Attention-based architectures: Transformers

Generative and Deep Learning (GDL)

Davide Bacciu (davide.bacciu@unipi.it)

UNIVERSITÀ DI PISA

Objectives

A particular form of attention which changed the world

- Self-attention
- Transformer
- Inductive bias and gradient propagation
- Self-supervised training

Self-Attention

Why self-attention?

Limitations of paradigms encountered so far

- Convolutional: local receptive fields, struggle with long-range dependencies
- Recurrent: sequential, hard to parallelize, vanishing gradients

Our question for the day: how can a model

- Capture global dependencies?
- Be fully parallelizable?

What are we loosing when we gain the two?

Self-Attention

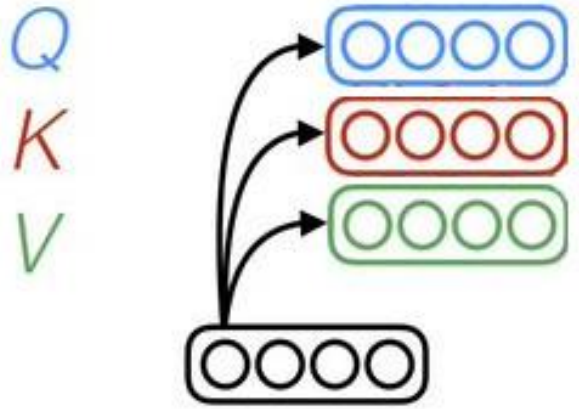
Intuition - Each element (token) in a sequence:

- Looks at all other elements
- Decides which are important

"The animal didn't cross the street because it was too tired" → "it" attends to "animal

How - Each element of an input sequence X_i projects into 3 vectors:

- **Query** Q_i
- **Key** K_i
- **Value** V_i



Self Attention – K,V,Q Generation

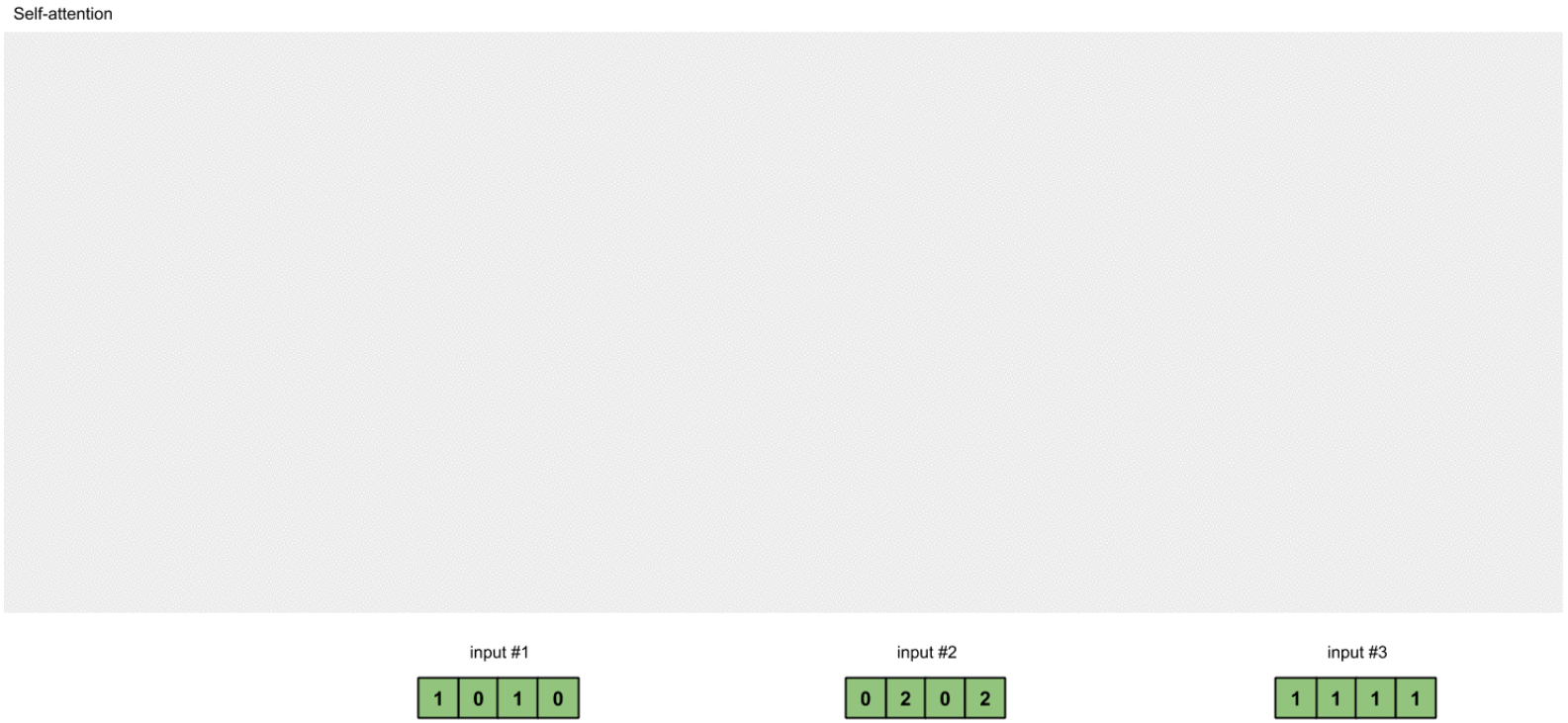
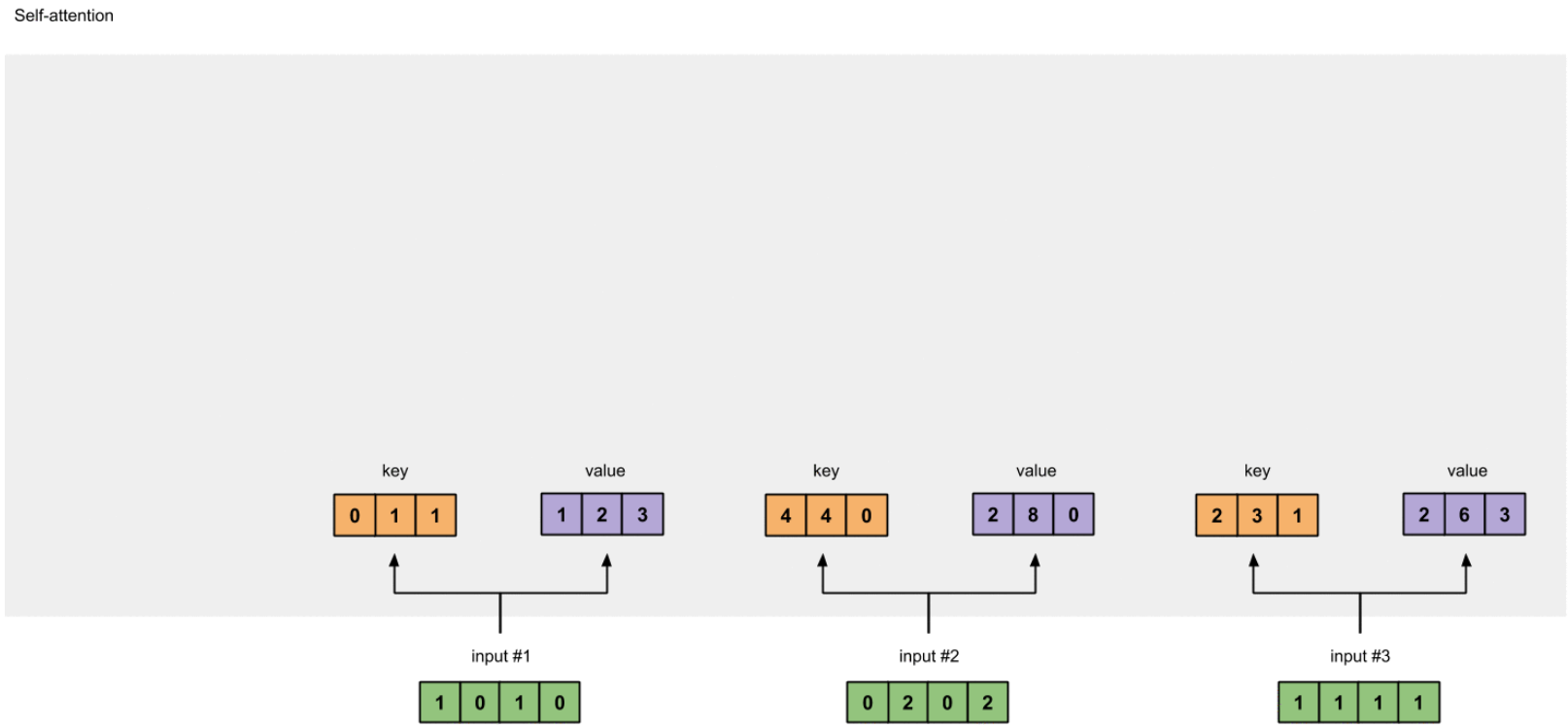
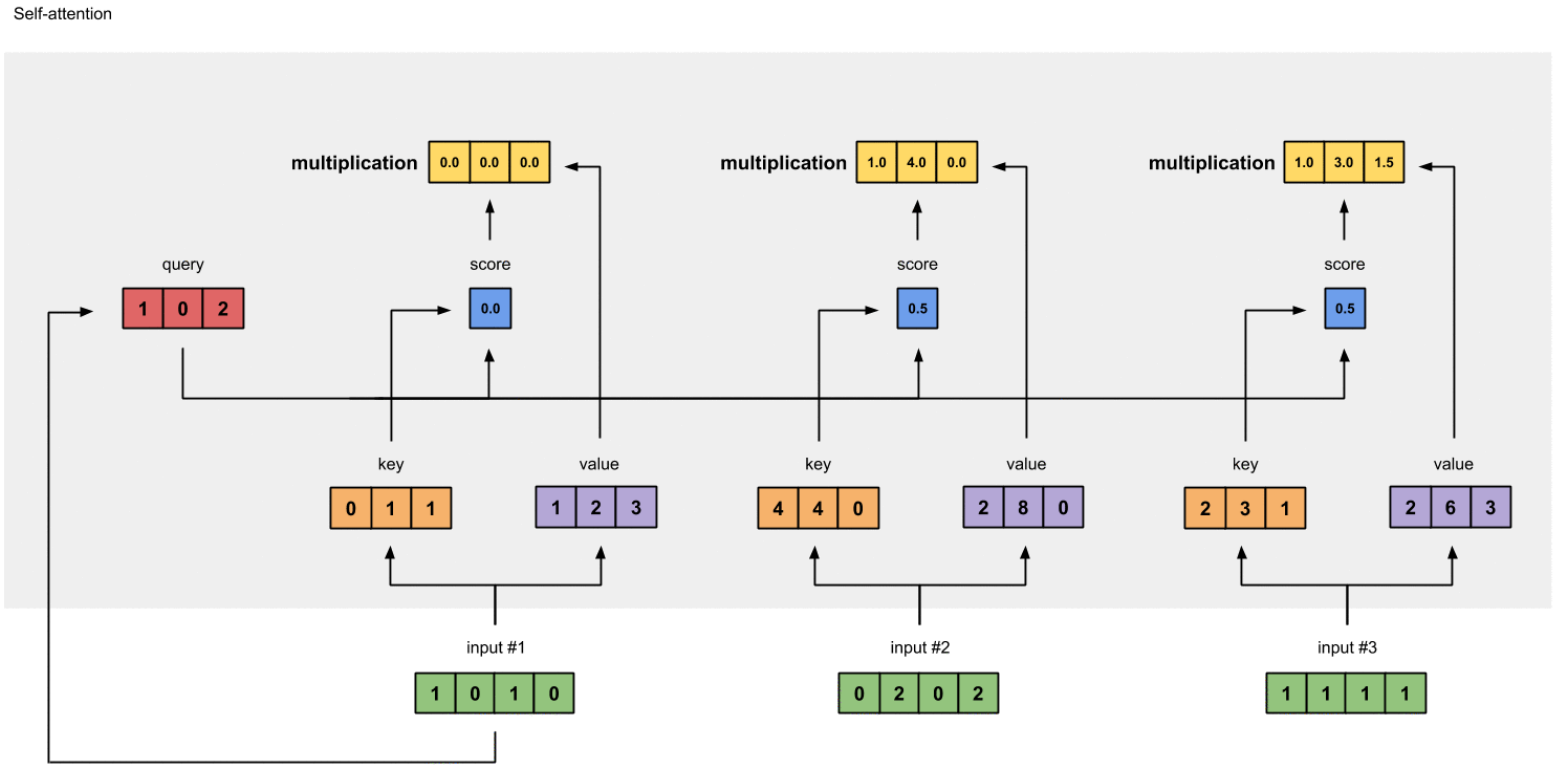


