Image analysis

5. Vision transformers

Clément Gorin
clement.gorin@univ-paris1.fr

Sorbonne School of Economics
Masters in Development Economics

Introduction

Introduction

Vaswani et al. (2017) introduced the transformer architecture was in the context of machine translation

- Dosovitskiy et al. (2021) adapted this architecture to image modelling with **vision transformers** (ViT)
- ViTs essentially replace the convolution operation with another operation called **self-attention**
- Emergence of self-supervised **pre-training** (e.g. He et al. 2022) and distillation (e.g. Caron et al. 2021)
- Large ViT have achieved state of the art performance on nearly all vision tasks (Khan et al. 2022)

Introduction

What makes transformers competitive is their ability to model efficiently[1] complex interactions among the inputs

- **Self-attention** mechanisms flexibly captures interaction patterns among the inputs i.e. few inductive biases
- **Long range dependencies** are not an issue, as the model processes the entire image at once
- **Multi-modality** as most data structures can be formatted as sequences e.g. images, text, audio

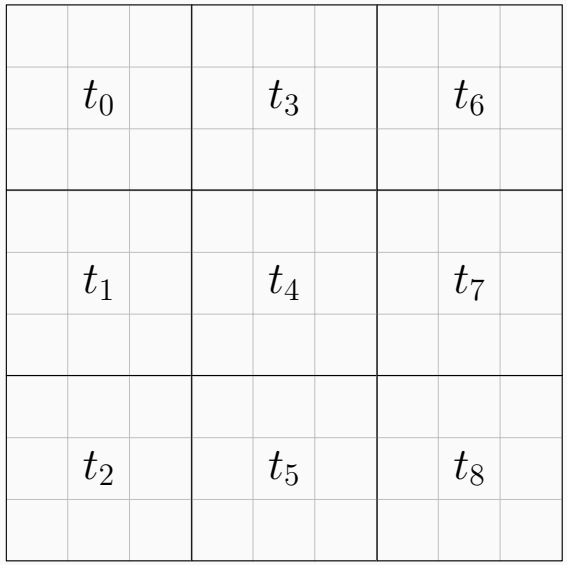
[1] Like convolutions, self-attention is computationally efficient and can be executed in parallel.

Overview

Overview

ViT treats images as sequences by splitting an image into a series of tile vectors or tokens $x_i \in \mathbb{R}^{(dhw) \times t}$

- This matrix is projected using a linear layer $x_i^{(0)} \in \mathbb{R}^{d_x \times t}$ e.g. $d_x = 784$
- Positional information is added to each vector (more on this)
- A classification token is can be added in the first index position[2]



[2] Tile-level predictions use tokens, while image-level predictions can be performed by pooling across the sequence or using the classification token.

Overview

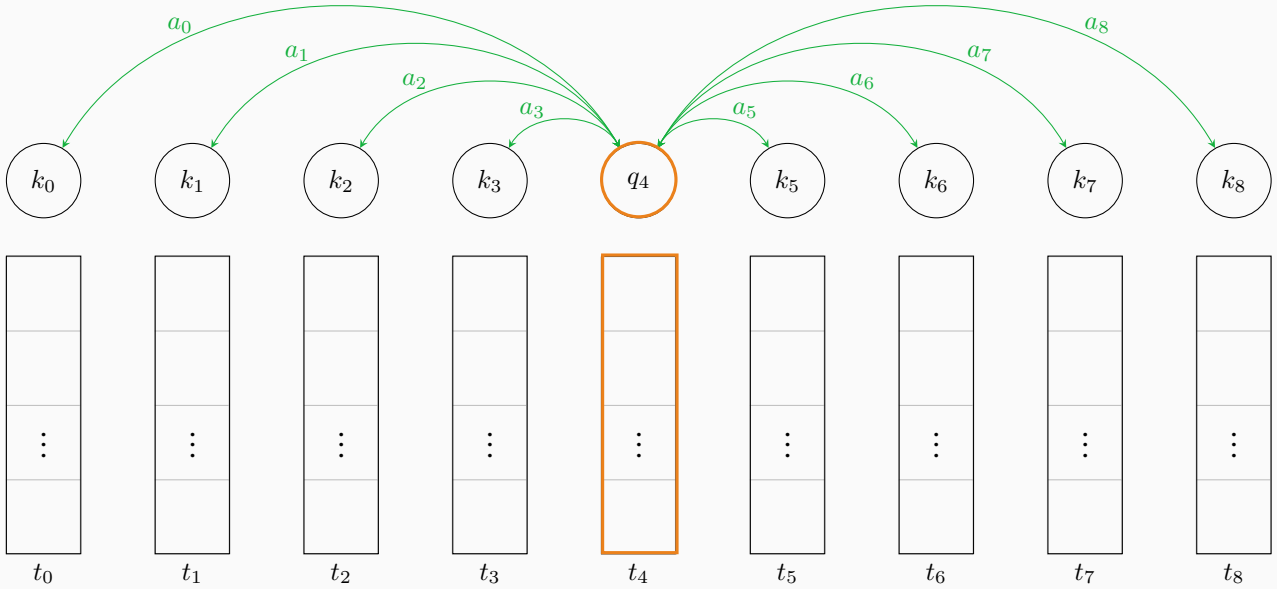
A transformer model is made of series of transformer modules, each module essentially performs two operations

