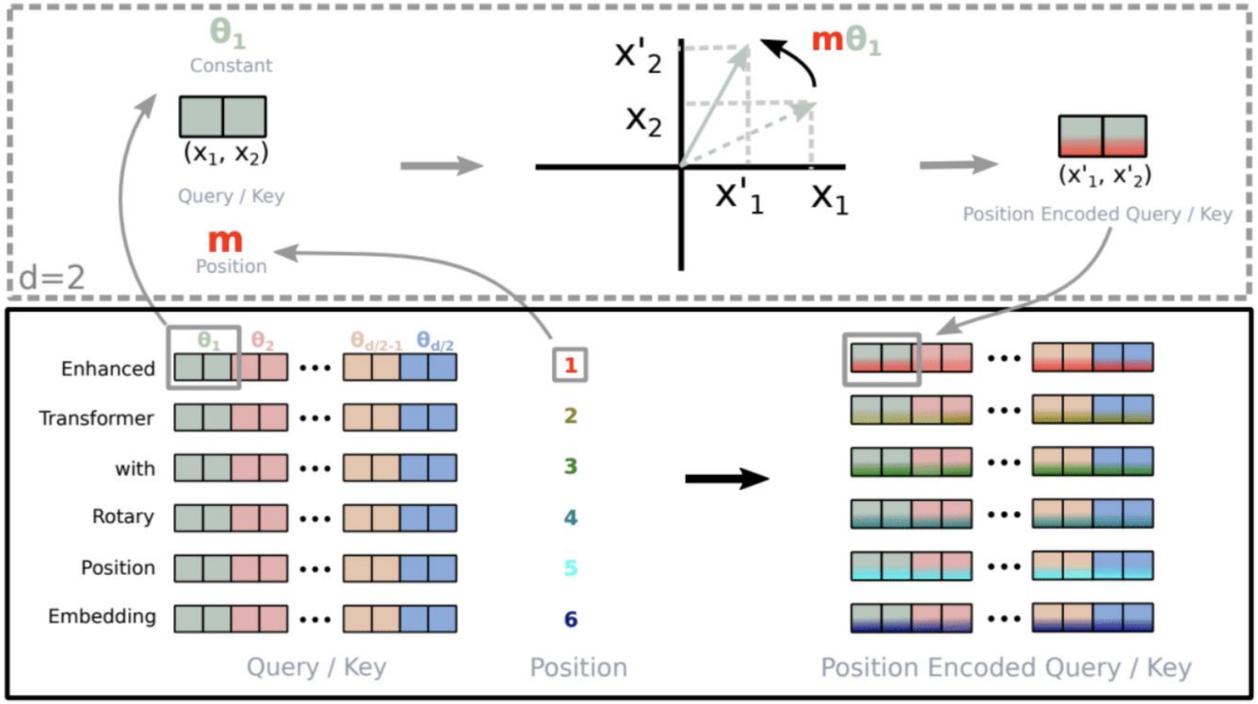


- Relative distance as offset added to attention matrix
- Absolute position embedding not needed

Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. 2021.

Rotary Position Encoding

- Used in Large Language Models such as LLAMA
- Rotate the embedding in 2D space



Su et al. RoFormer: Enhanced Transformer with Rotary Position Embedding. 2021.

Rotary Position Encoding

- General form

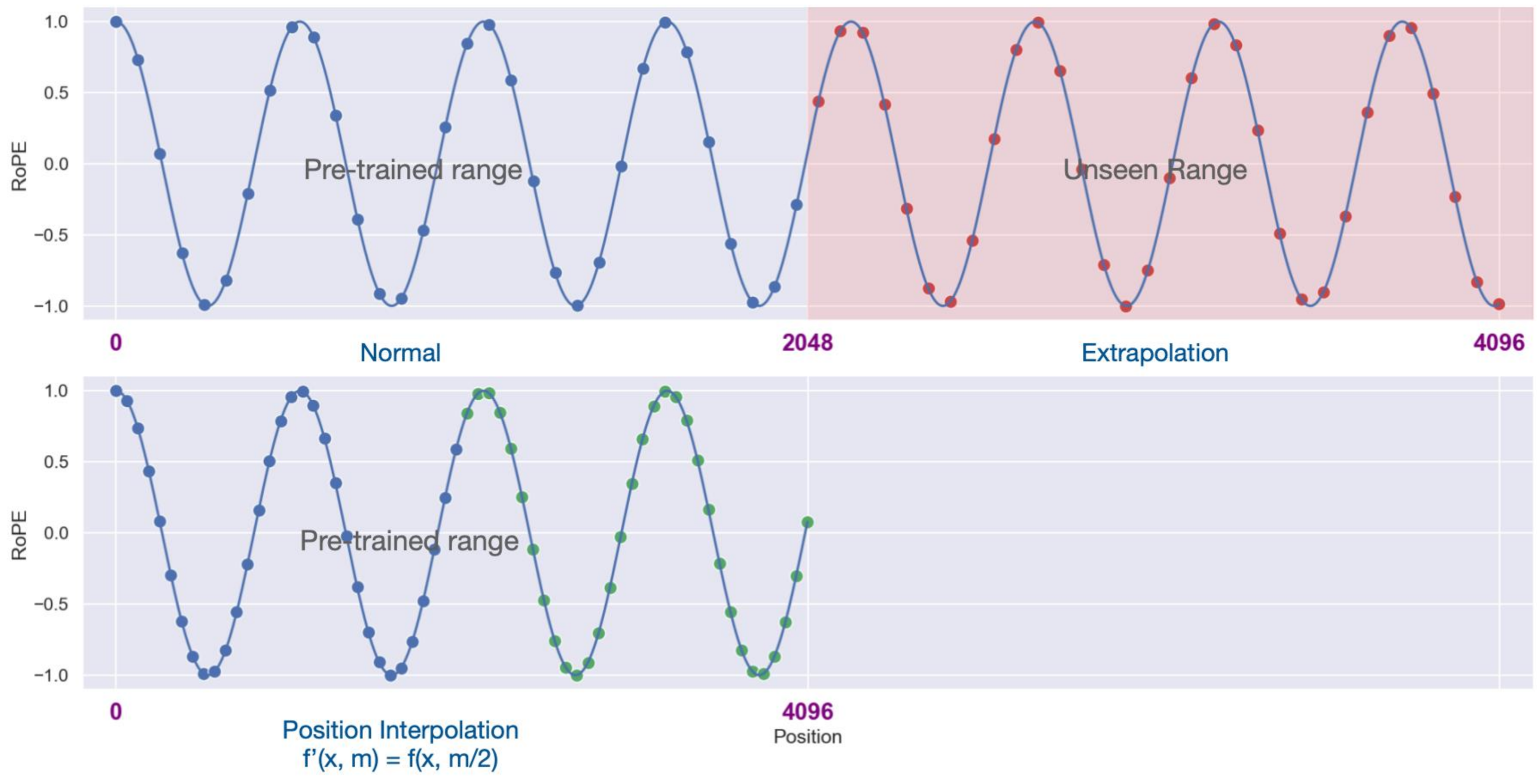
$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m$$

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{\boxed{d/2}} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

Su et al. RoFormer: Enhanced Transformer with Rotary Position Embedding. 2021.

Rotary Position Encoding

- Allows extension of the context window



Chen et al. Extending Context Window of Large Language Models via Positional Interpolation. 2023.

Overview

- Architecture
  - Encoder-Decoder
  - Encoder-Only
  - Decoder-Only
- Position Encoding
  - Relative Position Encoding
  - Rotary Position Encoding
- Efficient Attention Mechanism

Quadratic Complexity

- Self-attention has quadratic complexity to input length

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  – $O\left(L^2 d\right)$ FLOPS

- Many attempts for reducing the quadratic complexity to linear
  – Linear Attention
  – Other Variants

Linear Attention

- Modification on Softmax

$$\text{Softmax}\left(QK^T\right)V = \frac{\exp\left(QK^T\right)}{\sum_{i=1}^{L}\exp\left(QK_i^T\right)}V \longrightarrow \frac{\text{sim}(Q,K)}{\sum_{i=1}^{L}\text{sim}\left(Q,K_i\right)}V$$

- Kernel function

$$\text{sim}(Q,K) = \phi(Q)\cdot\phi(K) = \phi(Q)\phi(K)^T$$

- Linear form of attention

$$\frac{\boxed{\phi(Q)\phi(K)^T}}{\sum_{i=1}^{L}\phi(Q)\phi(K_i)^T}V = \frac{\phi(Q)\boxed{\left(\phi(K)^T V\right)}}{\phi(Q)\sum_{i=1}^{L}\phi(K_i)^T}$$

$O\left(L^2\right)$ $O\left(d'd\right)$

Katharopoulos et al. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. 2020.

Poll

Transformer in Computer Vision

CV Tasks with Transformer

- Classification
- Segmentation
- Detection
- Depth Prediction
- 3D Reconstruction
- Image/Video Generation
- …

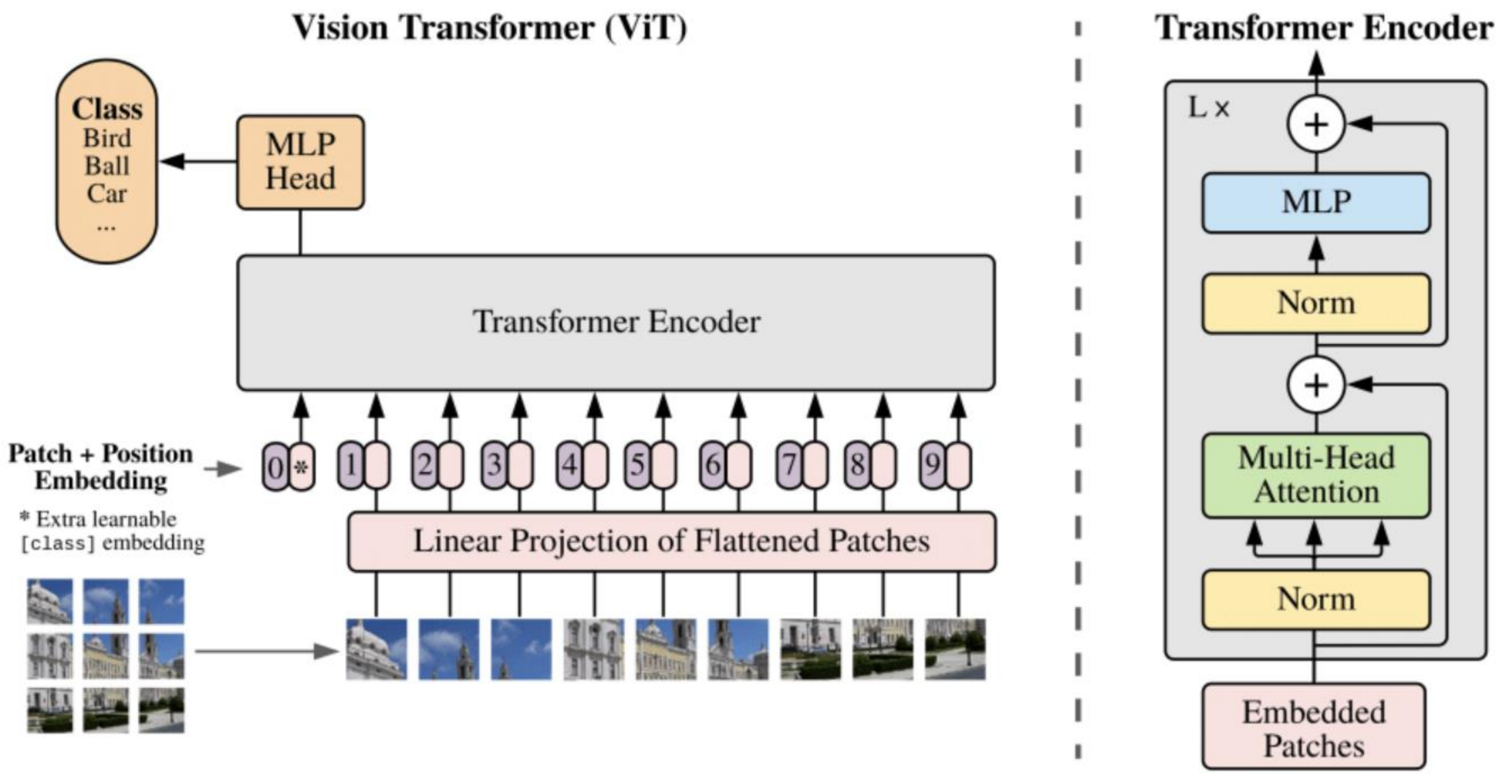
Overview

- Vision Transformer Architecture
- Efficient ViT
- Connection with Convolution
- Transformer Architectures in Vision