Figure credit to this article

Self Attention – Compute Attention Score



Self Attention – Produce Output



Self Attention - Formally

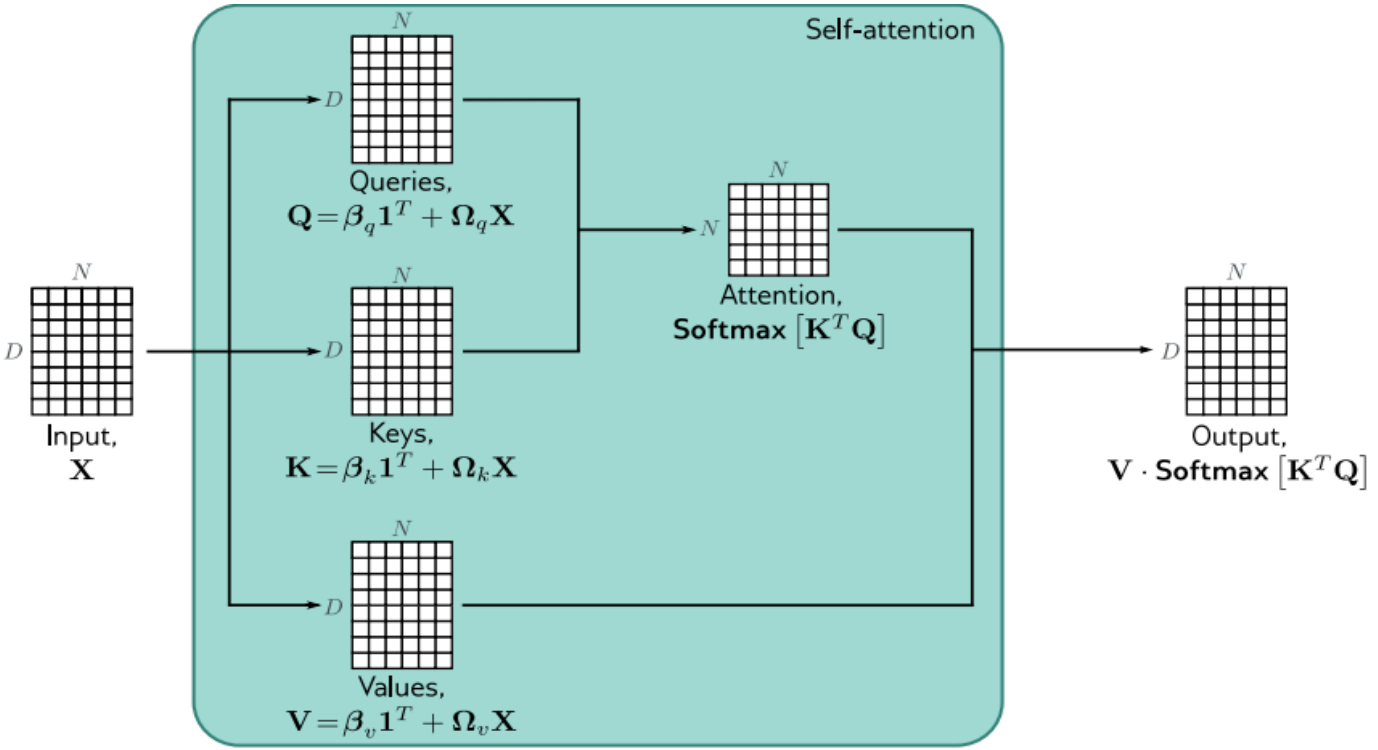
1) Project each input vector $X_i \in \mathbb{R}^D$

- $Q_i = \mathbf{\Omega}_q X_i$
- $K_i = \mathbf{\Omega}_k X_i$
- $V_i = \mathbf{\Omega}_v X_i$

Computed in parallel

2) Compute the scaled multiplicative self-attention embedding of i-th input

$$c_i = \sum_j softmax_j \left(\frac{Q_i \cdot \boldsymbol{K}^T}{\sqrt{d_k}} \right) V_j$$



Why separating key and query?

Because it allows for asymmetric queries

Example

Bank can query river, but river doesn't have to query bank

- Not all relevance relationships are symmetric

The scaling factor

$$\sum_j softmax_j\left(\frac{Q_i \cdot \boldsymbol{K}^T}{\sqrt{d_k}}\right) V_j$$

Assume Q_i, K_j have independent components with variance σ^2, then

$$Var(Q_i^T K_j) = d_k \sigma^4$$

- Dot products grow with dimension → softmax saturates.
- Scaling by $\sqrt{d_k}$ stabilizes gradients

Self Attention as.. any other attention

- Obtain the context info s (now query)

$$q_{\mathrm{i}} = \Omega_{\mathrm{q}} X_i$$

- Prepare the attention module input h (now key)

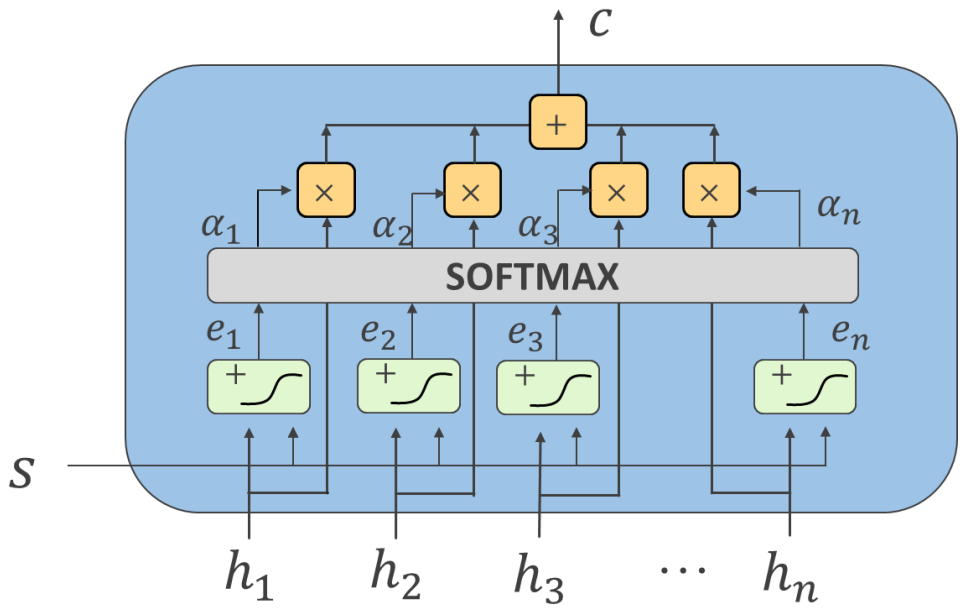
$$k_{\mathrm{j}} = \Omega_{\mathrm{k}} X_j$$

- Compute the similarity score:

$$e_{ij} = q_{\mathrm{i}}{}^{\mathrm{T}} k_{\mathrm{j}}$$

- Normalize into probabilities (the softmax)

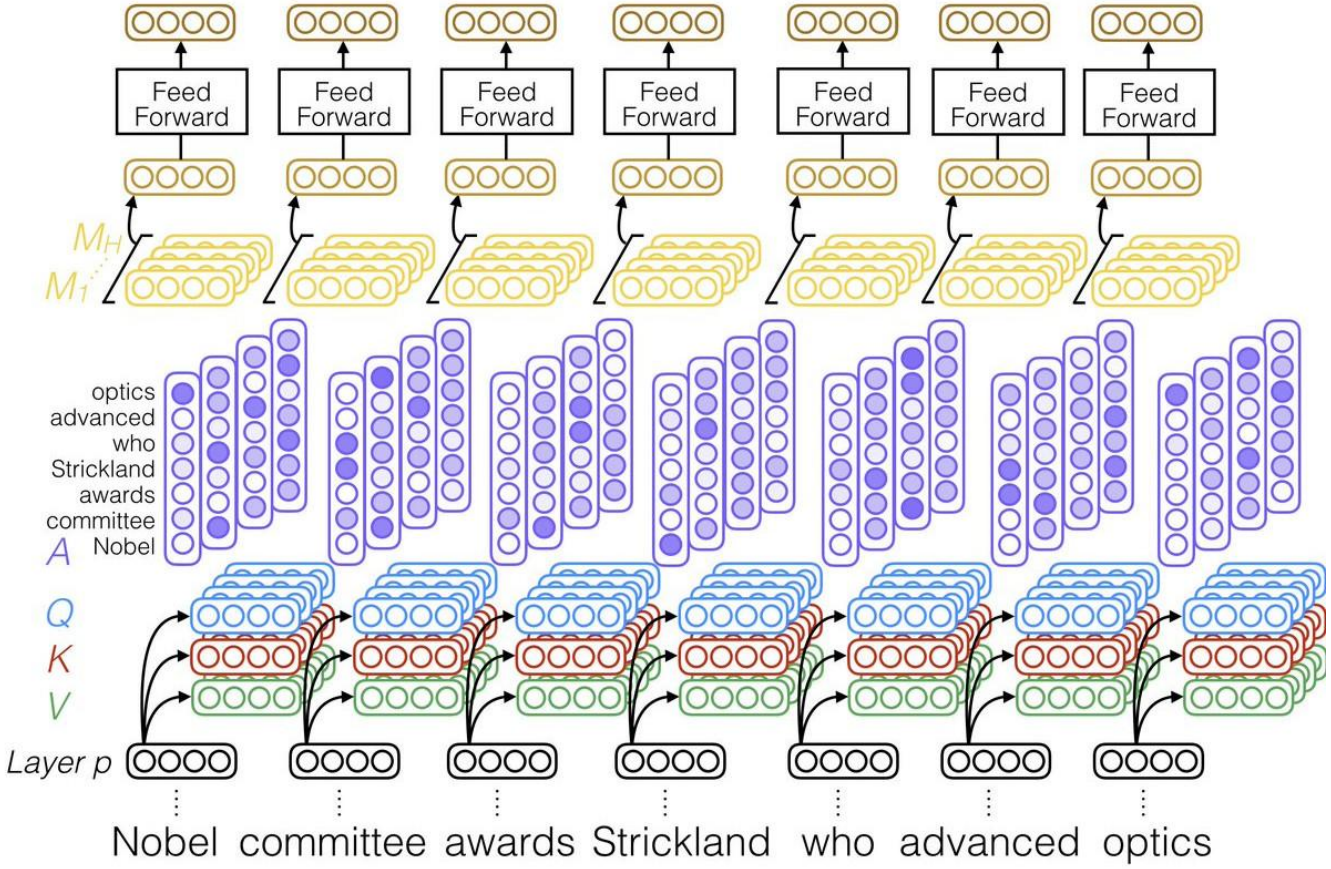
$$\alpha_{ij} = \frac{\exp\left(e_{ij}/\sqrt{d_k}\right)}{\sum_l \exp\left(e_{il}/\sqrt{d_k}\right)}$$