1. **Self-attention** operate across the sequence, refining each feature according to relationships between tokens
2. **Pointwise MLP** operates across features, refining the features of each token independently

Each module takes a sequence $x^{(l-1)} \in \mathbb{R}^{d_x \times t}$ and returns an updated representation $x^{(l)}$ with the same size

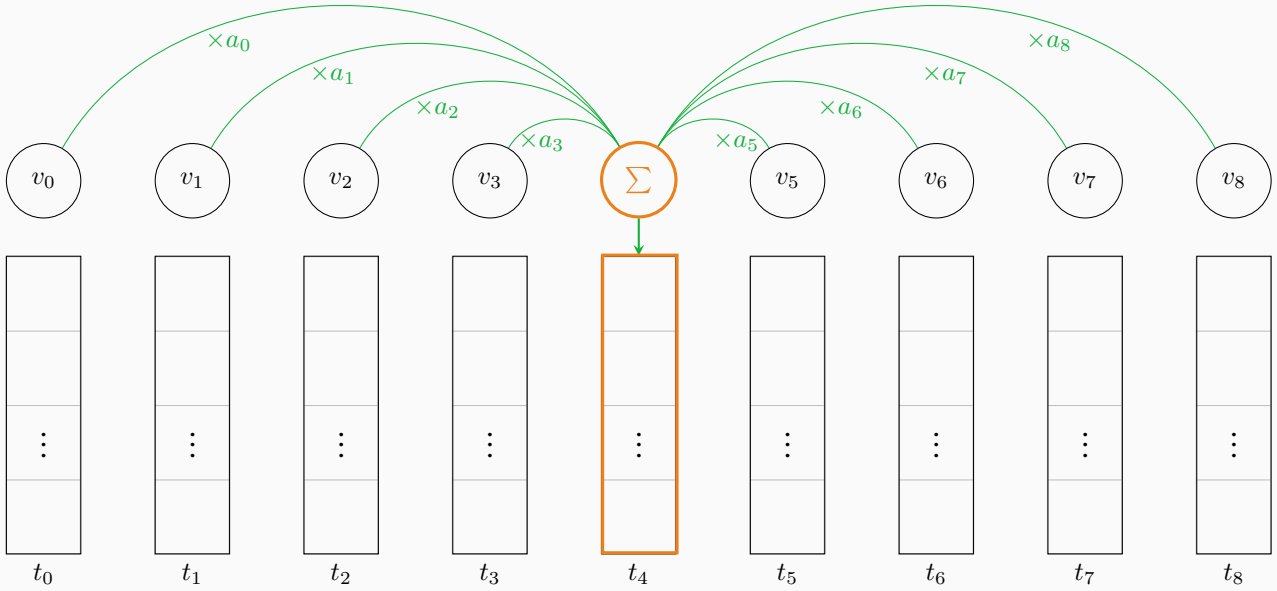
Self-attention

The self-attention mechanisms augments the embedding for a vector with contextual information from other vectors



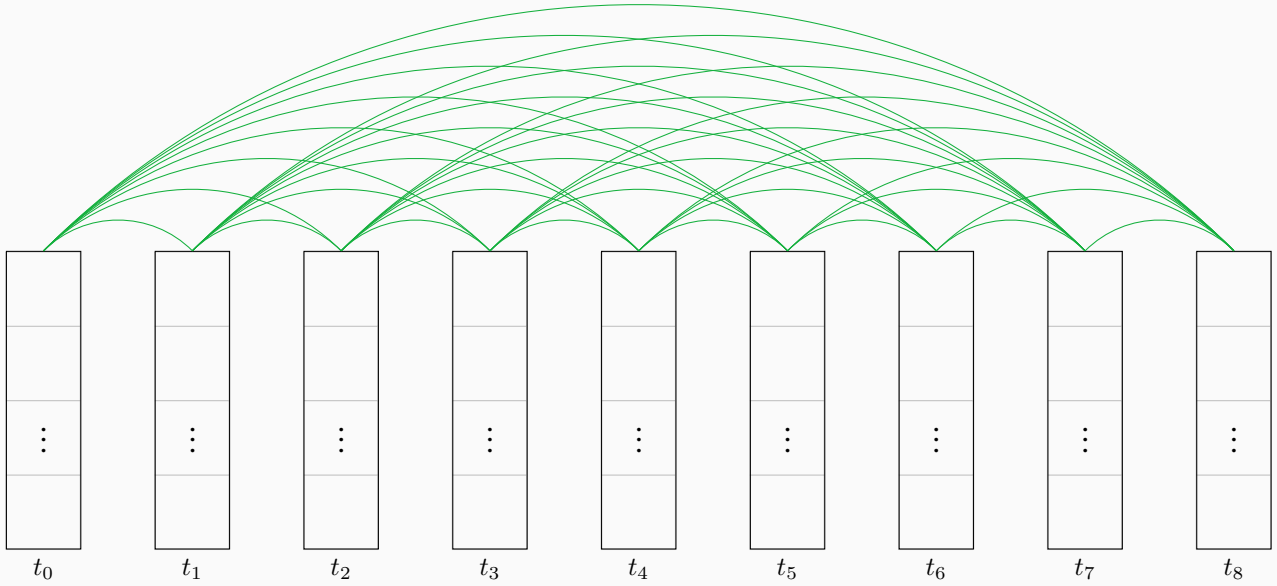
Each token can “ask” a question or **query**, while the others can “propose” a potentially matching answer or **key**