Overview

- Vision Transformer Architecture
- Efficient ViT (Training and Modeling)
- Connection with Convolution
- Transformer Architectures in Vision

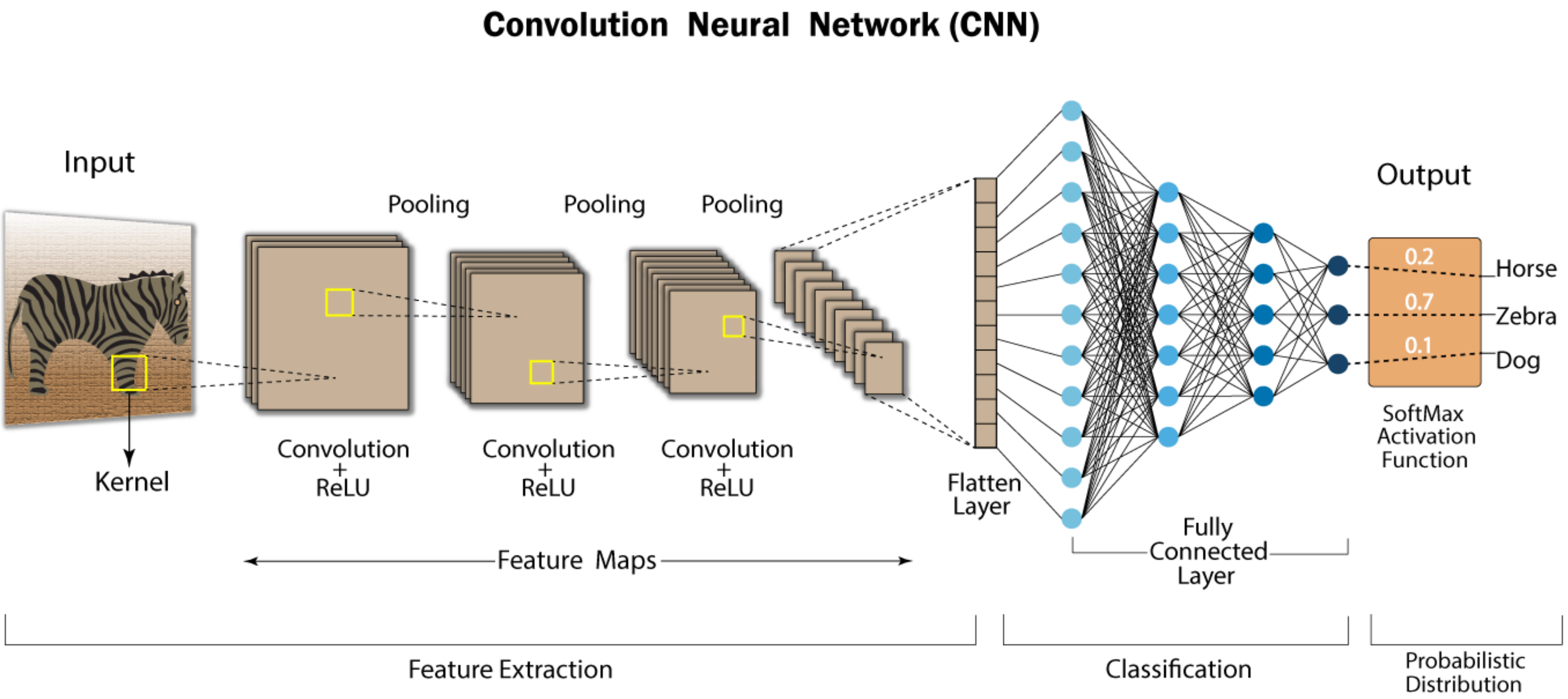
Vision Transformer (ViT)



- Transformer architecture can also be used for images
- How do we process an image into tokens?

Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

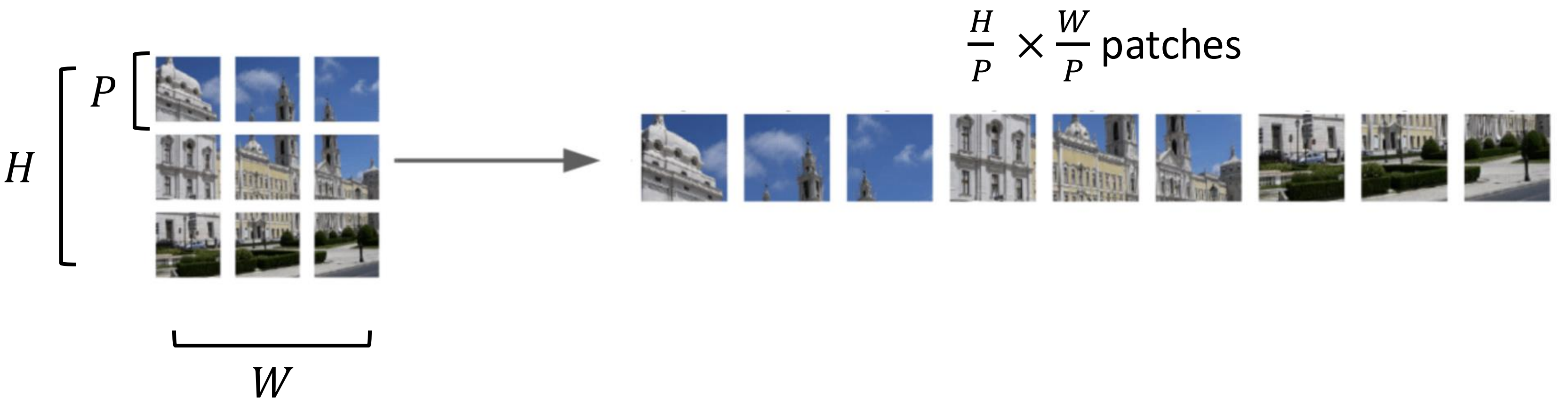
CNN



- Naturally fits to 2D images

ViT

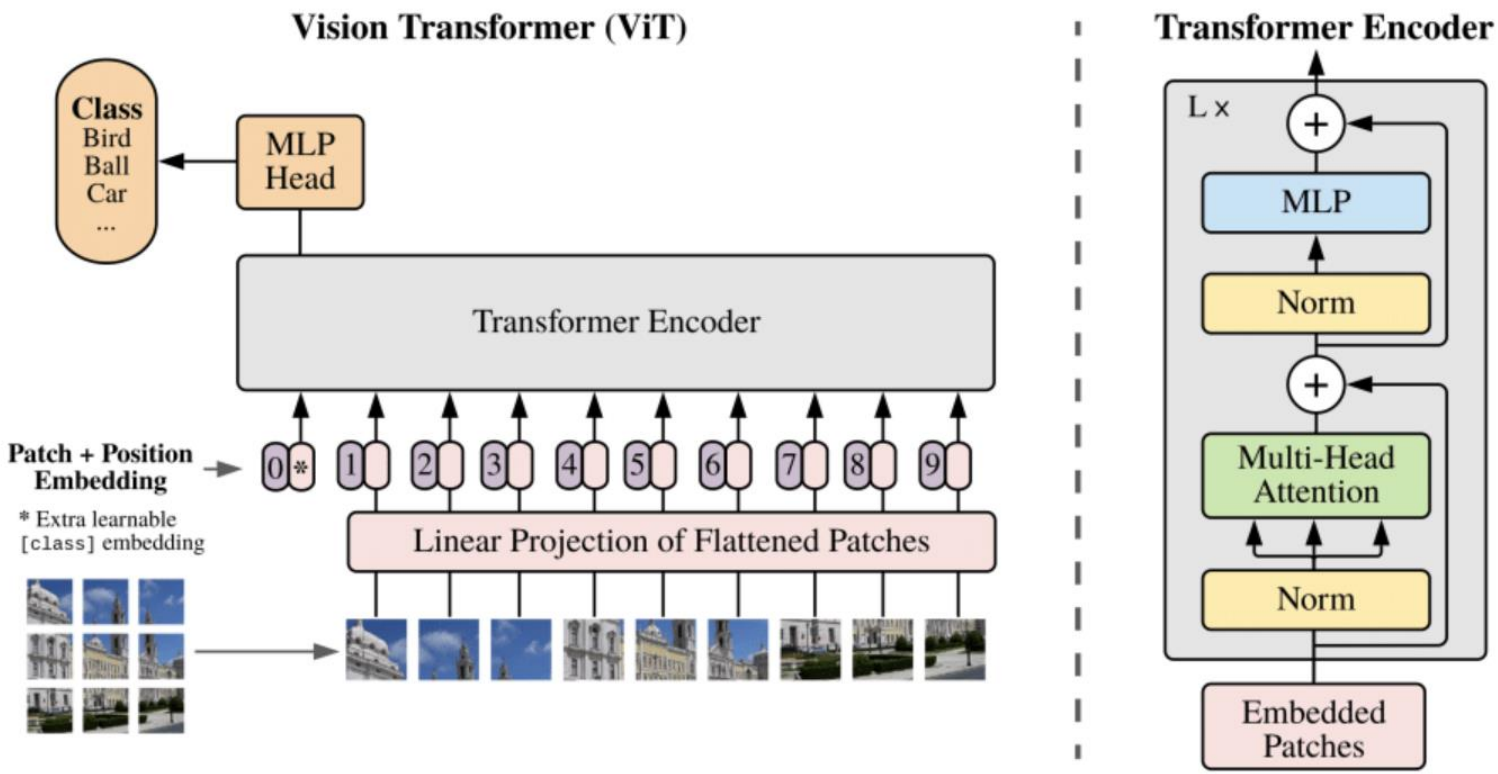
- Split images into a sequence of **patches**



- Each patch is treated as one token as input to ViT
  - A convolution layer with kernel P and stride P!
  - Or a linear layer on the flatten pixels

Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

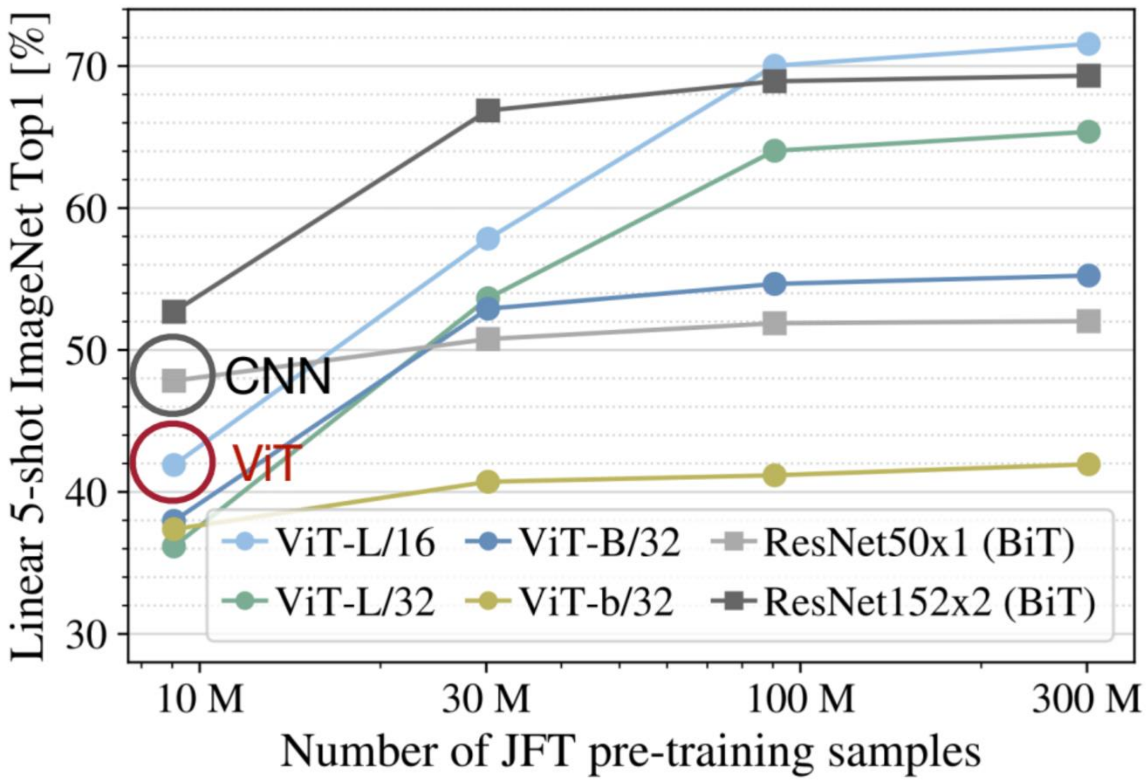
ViT



- The remaining is same as Transformer
  - As an encoder-only model

Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

Image Classification

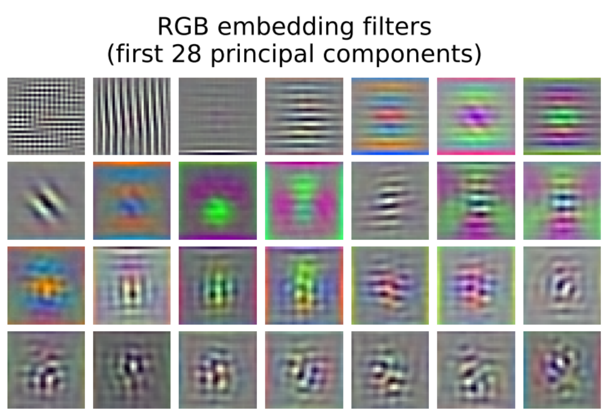


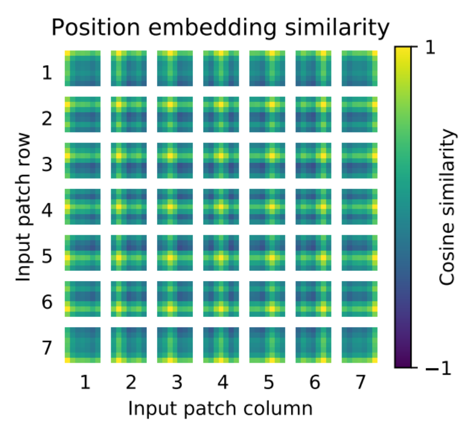
- Inferior performance compared to CNN when dataset size is limited – Why?

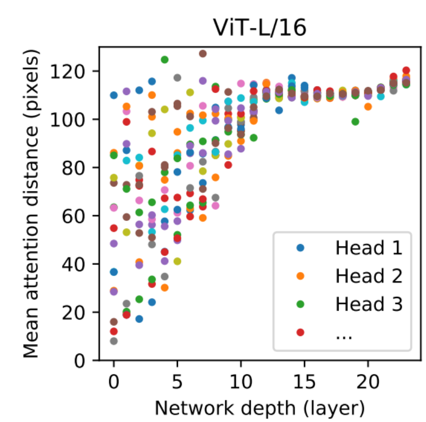
Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

Inductive Bias

- Convolutional Neural Networks
  - Locality
  - Sharing weights
- Vision Transformer
  - None!
  - Has to learn locality and dependency from data!
  - A lot lot lot lot lot lot lot lot of data!







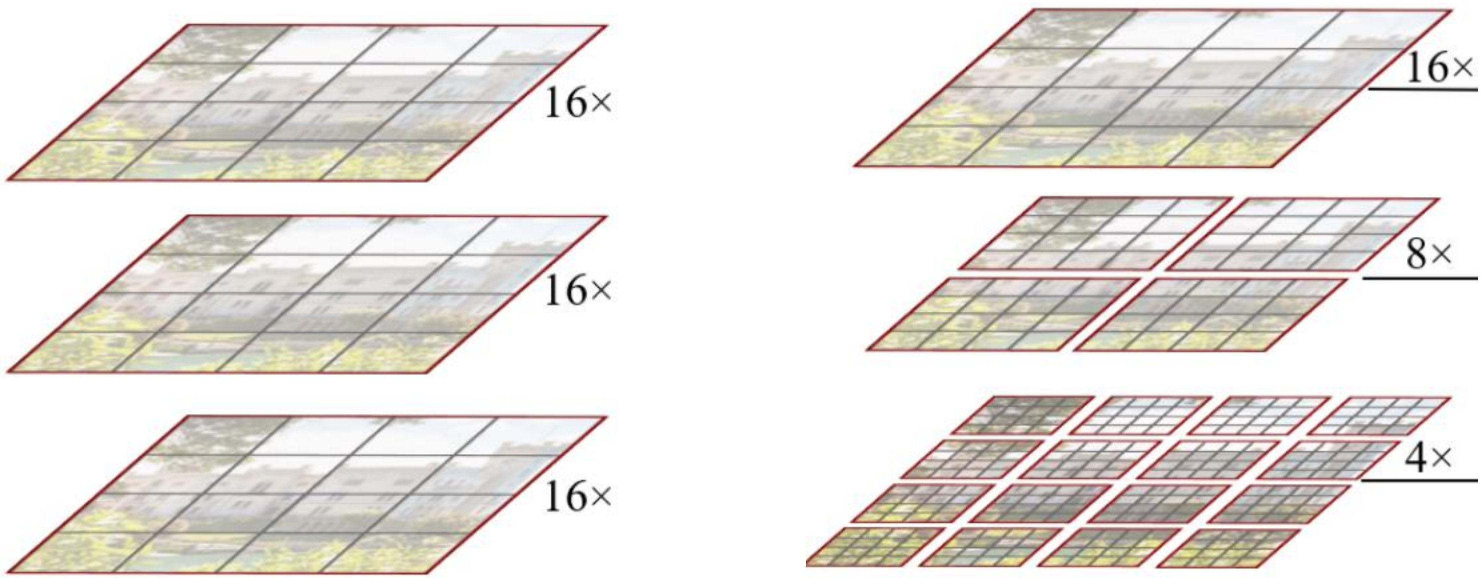
Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020.

Overview

- Vision Transformer Architecture
- Efficient ViT (Training and Modeling)
- Connection with Convolution
- Transformer Architectures in Vision

Swin-Transformer

- Window Attention
  - Restricts attention within a window of tokens
  - Brings locality back to transformer

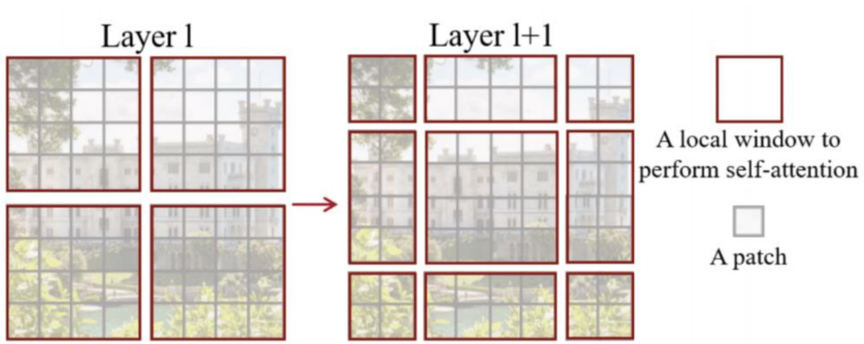


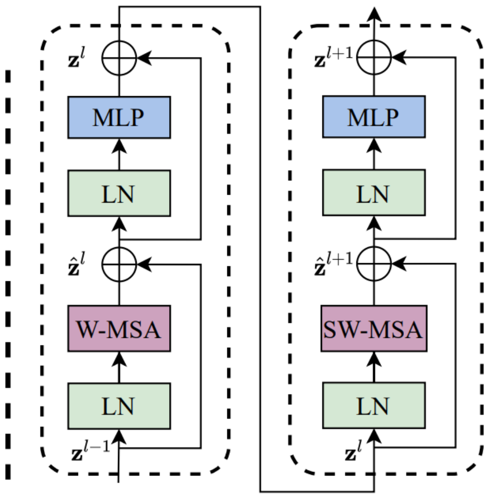
ViT Swin

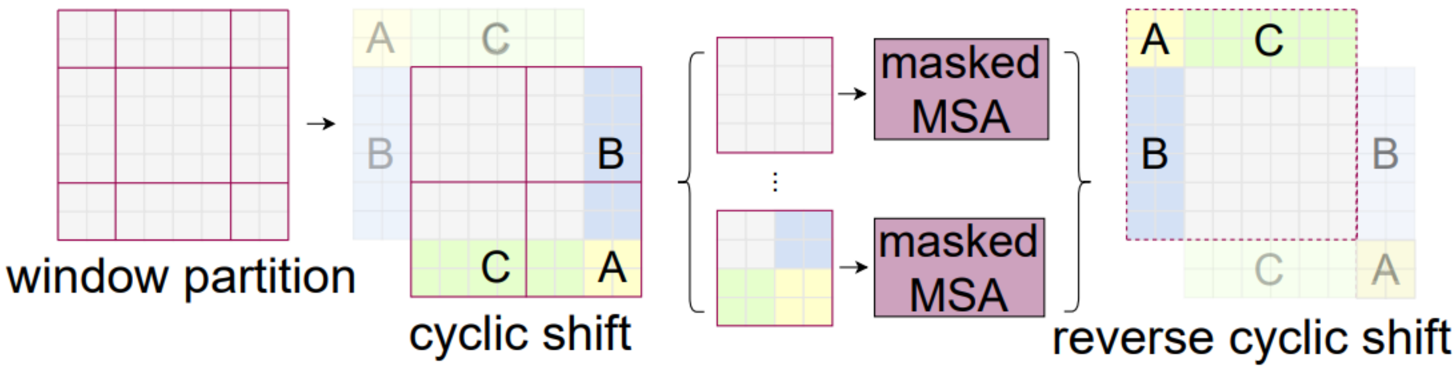
Liu et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. 2021.

Swin-Transformer

- How to compute attention across the windows?
  - Shift it!

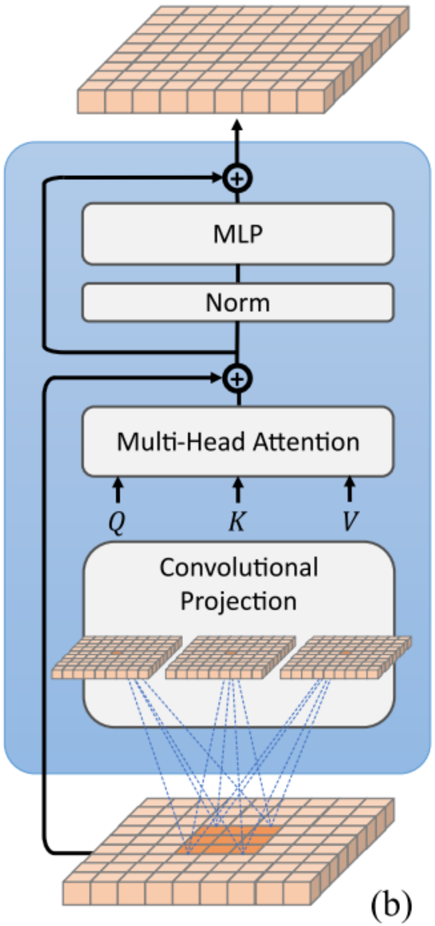




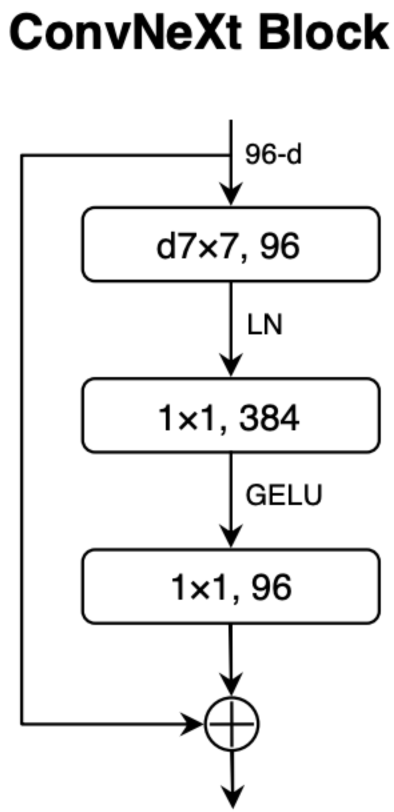


Liu et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. 2021.