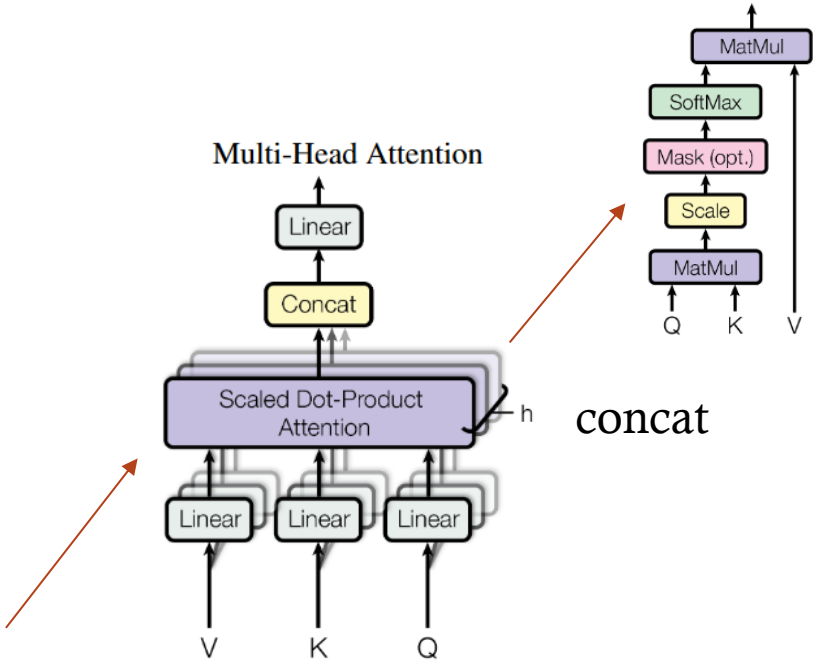


- Compute the output

$$c_i = \sum_j \alpha_{ij}\, v_j$$

Multihead attention





Strubell et al, Linguistically-Informed Self-Attention for Semantic Role Labeling, EMNLP 2018

Multihead attention

Split into h heads:

$$head_h = Attention(\boldsymbol{Q}\boldsymbol{\Omega}_q^h, \boldsymbol{K}\boldsymbol{\Omega}_k^h, \boldsymbol{V}\boldsymbol{\Omega}_v^h)$$

Then:

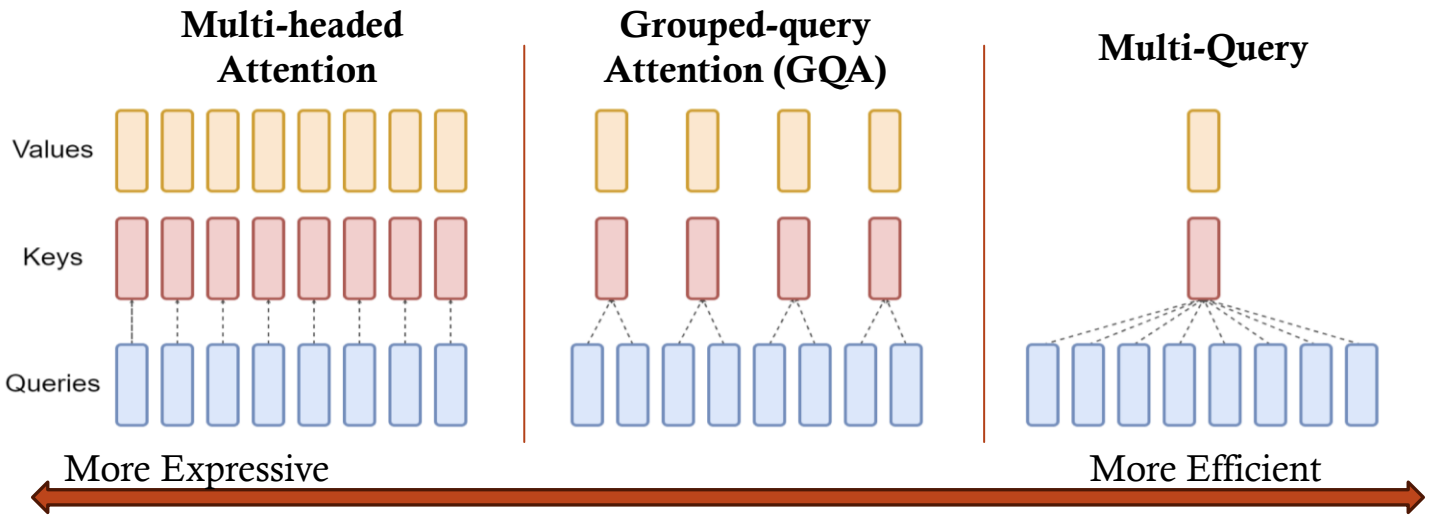
$$MultiHead = Concat(head^1, \dots, head_h)\boldsymbol{\Omega}_O$$

Interpretation:

- Each head learns a different subspace projection
- Equivalent to low-rank factorization of attention patterns

Forms of multi-headed attention

- Creating (and storing) keys $\mathbf{K}$, queries $\mathbf{Q}$, and values $\mathbf{V}$ for each head is costly
- Could we reuse the same keys and values with multiple queries?



Credit to GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

- Most models currently use Grouped Query Attention (GQA) which reuses keys and values across multiple query vectors

Characterizing Self-Attention

Is self-attention a good mechanism to model temporal dependencies?

What happens if I randomly shuffle some tokens?