Attention weights reflect how well the query matches the keys and whether the tokens should exchange information



Tokens can provide information or **values** and the target token is updated using a weighted sum of these values

Using the same mechanism, information is exchanged simultaneously by all the tokens in the sequence



Each token embedding is enriched with contextual information from the other tokens i.e. spatial connectivity in this case

Self-attention in ViT refines each patch embedding by attending to complementary object parts

- For instance, the token for a patch with a furry texture could represent different objects e.g. cat, rug
- If neighbouring tokens contain features representing whiskers, its embedding shifts toward the concept of *cat*
- If they contain features about coarse fibres or fringe, the embedding moves toward *rug* instead

In self-attention mechanisms, each token in the sequence simultaneously plays three distinct roles[3]

Query	Current token seeking information
Key	Context token being compared to the query
Value	Content used to update the current token

To allow a token to behave differently in each role, we compute query, key, and value vectors for every token using three distinct learnable projection matrices

[3] The terms query, key, and value are inspired by dictionary structures in programming.

Consider parameter matrices β_q, β_k and β_v projecting each of the t input tokens $x_{it} \in \mathbb{R}^{d_x}$ into a representation of its role

$$
\begin{aligned}
q_{it} &= x_{it} \cdot \beta_q, \quad \beta_q \in \mathbb{R}^{d_x \times d_k} \\
k_{it} &= x_{it} \cdot \beta_k, \quad \beta_k \in \mathbb{R}^{d_x \times d_k} \\
v_{it} &= x_{it} \cdot \beta_v, \quad \beta_v \in \mathbb{R}^{d_x \times d_v}
\end{aligned}
$$

where d_k and d_v are the chosen dimensions for the query, key and value vector respectively e.g. $d_k = d_v = 128$

- These parameters are estimated so the model learns the appropriate projection parameters
- The scaled dot product alignment function computes the relevance of a particular key to a query

Keys	Softmax ($\Uparrow$)					
$x_6 \xrightarrow{\beta_k} k_6$	$q_1 \cdot k_6$	$q_2 \cdot k_6$	$q_3 \cdot k_6$	$q_4 \cdot k_6$	$q_5 \cdot k_6$	$q_6 \cdot k_6$
$x_5 \xrightarrow{\beta_k} k_5$	$q_1 \cdot k_5$	$q_2 \cdot k_5$	$q_3 \cdot k_5$	$q_4 \cdot k_5$	$q_5 \cdot k_5$	$q_6 \cdot k_5$
$x_4 \xrightarrow{\beta_k} k_4$	$q_1 \cdot k_4$	$q_2 \cdot k_4$	$q_3 \cdot k_4$	$q_4 \cdot k_4$	$q_5 \cdot k_4$	$q_6 \cdot k_4$
$x_3 \xrightarrow{\beta_k} k_3$	$q_1 \cdot k_3$	$q_2 \cdot k_3$	$q_3 \cdot k_3$	$q_4 \cdot k_3$	$q_5 \cdot k_3$	$q_6 \cdot k_3$
$x_2 \xrightarrow{\beta_k} k_2$	$q_1 \cdot k_2$	$q_2 \cdot k_2$	$q_3 \cdot k_2$	$q_4 \cdot k_2$	$q_5 \cdot k_2$	$q_6 \cdot k_2$
$x_1 \xrightarrow{\beta_k} k_1$	$q_1 \cdot k_1$	$q_2 \cdot k_1$	$q_3 \cdot k_1$	$q_4 \cdot k_1$	$q_5 \cdot k_1$	$q_6 \cdot k_1$
Queries	$x_1 \xrightarrow{\beta_q} q_1$	$x_2 \xrightarrow{\beta_q} q_2$	$x_3 \xrightarrow{\beta_q} q_3$	$x_4 \xrightarrow{\beta_q} q_4$	$x_5 \xrightarrow{\beta_q} q_5$	$x_6 \xrightarrow{\beta_q} q_6$

These scores are scaled and normalised using the $softmax$ transformation applied column-wise to obtain the attention weights a_{ij} (more on this)