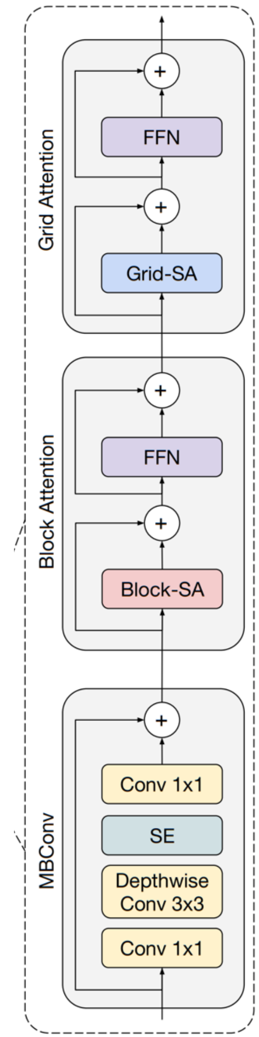
More Variants



CvT



ConvNext

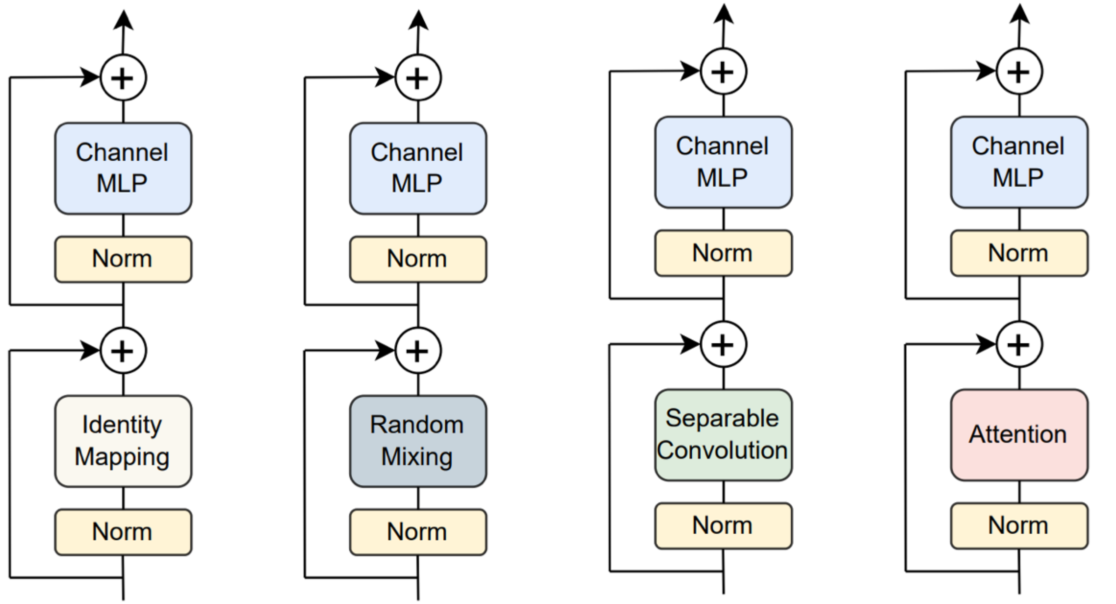


MaxViT

Wu et al. CvT: Introducing Convolutions to Vision Transformers. 2021.
Liu et al. A ConvNet for the 2020s. 2022.
Tu et al. MaxViT: Multi-Axis Vision Transformer. 2022

Metaformer

- Meta architecture of transformer matters



(e) IdentityFormer block (f) RandFormer block (g) ConvFormer block (h) Transformer block

- These variants produce similar classification results
- In practice, select the best one for your task

Yu et al. MetaFormer Is Actually What You Need for Vision. 2022.

Overview

- Vision Transformer Architecture
- Efficient ViT (Training and Modeling)
- Connection with Convolution
- Transformer Architectures in Vision

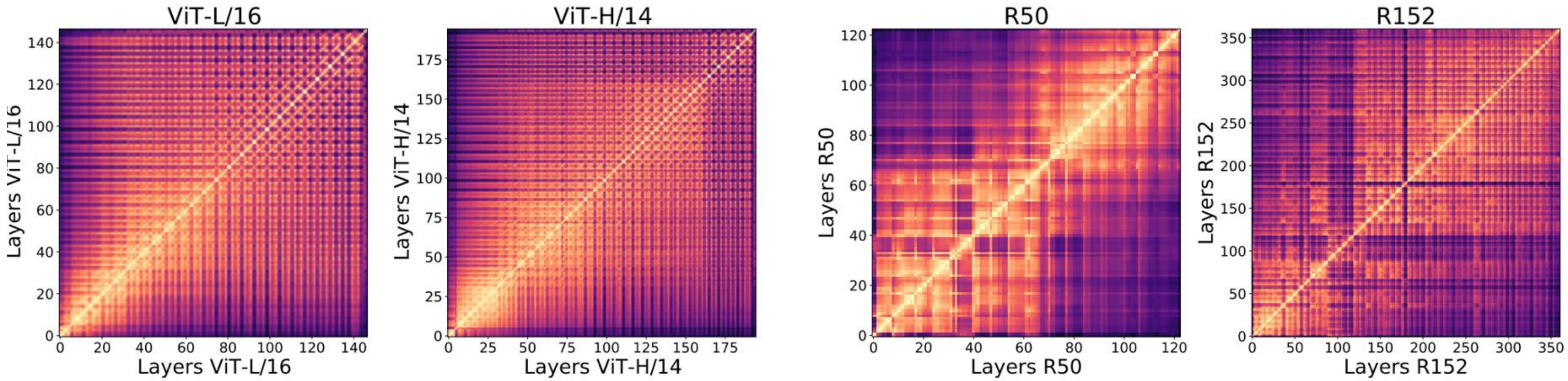
CNN and Transformer

- Convolutional Neural Networks
  - Locality
  - Sharing weights
- Vision Transformer
  - Learns global dependency from data
  - Dynamic weights from data

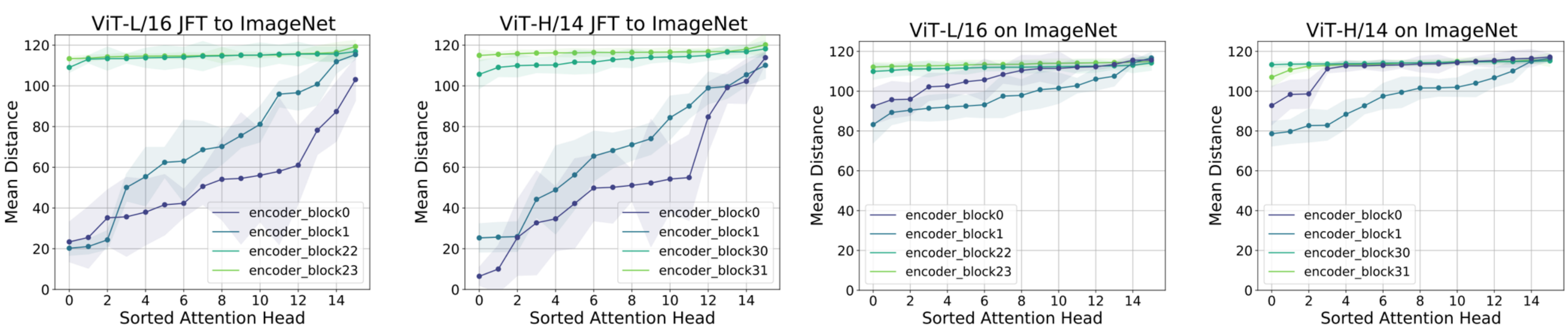
- But…are they really un-related?

ViT Learns Different Features

- Self-attention in ViT learns more uniform features across layers



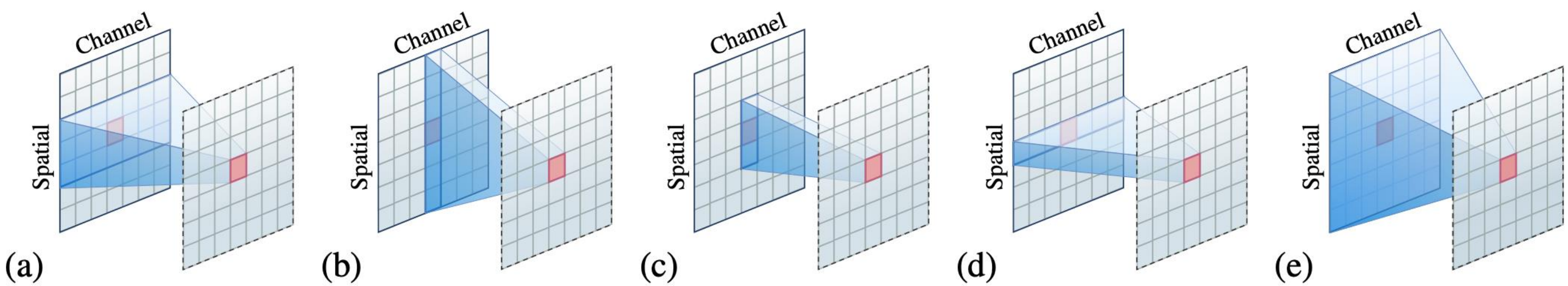
- Lower layers attend locally and globally, higher layers mainly attend globally
- Less data do not learn local attention at lower layers well



Raghu et al. Do Vision Transformers See Like Convolutional Neural Networks?. 2022.

Convolution -> Self-attention

- Difference
  - Locality vs. Global Dependency
  - Weight sharing vs. Dynamic weights

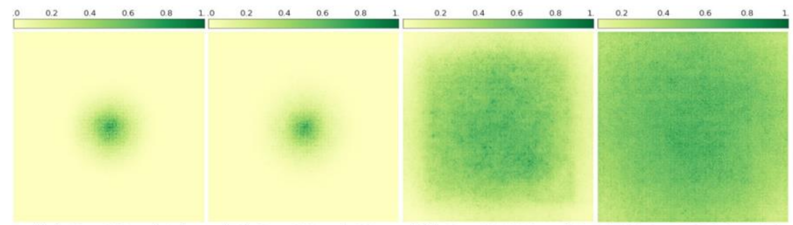


Convolution | Attention | Local Attention Depth-wise Conv. | Point-wise MLP 1x1 Conv. | MLP

Han et al. On the Connection between Local Attention and Dynamic Depth-wise Convolution. 2022.

Large-Kernel CNN

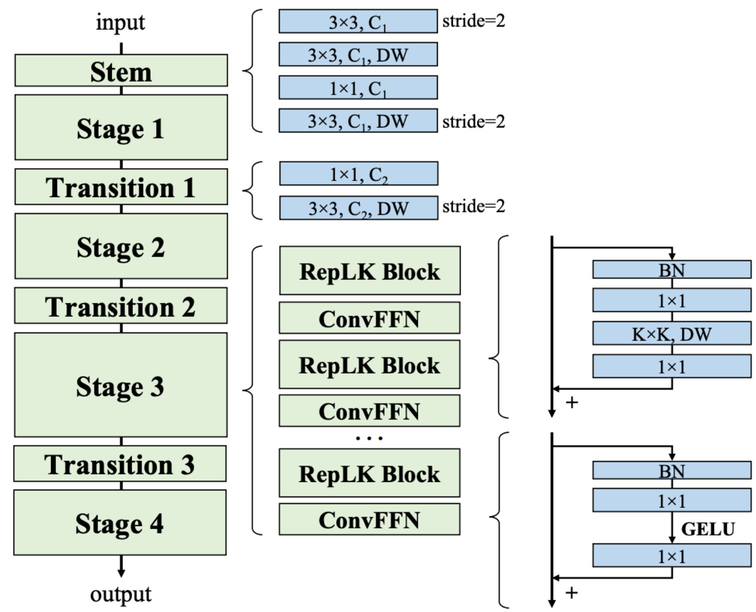
- Scaling up kernel size to 31x31



(A) ResNet-101 (B) ResNet-152 (C) RepLKNet-13 (D) RepLKNet-31

Table 5. RepLKNet with different kernel sizes. The models are pretrained on ImageNet-1K in 120 epochs with 224×224 input and finetuned on ADE20K with UperNet in 80K iterations. On ADE20K, we test the *single-scale* mIoU, and compute the FLOPs with input of 2048×512, following Swin.