Positional Encoding

- Self-attention is order-independent… but in sequences we need ordering information
- Modify the input encoding to contain information about the position of the token in the input
- We add a positional embedding $\boldsymbol{r}_n \in \mathbb{R}^D$ vector encoding the position to the initial input embedding $\mathbf{x}_n$ of each input token n

$$\tilde{\mathbf{x}}_n = \mathbf{x}_n + \mathbf{r}_n$$

Desiderata for a positional embedding

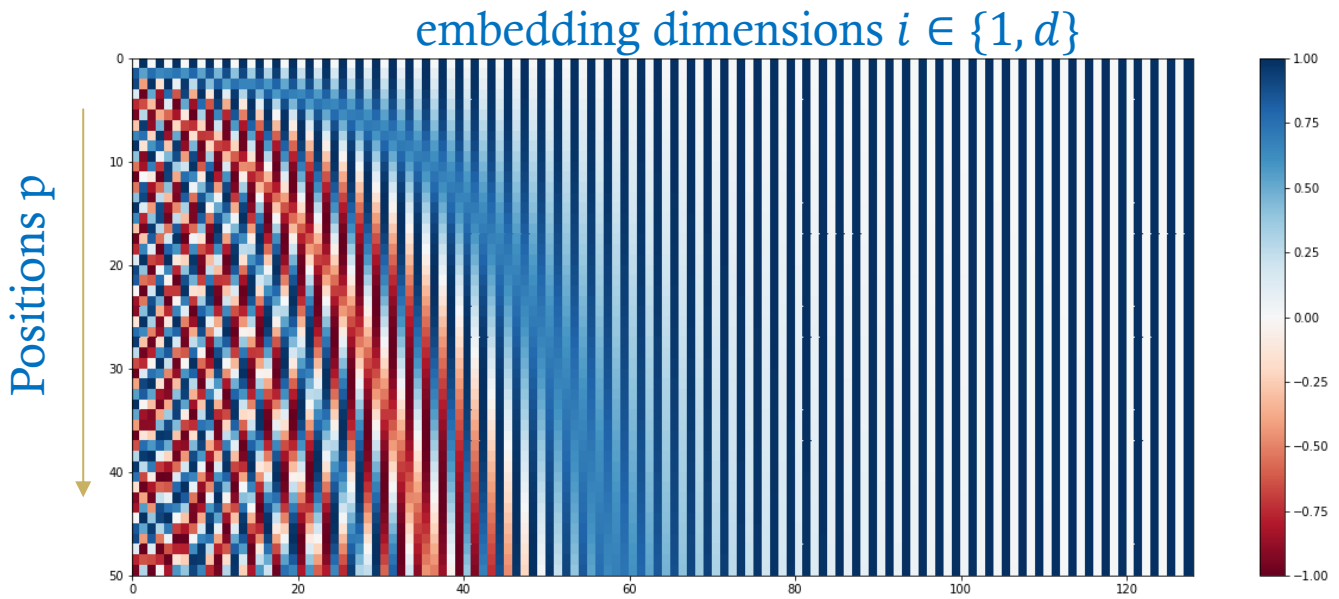
- Provide a unique representation $\mathbf{r}_n \in \mathbb{R}^D$ for each location.
- Be bounded on the scale of the original features $\mathbf{x}_n$ (e.g., [-1,1]).
  - Needed for stable training.
- Easily express relative distances between tokens
  - Inductive Bias: relative location is important for images and language
- Should work for arbitrary number of tokens N
  - Not all sentence or images are the same size.

Absolute Positional Encoding

Sinusoidal embeddings encoding positions with waves

frequency

$$PE(p, 2i) = \sin(p/10000^{2i/d})$$
$$PE(p, 2i+1) = cos(p/10000^{2i/d})$$



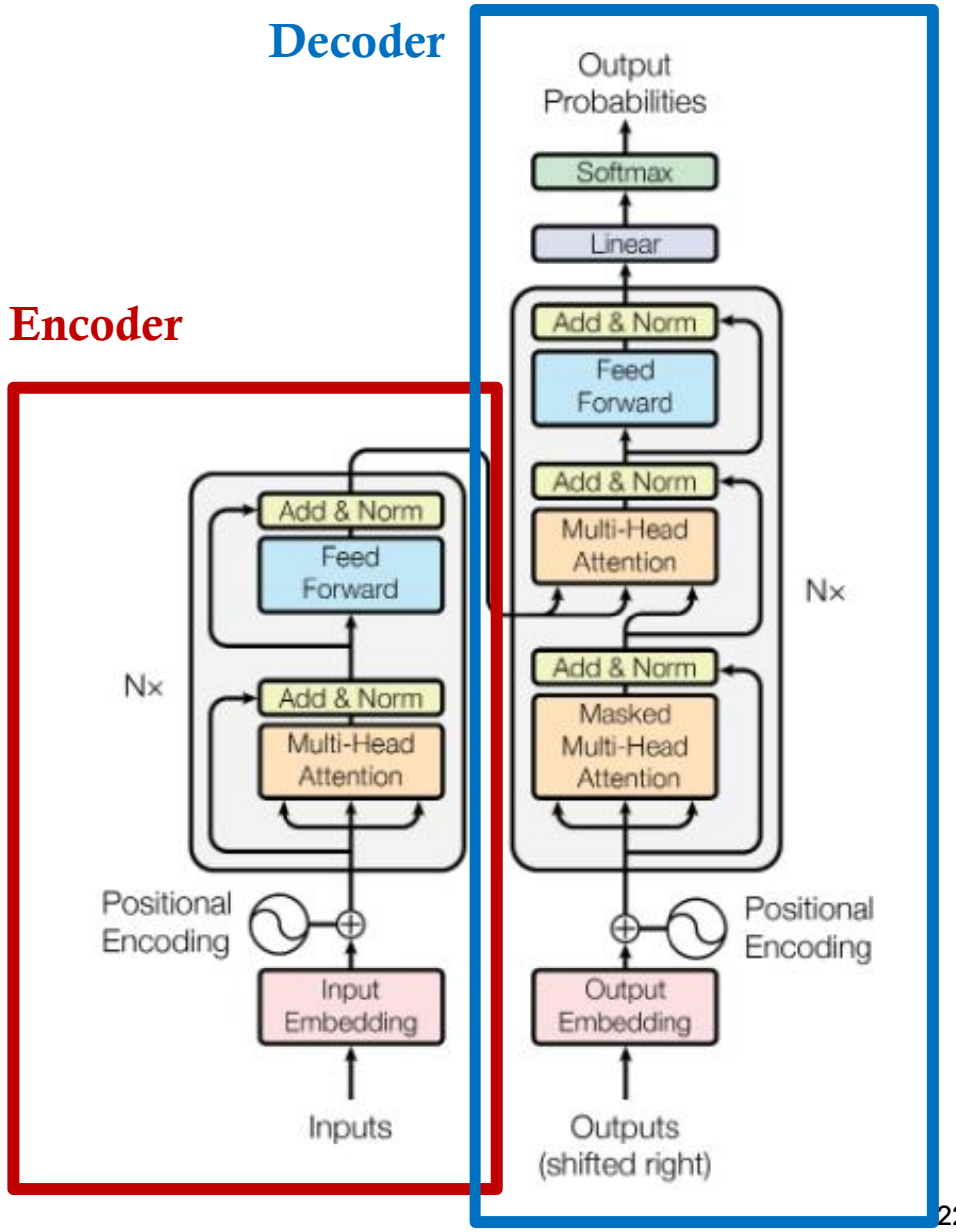
Like a binary encoding of the position but continuous