Values	Sum ($\Uparrow$)					
$x_6 \xrightarrow{\beta_v} v_6$	$a_{16} \cdot v_6$	$a_{26} \cdot v_6$	$a_{36} \cdot v_6$	$a_{46} \cdot v_6$	$a_{56} \cdot v_6$	$a_{66} \cdot v_6$
$x_5 \xrightarrow{\beta_v} v_5$	$a_{15} \cdot v_5$	$a_{25} \cdot v_5$	$a_{35} \cdot v_5$	$a_{45} \cdot v_5$	$a_{55} \cdot v_5$	$a_{65} \cdot v_5$
$x_4 \xrightarrow{\beta_v} v_4$	$a_{14} \cdot v_4$	$a_{24} \cdot v_4$	$a_{34} \cdot v_4$	$a_{44} \cdot v_4$	$a_{54} \cdot v_4$	$a_{64} \cdot v_4$
$x_3 \xrightarrow{\beta_v} v_3$	$a_{13} \cdot v_3$	$a_{23} \cdot v_3$	$a_{33} \cdot v_3$	$a_{43} \cdot v_3$	$a_{53} \cdot v_3$	$a_{63} \cdot v_3$
$x_2 \xrightarrow{\beta_v} v_2$	$a_{12} \cdot v_2$	$a_{22} \cdot v_2$	$a_{32} \cdot v_2$	$a_{42} \cdot v_2$	$a_{52} \cdot v_2$	$a_{62} \cdot v_2$
$x_1 \xrightarrow{\beta_v} v_1$	$a_{11} \cdot v_1$	$a_{21} \cdot v_1$	$a_{31} \cdot v_1$	$a_{41} \cdot v_1$	$a_{51} \cdot v_1$	$a_{61} \cdot v_1$
Target	x_1	x_2	x_3	x_4	x_5	x_6

The embedding of token i is updated by taking a weighted sum of the value vectors v_j from all context tokens j, where the weights a_{ij} represent the attention from token i to token j. This weighted sum is then used to replace the representation of token i.

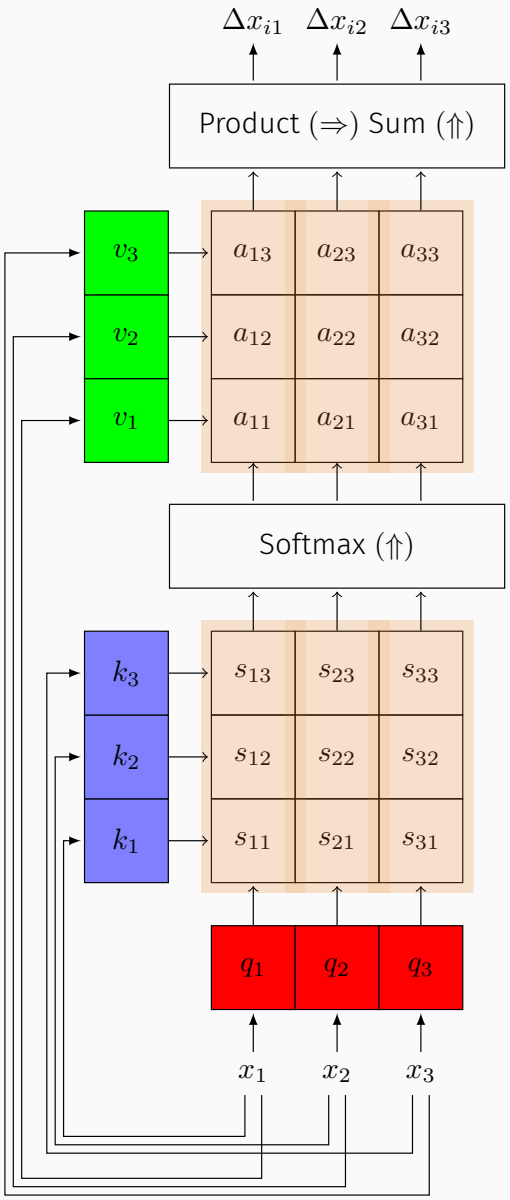
Since the operations are independent, attention weights can be computed efficiently using linear algebra [4]

$$C = \underbrace{softmax\overbrace{\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)}^{E} \cdot V}_{A}$$

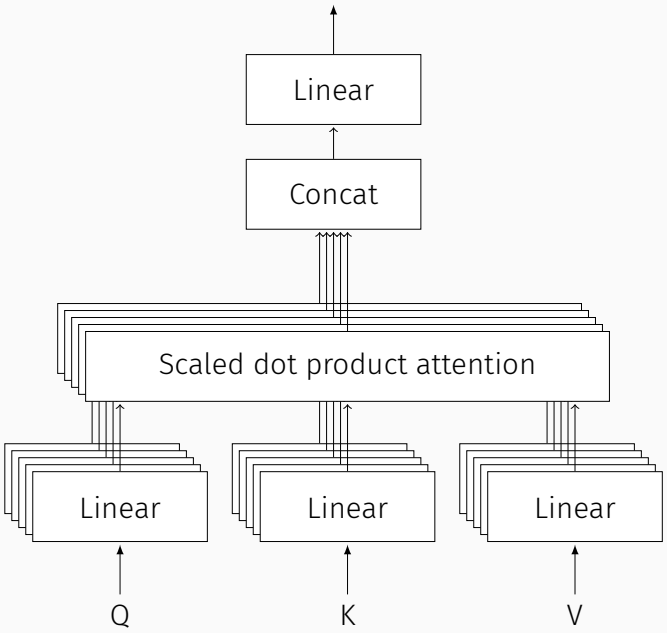
$$Q = X \cdot \beta_q \in \mathbb{R}^{n \times d_k}$$
$$K = X \cdot \beta_k \in \mathbb{R}^{n \times d_k}$$
$$V = X \cdot \beta_v \in \mathbb{R}^{n \times d_v}$$

The resulting context matrix $C \in \mathbb{R}^{n \times d_v}$. Typically $d_v = d_x$ but a projection may be used to match dimensions.