	ImageNet			ADE20K		
Kernel size	Top-1	Params	FLOPs	mIoU	Params	FLOPs
3-3-3-3	82.11	71.8M	12.9G	46.05	104.1M	1119G
7-7-7-7	82.73	72.2M	13.1G	48.05	104.6M	1123G
13-13-13-13	83.02	73.7M	13.4G	48.35	106.0M	1130G
25-25-25-13	83.00	78.2M	14.8G	48.68	110.6M	1159G
31-29-27-13	83.07	79.3M	15.3G	49.17	111.7M	1170G



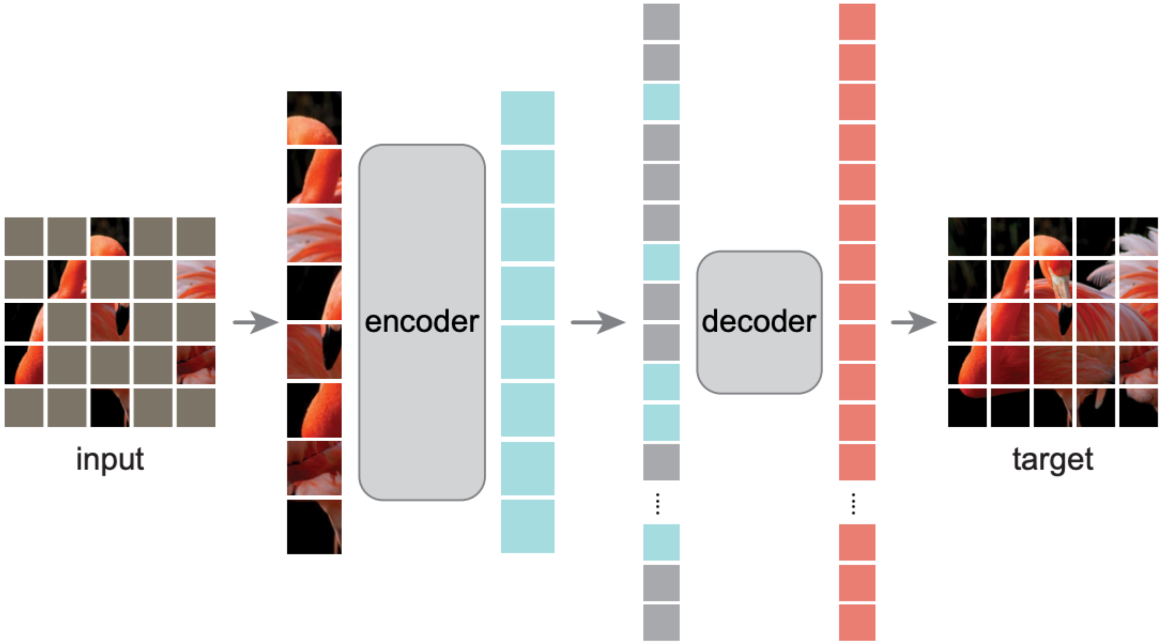
Ding et al. Scaling Up Your Kernels to 31x31: Revisiting Large Kernel Design in CNNs . 2022.

Overview

- Vision Transformer Architecture
- Efficient ViT (Training and Modeling)
- Connection with Convolution
- Transformer Architectures in Vision

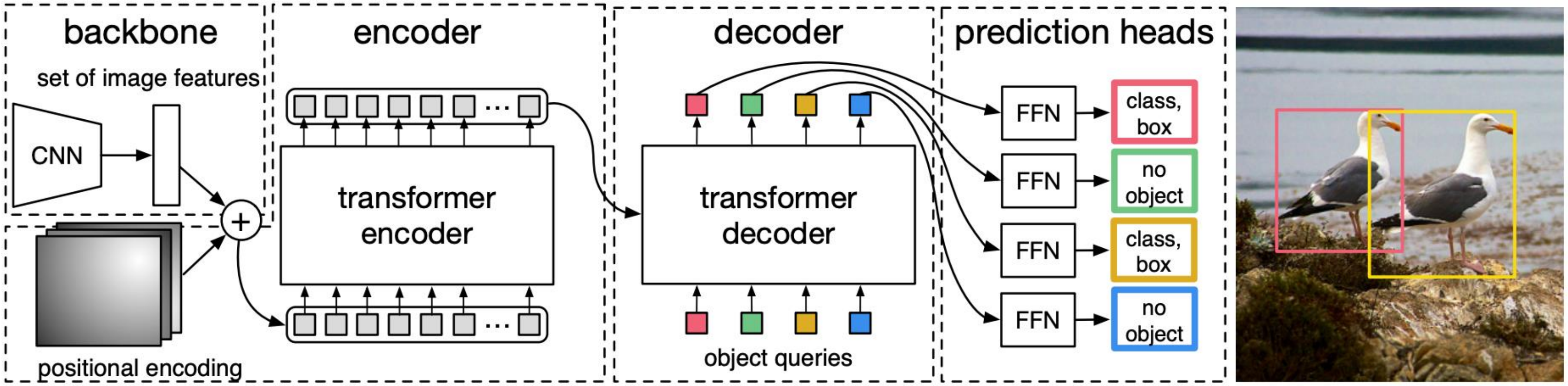
Encoder-Decoder - MAE

- Masked Auto-Encoder (MAE)

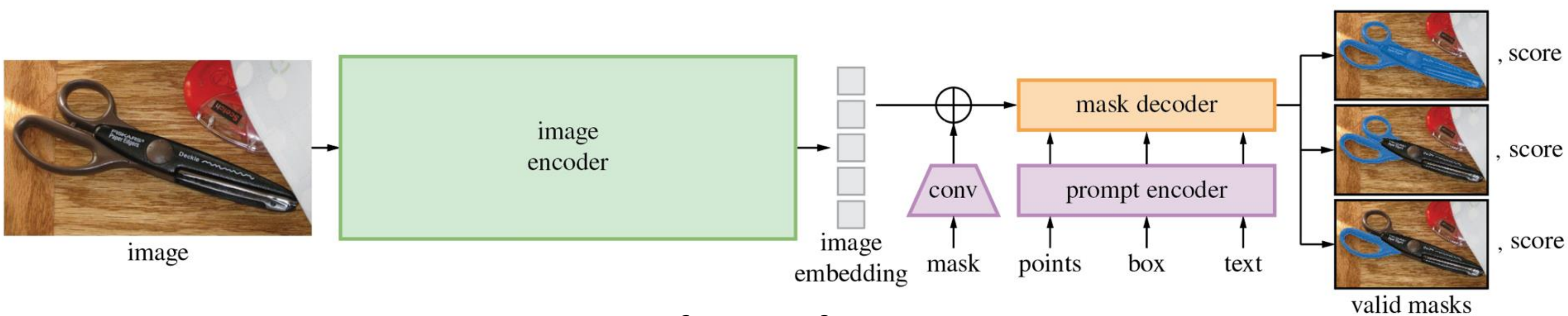


He et al. Masked Autoencoders Are Scalable Vision Learners. 2021.

Encoder-Decoder – DERT/SAM/etc.



DETR – Transformer for detection



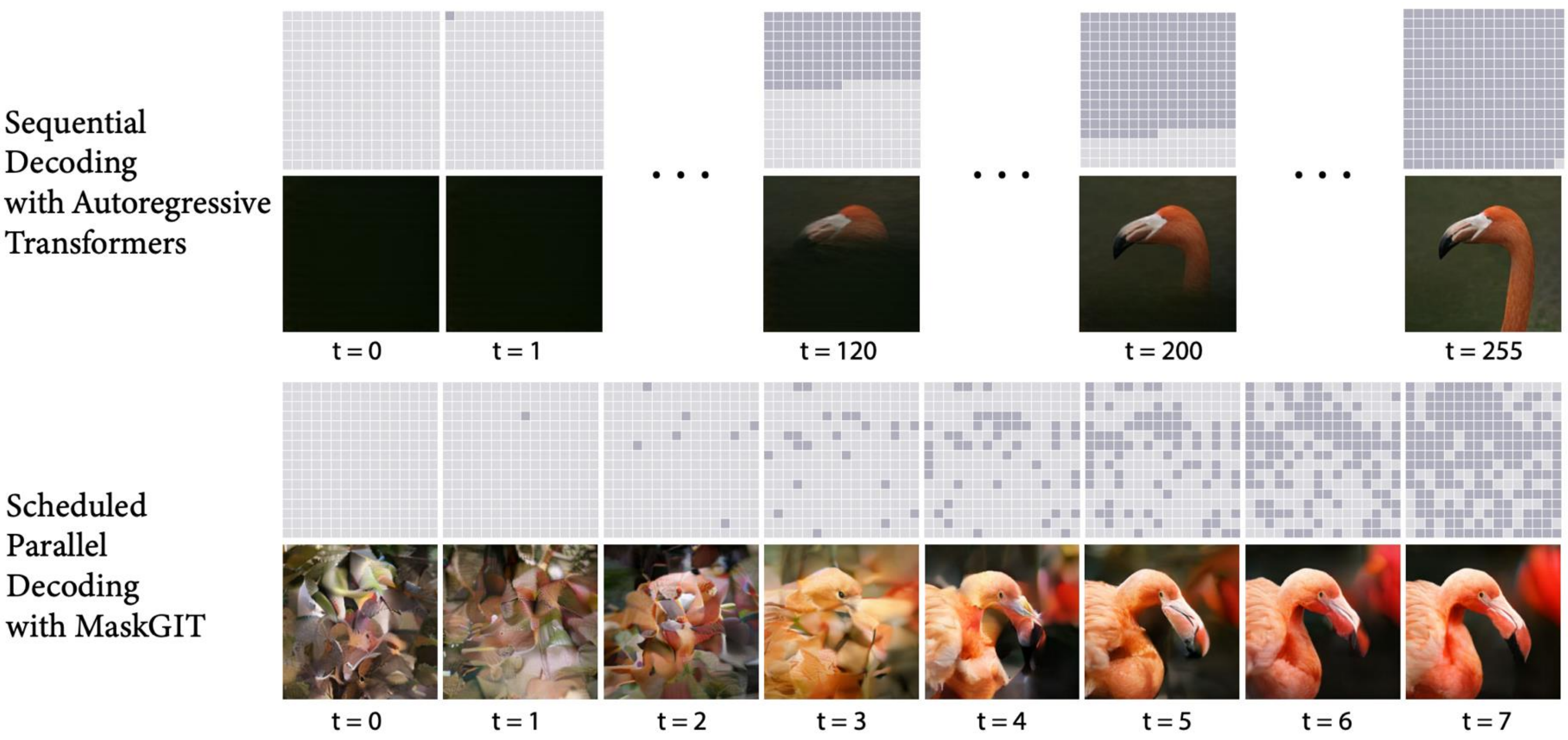
SAM – Transformer for segmentation

Carion et al. End-to-End Object Detection with Transformers. 2020.

Segment Anything.