Attention Is All You Need, https://arxiv.org/pdf/1706.03762.pdf

Transformers

Transformers

- First pure attention-based model
- Heavy use of self-attention
- No recurrence (feedforward)
- Encoder-decoder architecture (at least originally)



Attention Is All You Need,
https://arxiv.org/pdf/1706.03762.pdf

Transformers – Layer-by-layer

Given input $\boldsymbol{X} \in \mathbb{R}^{NxD}$

- **Step 1**: Self-attention

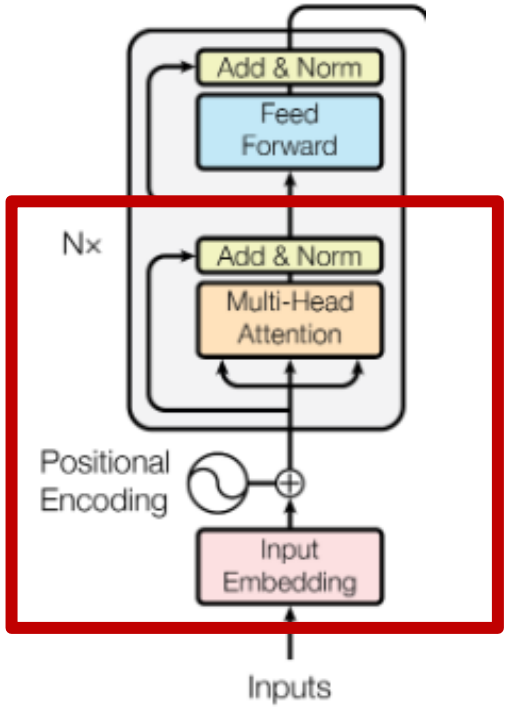
$$\boldsymbol{H} = MultiHead(\boldsymbol{X})$$

- **Step 2**: Residual connection + layer normalization

$$LayerNorm(\boldsymbol{X} + \boldsymbol{H})$$

$$LayerNorm(\boldsymbol{x}) = \frac{(\boldsymbol{x} - \boldsymbol{\mu})}{\sqrt{\boldsymbol{\sigma}^2 + \boldsymbol{\epsilon}}} * \gamma + \beta$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are computed per input



Transformers – Layer-by-layer

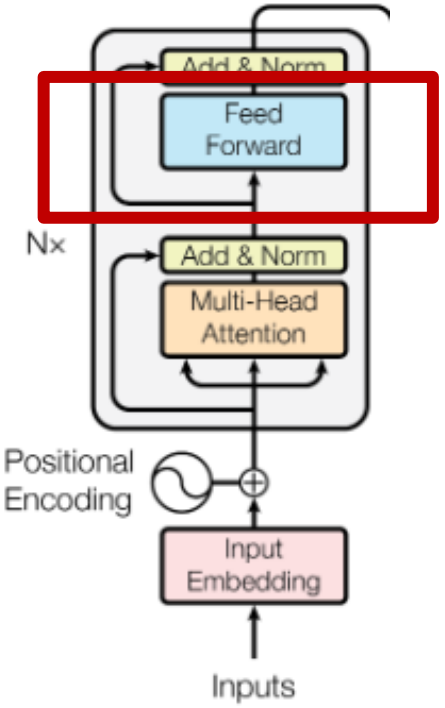
Step 3- Feedforward Network applied position-wise with ReLU (like) activation

$$FFN(x) = \max(0, \boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1)\, \boldsymbol{W}_2 + \boldsymbol{b}_2$$

Typically

- $\boldsymbol{W}_1$ expands dimension: D $\rightarrow$ 4D
- $\boldsymbol{W}_2$ projects back

Ultimately: $Z = FFN(\boldsymbol{H}')$



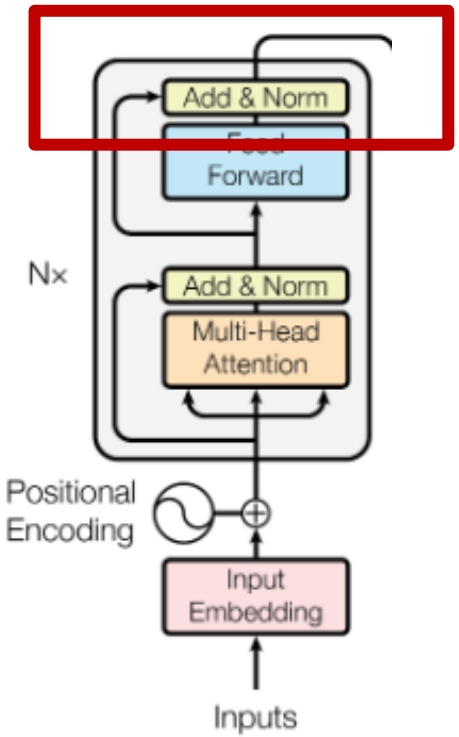
Transformers – Layer-by-layer

- **Step 4** - Final Output

$$\boldsymbol{Y} = LayerNorm(\boldsymbol{H}' + \boldsymbol{Z})$$

Thus each transformer layer computes

$$\boldsymbol{H}' = LN(\boldsymbol{X} + Attention(\boldsymbol{X}))$$
$$Y = LN\big(\boldsymbol{H}' + FFN(\boldsymbol{H}')\big)$$



The **Decoder** works similarly but uses masked attention to enforce causality

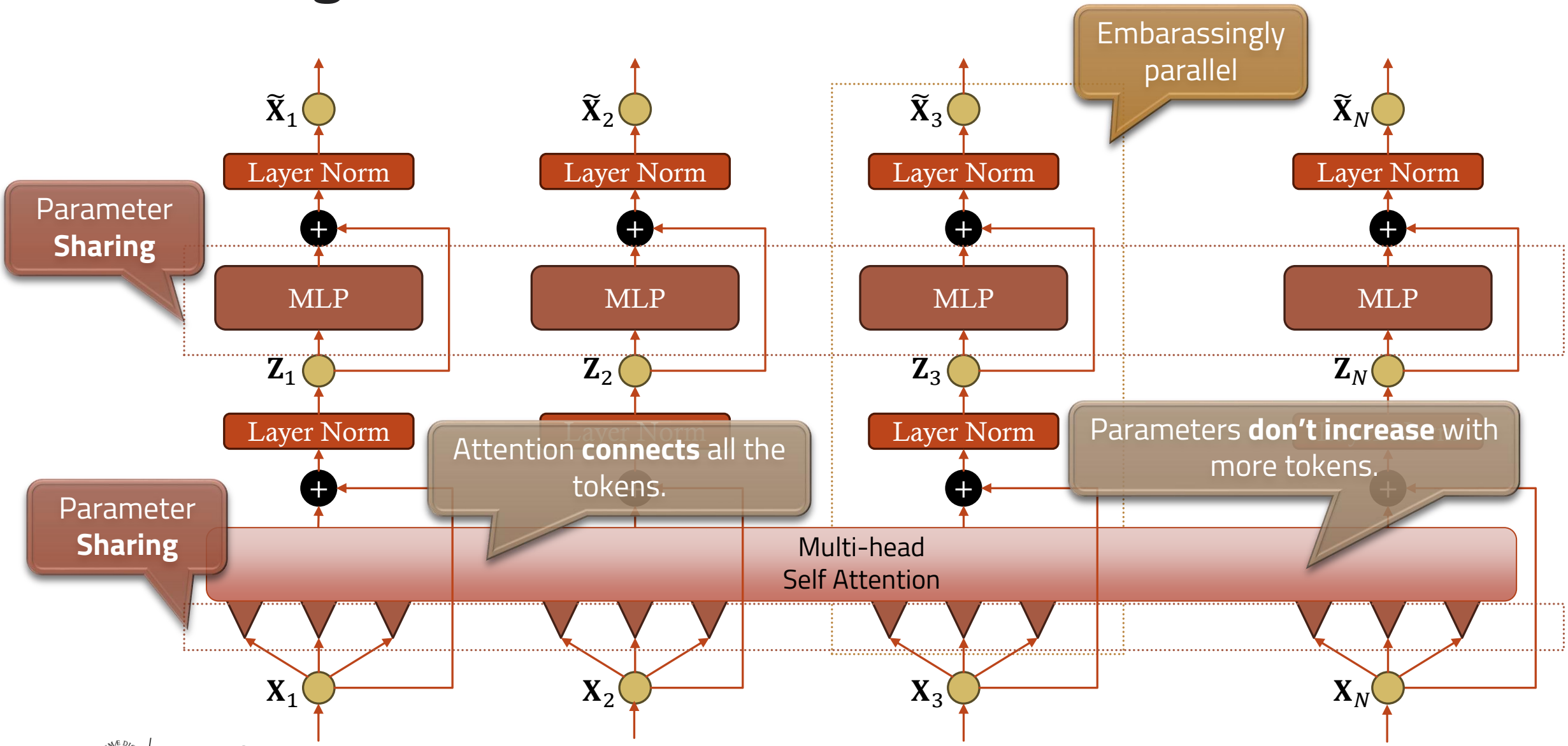
$$softmax\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T + \boldsymbol{M}}{\sqrt{d_k}}\right)$$