[4] The exponentiated dot product can produce large values → scaling proportionally to the embedding size (e.g. $\sqrt{d_k}$). The connectivity structure is dynamic since each observation has different attention weights.



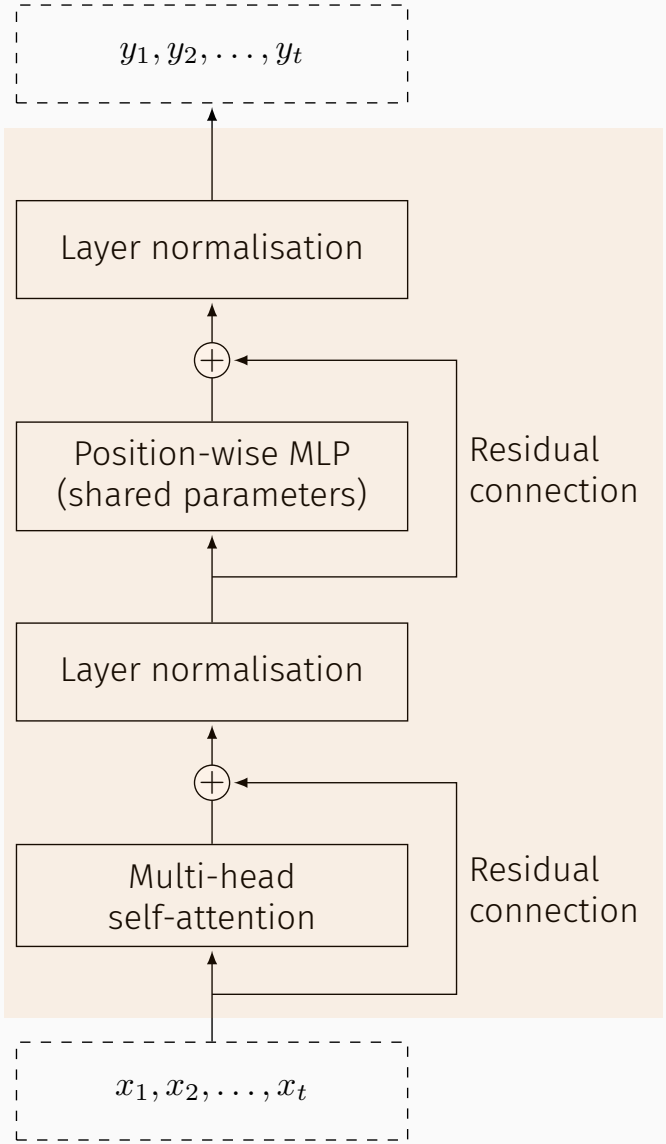
Note that $A \in \mathbb{R}^{n \times n}$ so the attention operation is quadratic in the sequence length.



Multi-head attention

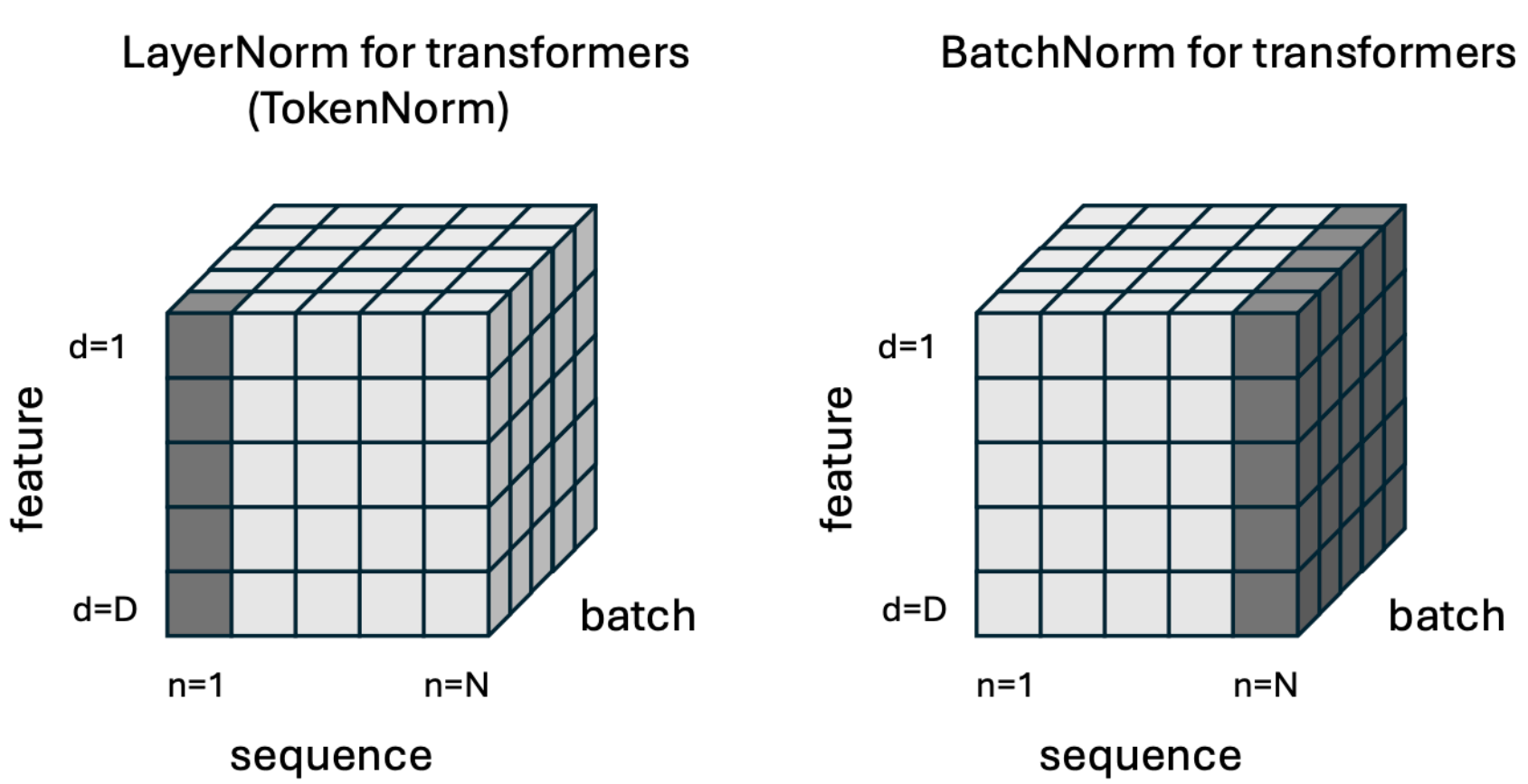
- Each head has it's own queries, keys and values of dimension $d_h = d/h$
- The output of the attention heads $\in \mathbb{R}n \times d_h \times t$ are concatenated along the feature dimension
- The linear layer combines projects the concatenated outputs to capture interactions between heads