Decoder-Only - MaskGiT

- Masked Generative Image Transformer (MaskGiT)
  - Image Generation with decoder-only arch.
  - Trained with masked token prediction

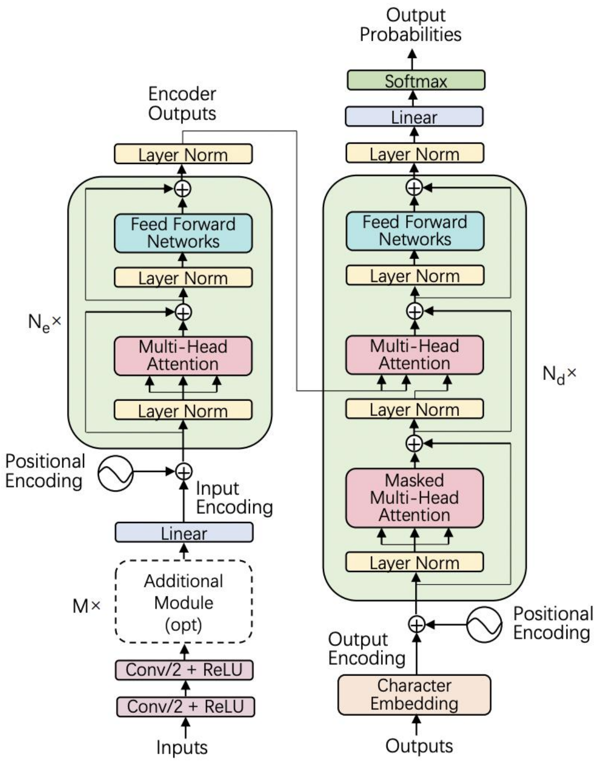


Chang et al. MaskGIT: Masked Generative Image Transformer. 2022.

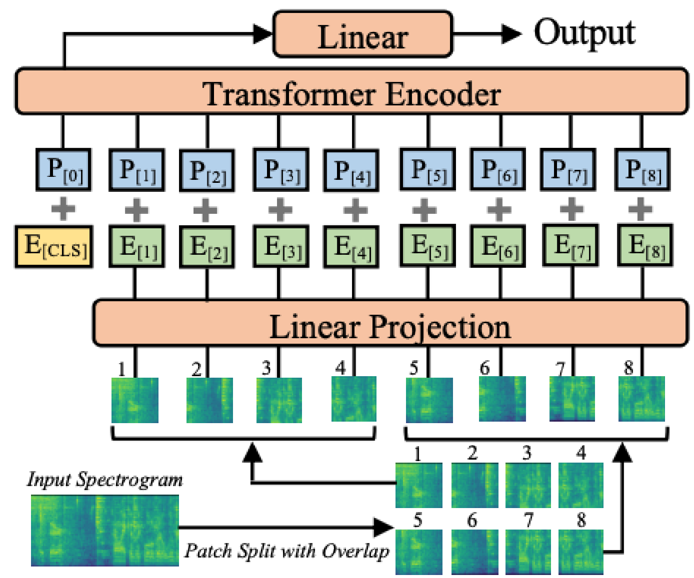
Poll

Transformer in Audio

Transformer in Audio



Speech Transformer for ASR



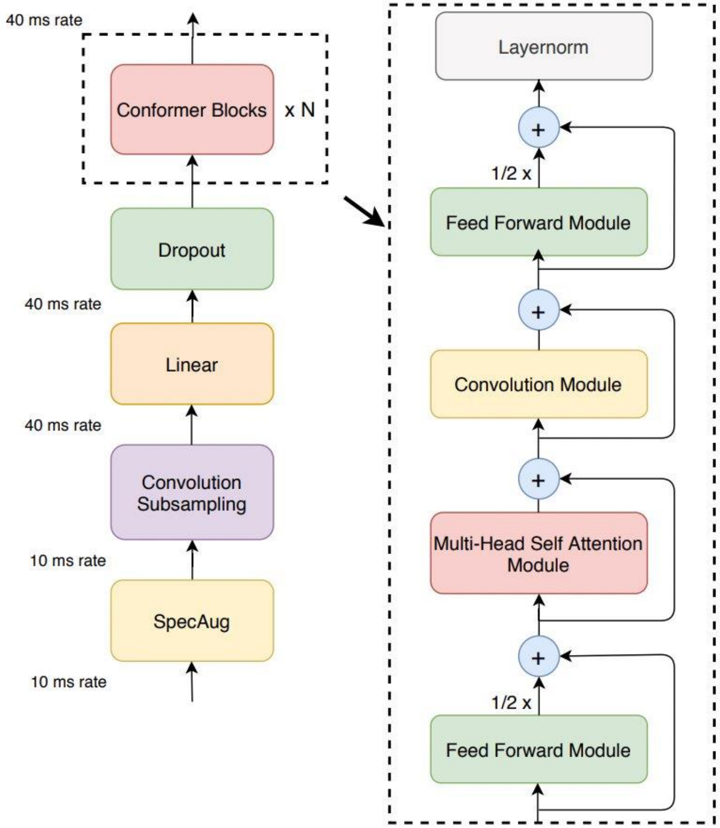
Audio Spectrogram Transformer

[1] Dong, Linhao, Shuang Xu, and Bo Xu. "Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition." *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018.
[2] Gong, Yuan, Yu-An Chung, and James Glass. "Ast: Audio spectrogram transformer." *arXiv preprint arXiv:2104.01778* (2021).

Conformer

- The Conformer architecture augments a transformer by embedding convolution layers within the transformer blocks.

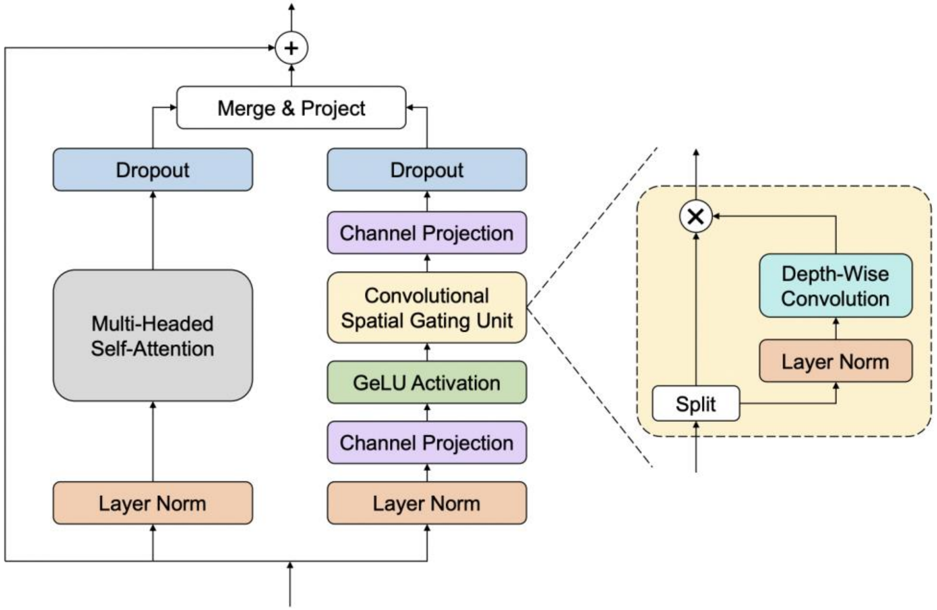
- Transformers capture global dependencies, CNNs capture local features efficiently.



Gulati, Anmol, et al. "Conformer: Convolution-augmented transformer for speech recognition." *arXiv preprint arXiv:2005.08100* (2020).

Branchformer

- Branchformer introduces a parallel-branch layer.
- One branch uses self-attention, while the other branch employs a convolutional-gated MLP (cgMLP).
- The two branches are merged using either concatenation or weighted average.
- The branch weights reveal how global and local relationships are utilized across layers.



Peng, Yifan, et al. "Branchformer: Parallel mlp-attention architectures to capture local and global context for speech recognition and understanding." *International Conference on Machine Learning*. PMLR, 2022.

Parameter Efficient Tuning

Overview

- Parameter Efficient Tuning Methods
  - Prompt Tuning
  - Adapter
  - LoRA
- Interpretation

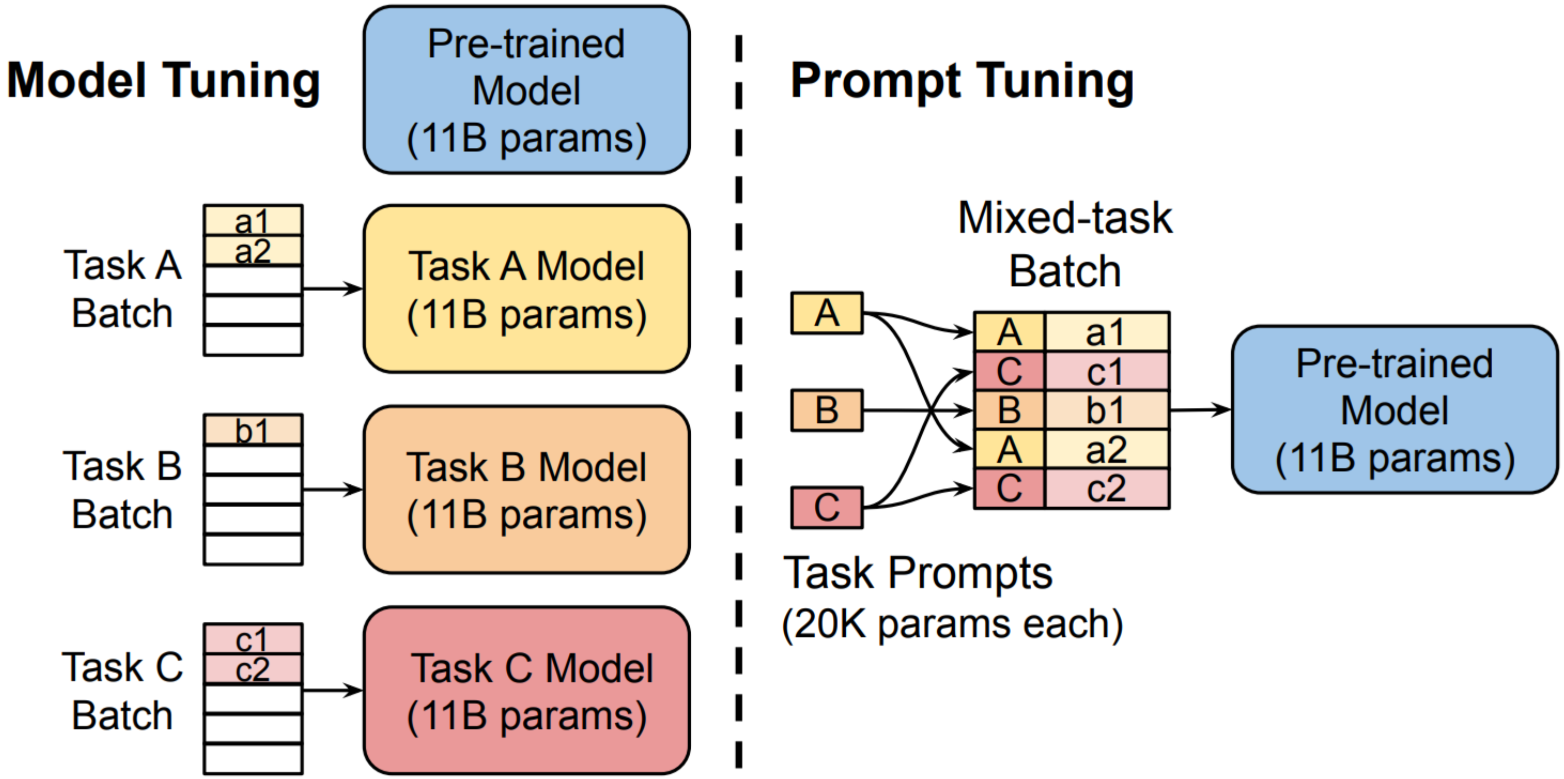
Parameter Efficient Tuning

- Traditionally, you need to fine-tune entire network on specific downstream tasks

- Parameter Efficient Tuning – Only tune a small proportion of parameters of the pre-trained transformer
  – Prompt tuning
  – Adapter
  – LoRA