$$M_{ij} = -\infty \text{ if } j > i$$

Complexity and Gradient Paths

- Vanishing gradients depend on the depth of the (unrolled) network
- How much unrolling do we expect in a self-attention module?

Unrolling the Transformer



Complexity and Gradient Paths

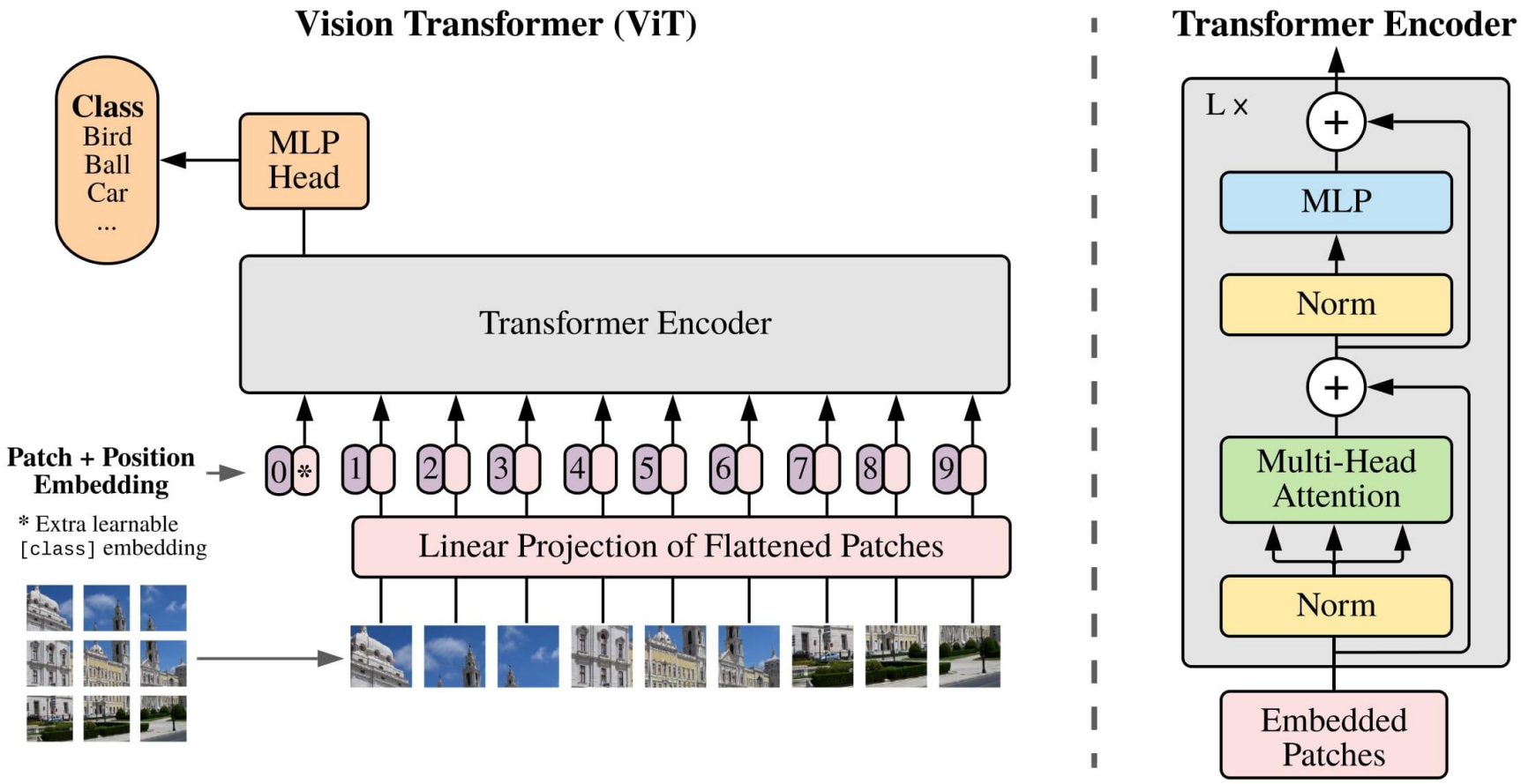
- Vanishing gradients depend on the depth of the (unrolled) network
- How much unrolling do we expect in a self-attention module?

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

The price is heavy complexity w.r.t. input length/size

The Vision Transformer (ViT)