Transformers

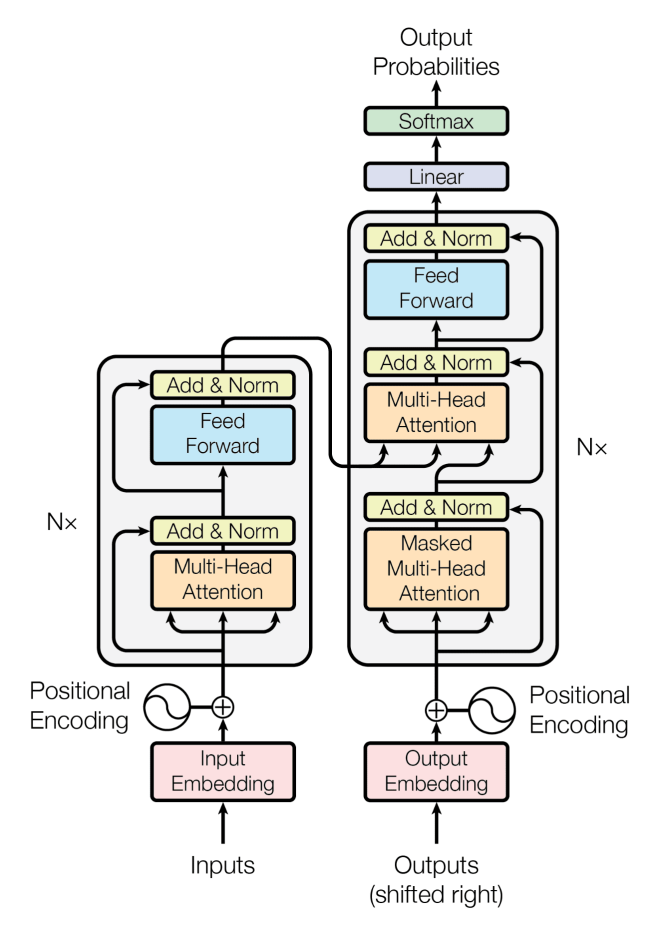


Transformer block

- The output of the attention heads are concatenated along the feature dimension
- The position-wise MLP introduces non-linearities, parameters are shared across time steps
- Normalisation is applied independently to each token embedding (i.e. layer)
- Residual connection enable more efficient optimisation

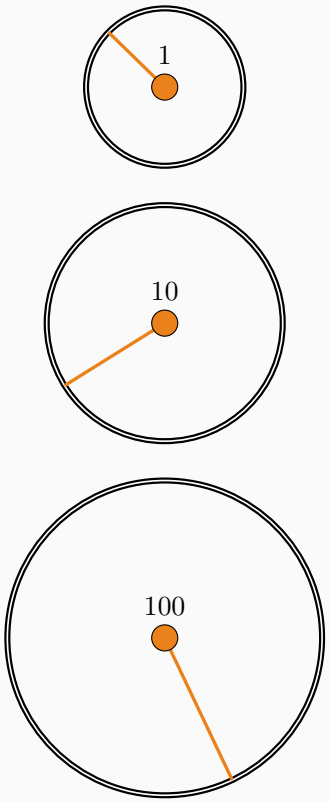

LayerNorm for transformers
(TokenNorm)
d=1
feature
d=D
batch
n=1
n=N
sequence
BatchNorm for transformers
d=1
feature
d=D
batch
n=1
n=N
sequence