Prompt Tuning

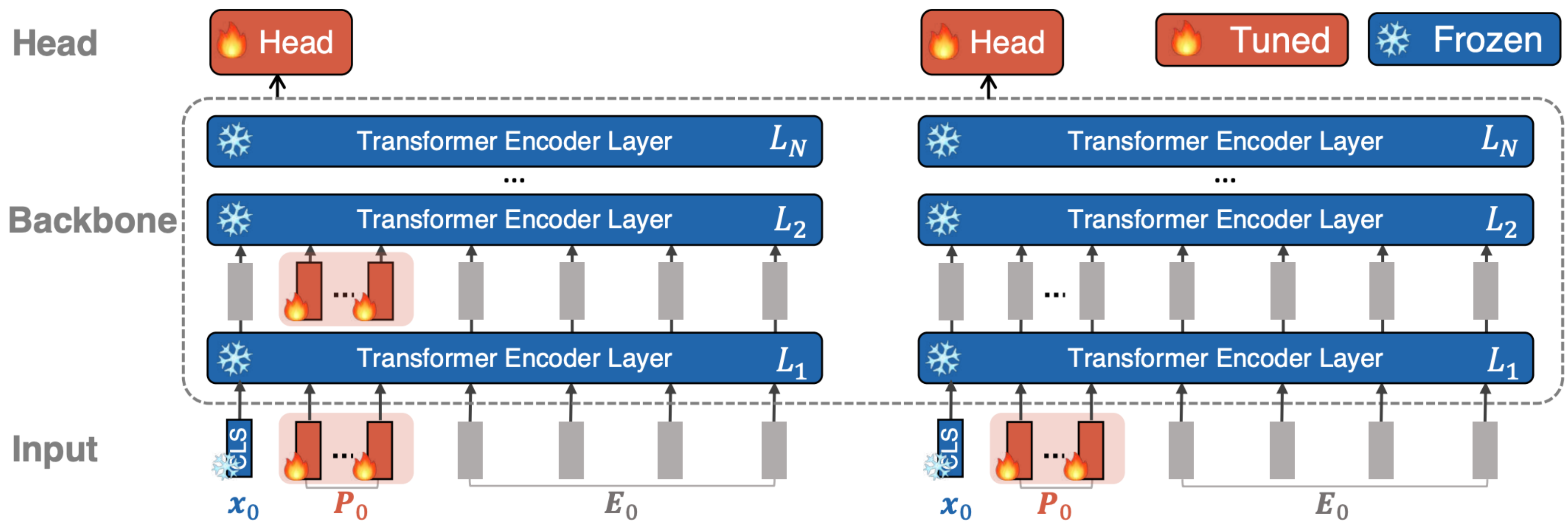
- Only learns a set of 'prompt' or 'token' for each task



Lester et al. The Power of Scale for Parameter-Efficient Prompt Tuning. 2021.

Visual Prompt Tuning

- Prompt tuning also applicable to vision transformers

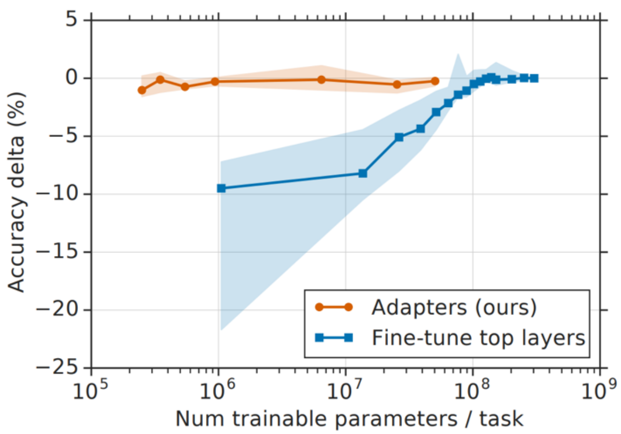


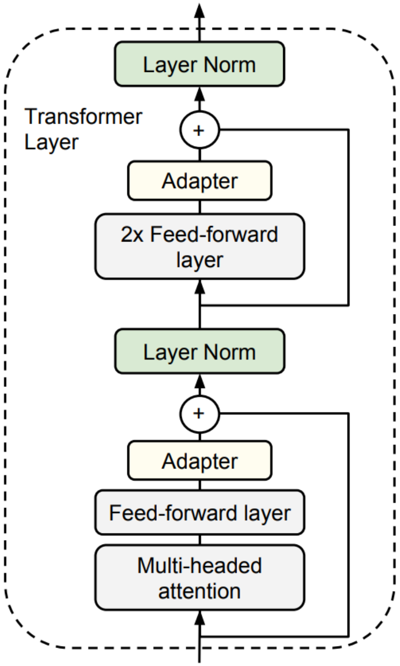
(a) Visual-Prompt Tuning: Deep (b) Visual-Prompt Tuning: Shallow

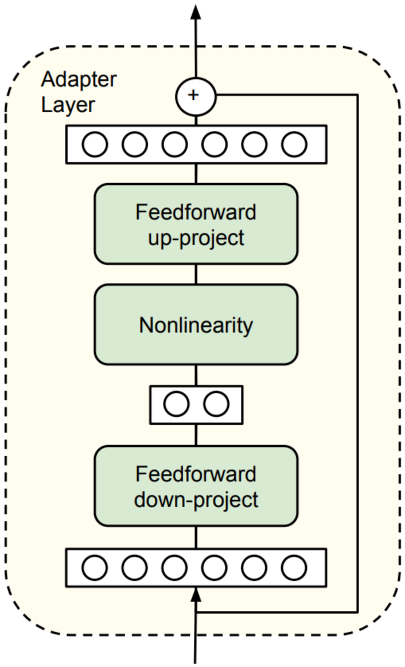
Jia et al. Visual Prompt Tuning. 2022.

Adapter

- Insert MLP at Feed-forward layers





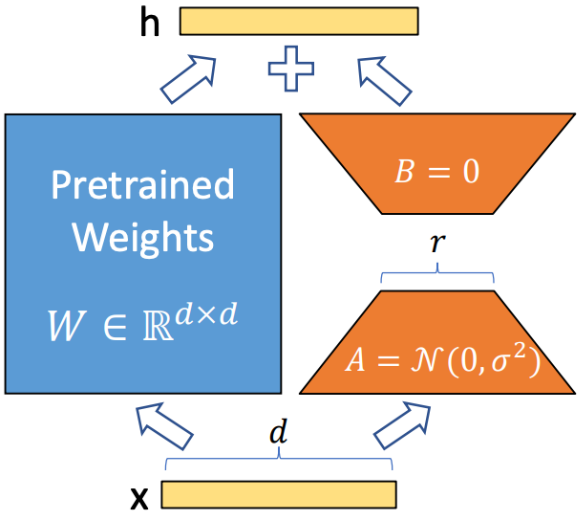


Houlsby et al. Parameter-Efficient Transfer Learning for NLP. 2022.

LoRA

- Low-rank Adaptation (LoRA)
- No activation in-between
- A and B can be fused into W

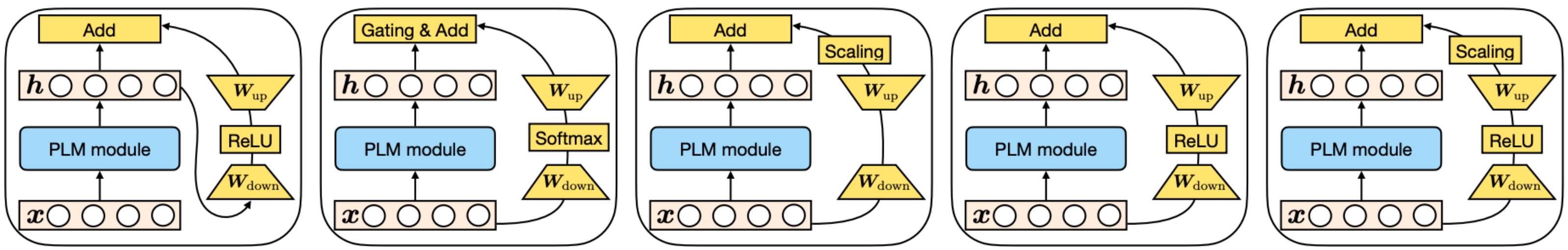
$$h = W_0x + \Delta W x = W_0 x + BAx$$



Hu et al. LoRA: Low-Rank Adaptation of Large Language Models. 2022.

Interpretation

- Essentially, all parameter efficient tuning methods do the same thing – modifying pre-trained features with minimal amount of parameters



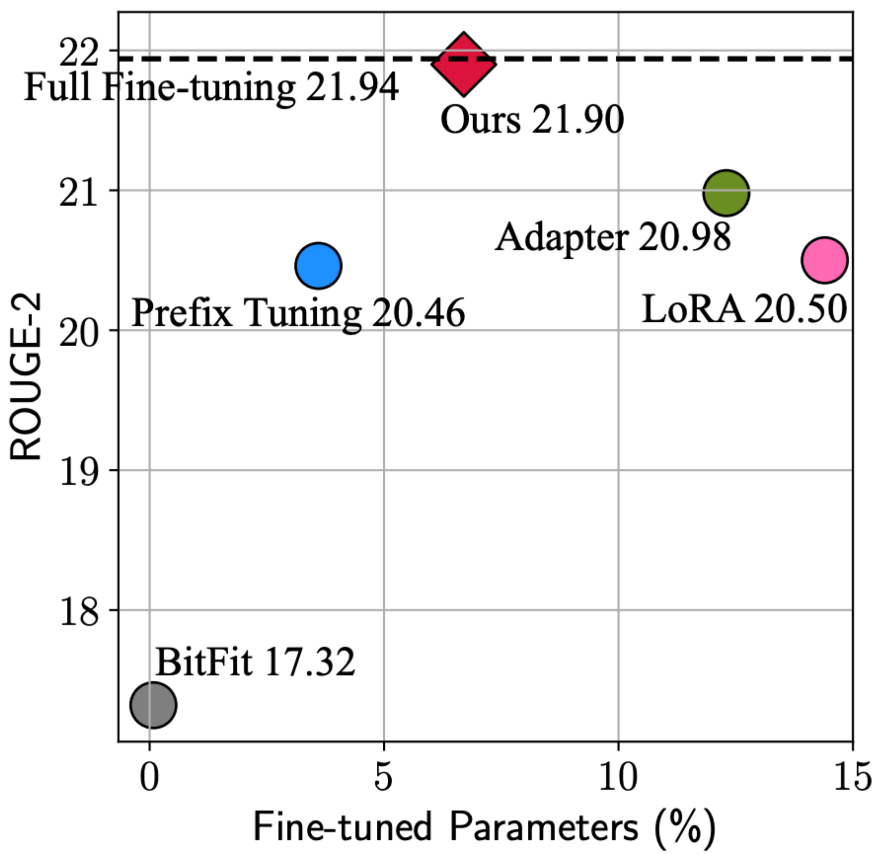
(a) Adapter (b) Prefix Tuning (c) LoRA (d) Parallel Adapter (e) Scaled PA

Method	Δh functional form	insertion form	modified representation	composition function
		Existing Methods		
Prefix Tuning	$\text{softmax}(xW_q P_k^\top)P_v$	parallel	head attn	$h \leftarrow (1-\lambda)h + \lambda\Delta h$
Adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	sequential	ffn/attn	$h \leftarrow h + \Delta h$
LoRA	$xW_{\text{down}}W_{\text{up}}$	parallel	attn key/val	$h \leftarrow h + s \cdot \Delta h$

He et al. Towards a Unified View of Parameter-Efficient Transfer Learning. 2022.