Transformer architecture (Vaswani et al. 2017)

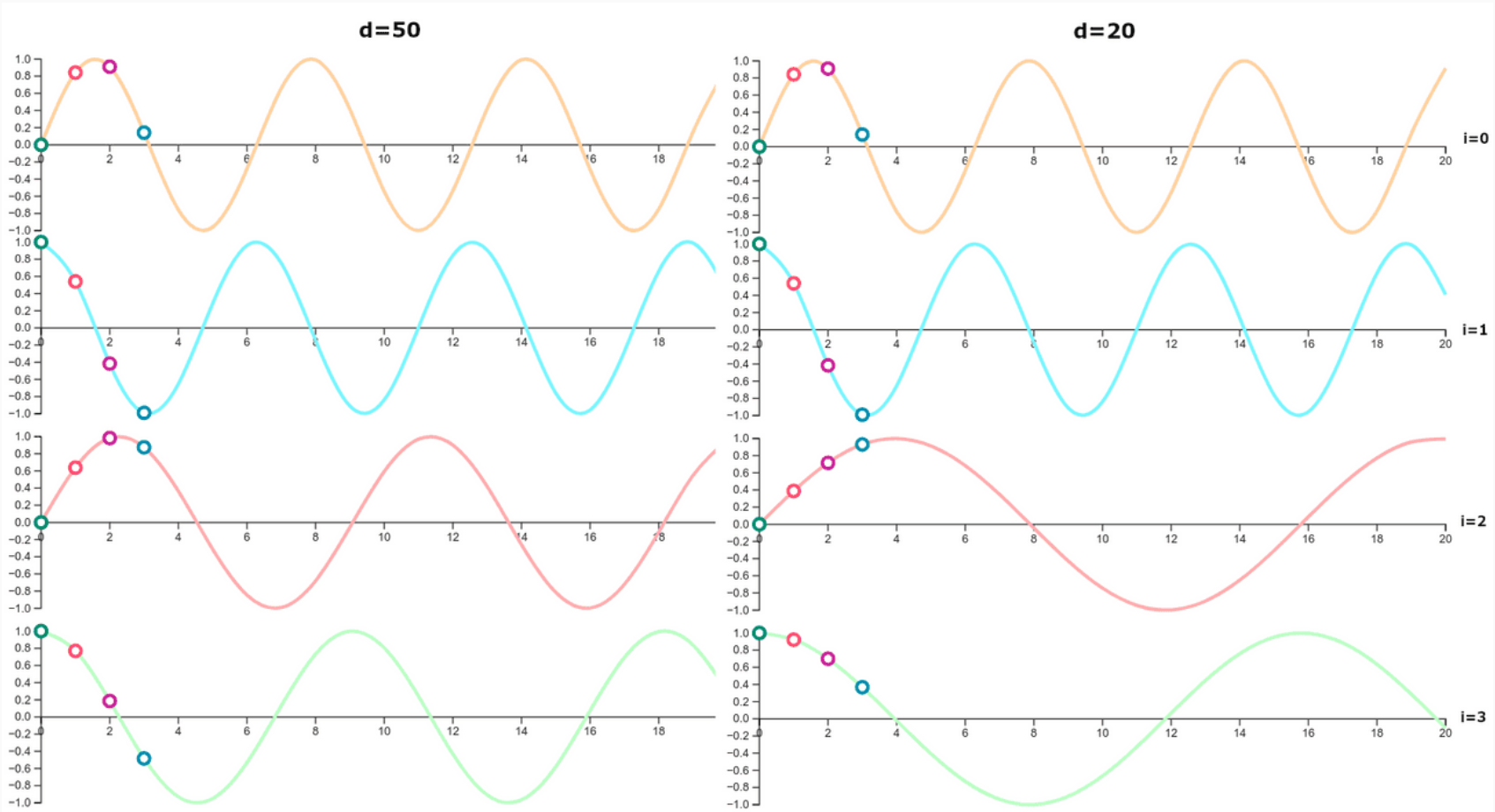


- Encoder-decoder architecture for machine translation
- Multi-head attention (MHA) uses 6 identical attention blocks
- Decoder masked MHA is used for auto-regressive text generation
- Decoder MHA uses queries from the decoder, and keys, values come from the encoder

The attention mechanism does explicitly encode the sequential structure of the data



- Represent token positions consistently across sequences of arbitrary length?
- Position encoding incorporate positional information into input tokens' embedding
- Sinusoidal encoding assigns each position in the sequence a unique sinusoidal pattern
- Position (embeddings) can also be learned using a dedicated embedding layer


d=50
d=20
i=0
i=1
i=2
i=3

Architectures

Caption generation (Xu et al. 2015)

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

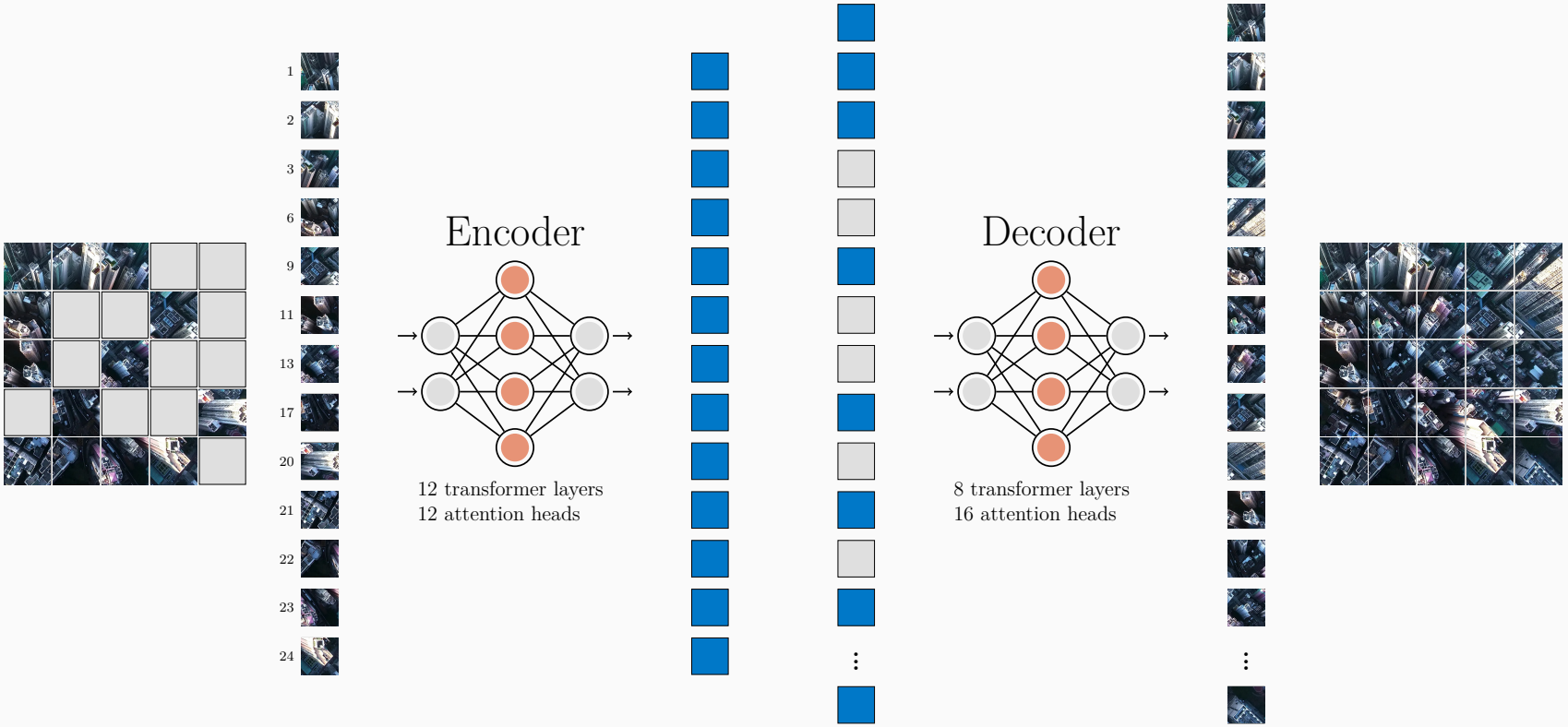
A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

ViT trained via masked auto-encoder



The encoder maps the unmasked tiles and their positions to feature representations that capture both their content and their relationships with neighbouring tiles. The decoder then uses these features to reconstruct the masked tiles. Each square represents a 768-dimensional vector. At inference time, the decoder can be discarded.

Summary

Summary

Large-scale training trumps inductive bias. With enough data, the model can learn to implicitly incorporate the necessary

- biases and structures, reducing the need for explicit inductive biases in the model architecture.
- In some cases, overly strong inductive biases may restrict the model's flexibility, limiting its ability to capture complex relationships present in the data.

Summary

	CNN	ViT
Minimal prior[5]	−	+
Global dependencies	−	+
Data efficiency	+	−
Compute efficiency	+	+
Generalisability	−	+
Multiple modalities	−	+

[5] Also called inductive biases. Convolution forces spatial locality and translation invariance. Self- attention is not limited to local interaction, but requires explicit spatial encoding.

Thank you for your attention!

References

References i

- Xu, Kelvin et al. (2015). **“Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”.** In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37, pp. 2048–2057 (cit. on p. 27).
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). ***Deep Learning*.** MIT Press.
- Vaswani, Ashish et al. (2017). **“Attention is all you need”.** In: *Advances in Neural Information Processing Systems*. Vol. 30 (cit. on pp. 3, 23).

References ii

- Caron, Mathilde et al. (2021). **“Emerging Properties in Self-Supervised Vision Transformers”.** In: *arXiv preprint arXiv:2104.14294* (cit. on p. 3).
- Dosovitskiy, Alexey et al. (2021). **“An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”.** In: *International Conference on Learning Representations (ICLR)* (cit. on p. 3).
- He, Kaiming et al. (2022). **“Masked Autoencoders Are Scalable Vision Learners”.** In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16000–16009 (cit. on p. 3).

References iii

- Khan, Salman et al. (2022). **“Transformers in Vision: A Survey”.** In: *ACM Computing Surveys* 54.10 (cit. on p. 3).