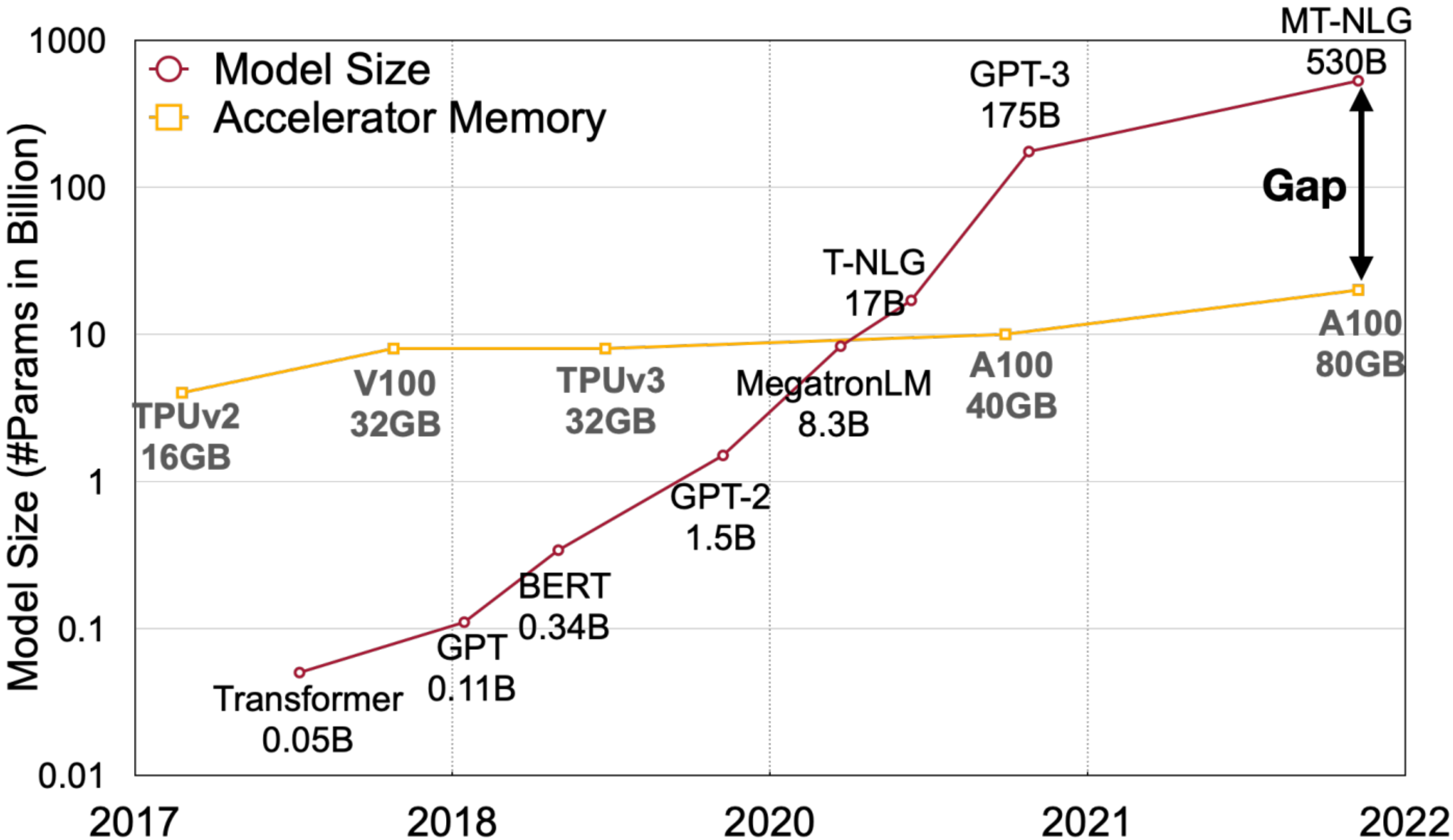
Parameter-Efficient Tuning

- Performance close to full fine-tuning while just train less than 15% of original parameters



Scaling Laws

“Magic” of Transformer - Scaling

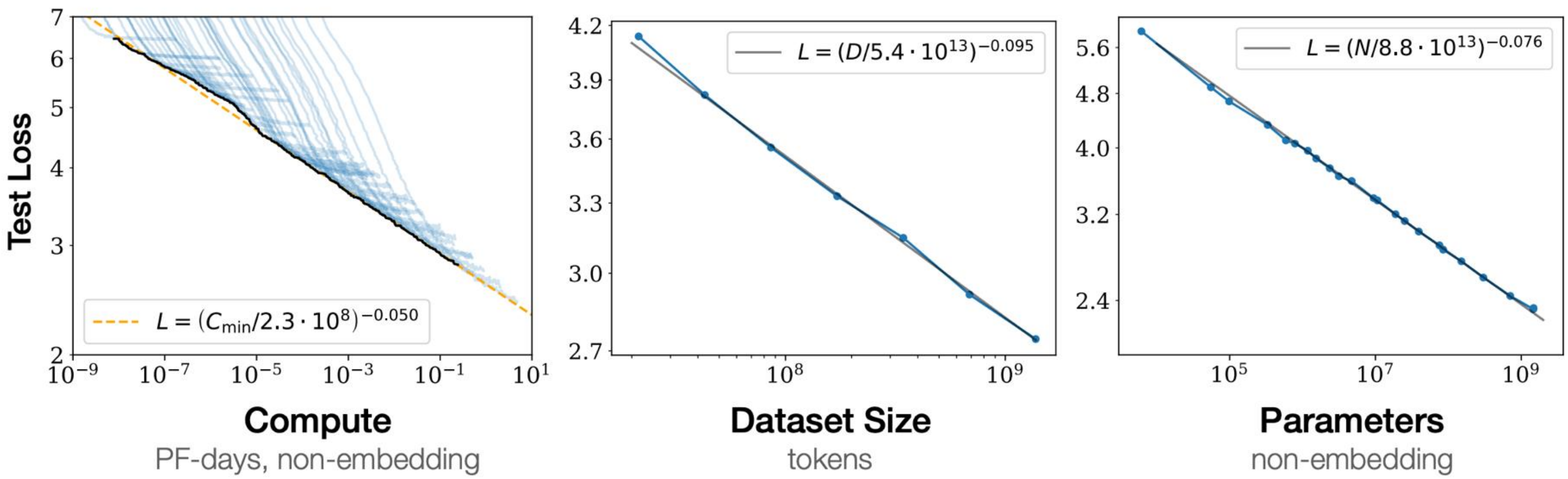


- Performance gets better as transformer scales up

Xiao et al. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. 2024.

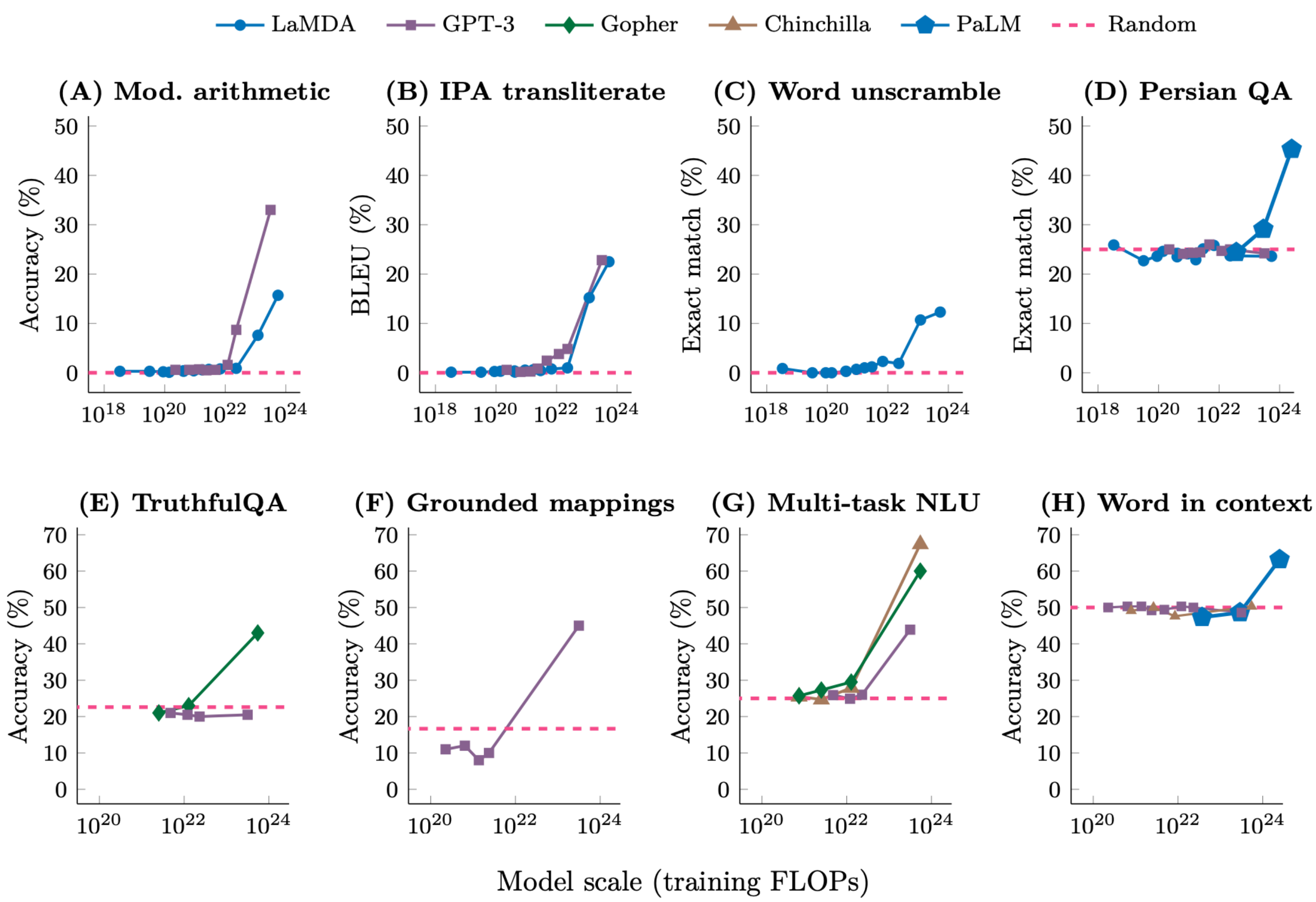
Scaling Law

- For decoder-only models, the final performance is only related to **Compute**, **Data Size**, and **Parameter Size**
  - power law relationship for each factor
  - w/o constraints by the others



Kaplan et al. Scaling Laws for Neural Language Models. 2020.

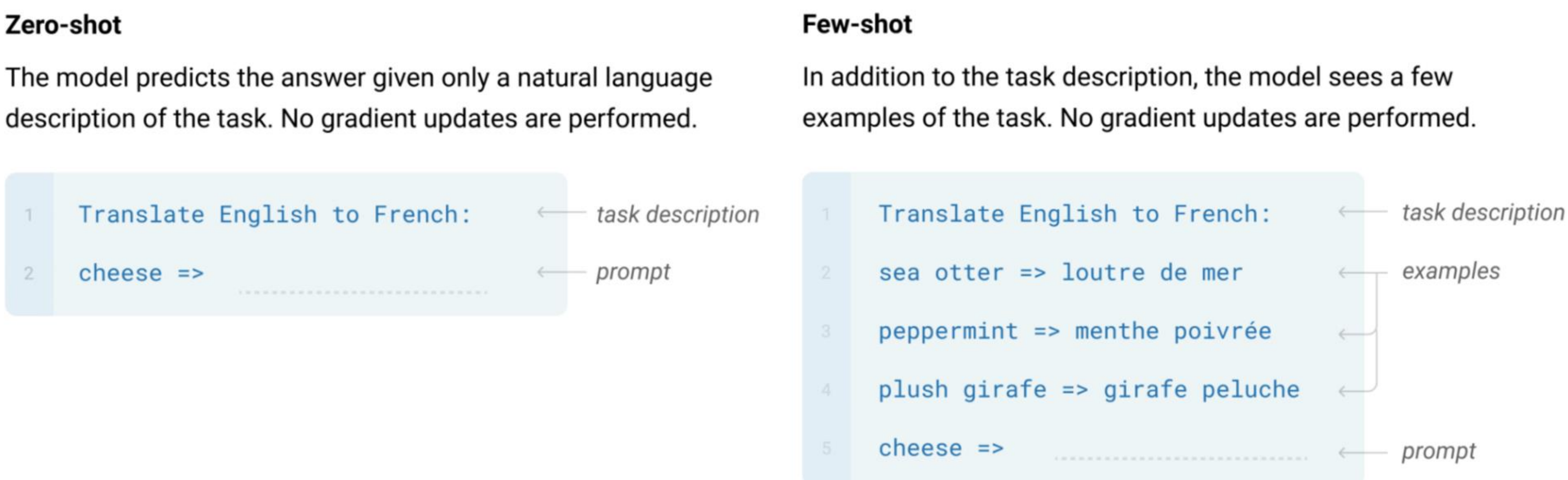
“Emergent” Capability



Wei et al. Emergent Abilities of Large Language Models. 2020.

In-Context Learning

- Scaled models can generalize to new tasks without fine-tuning!
  - Zero-shot
  - Few-shot



Language Models are Few-shot Learners.

We learned…

- Transformer Architecture
- Transformer in Language
- Transformer in Vision
- Transformer in Audio
- Parameter Efficient Tuning
- Scaling Laws