A. Dosovitskiy et al, ICLR 2021

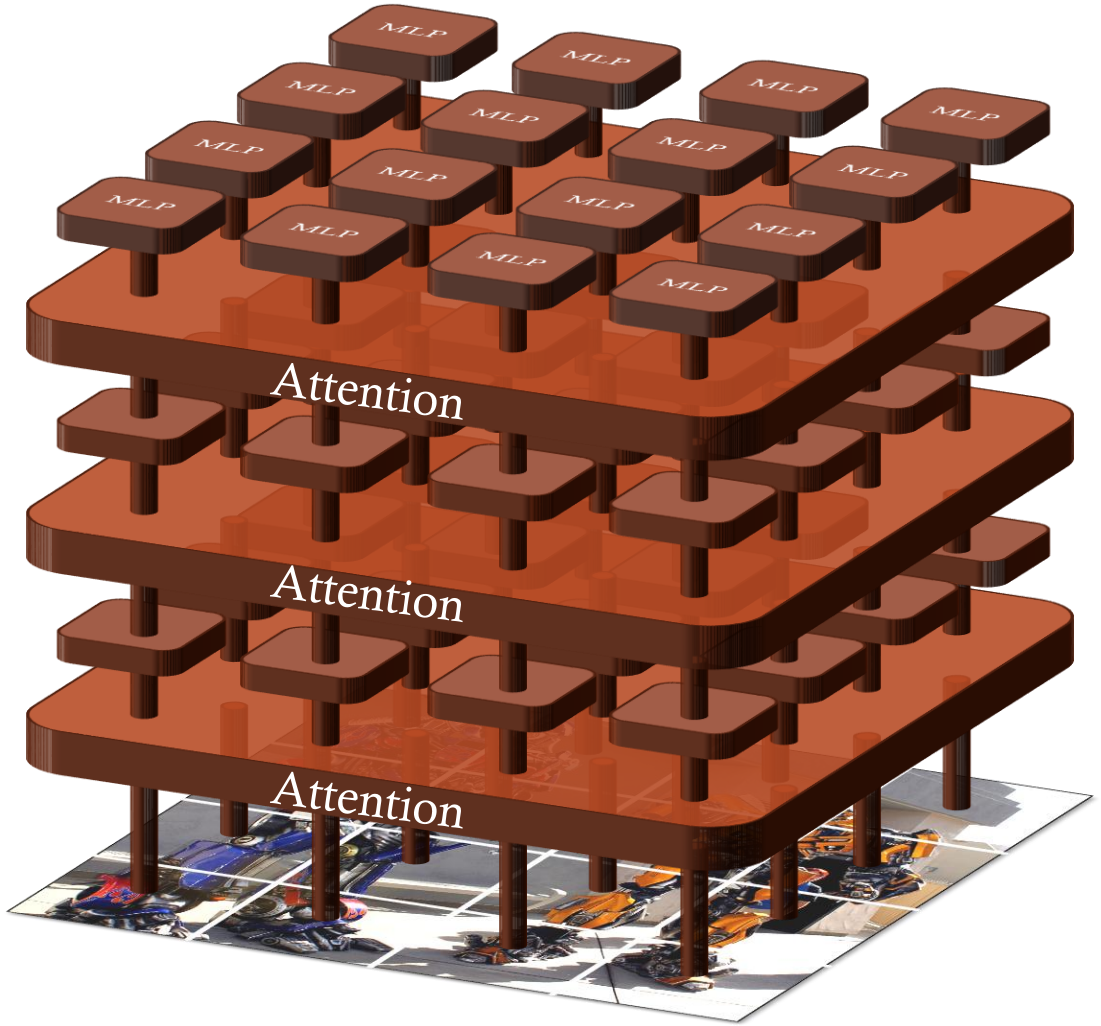


Slide credit to J.E. Gonzalez @ CS189 Berkeley

Reading the Output of the Transformer

Suppose we want to apply the vision transformer to image classification

- The top layer now has many MLP outputs
- How do we combine them?
  - Pooling + Linear Layer
  - Concatenation + Layer
- Common Trick:

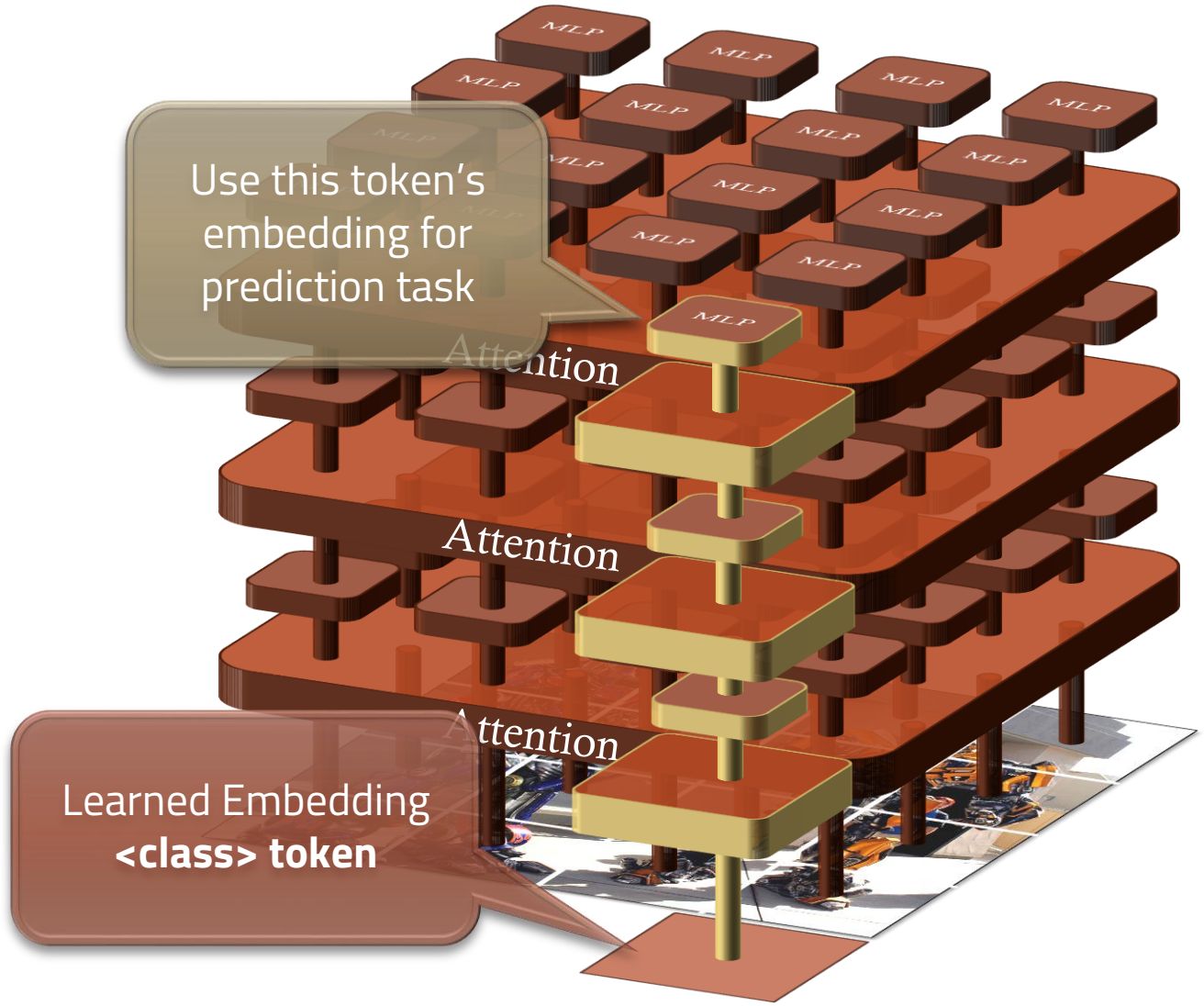


Slide credit to J.E. Gonzalez @ CS189 Berkeley

Reading the Output of the Transformer

Suppose we want to apply the vision transformer to image classification

- The top layer now has many MLP outputs
- How do we combine them?
  - Pooling + Linear Layer
  - Concatenation + Layer
- Common Trick:
  - Add another token



Learning with self-generated supervision

Self-generated supervision

Rationale

- Train a model to solve a pre-text task on unlabelled data
- Reuse and fine-tune the trained model on specialized tasks
- Self-generated refers to creating its own supervision

Two **main approaches**

- Self supervision
- Contrastive learning

Self-Supervised Learning

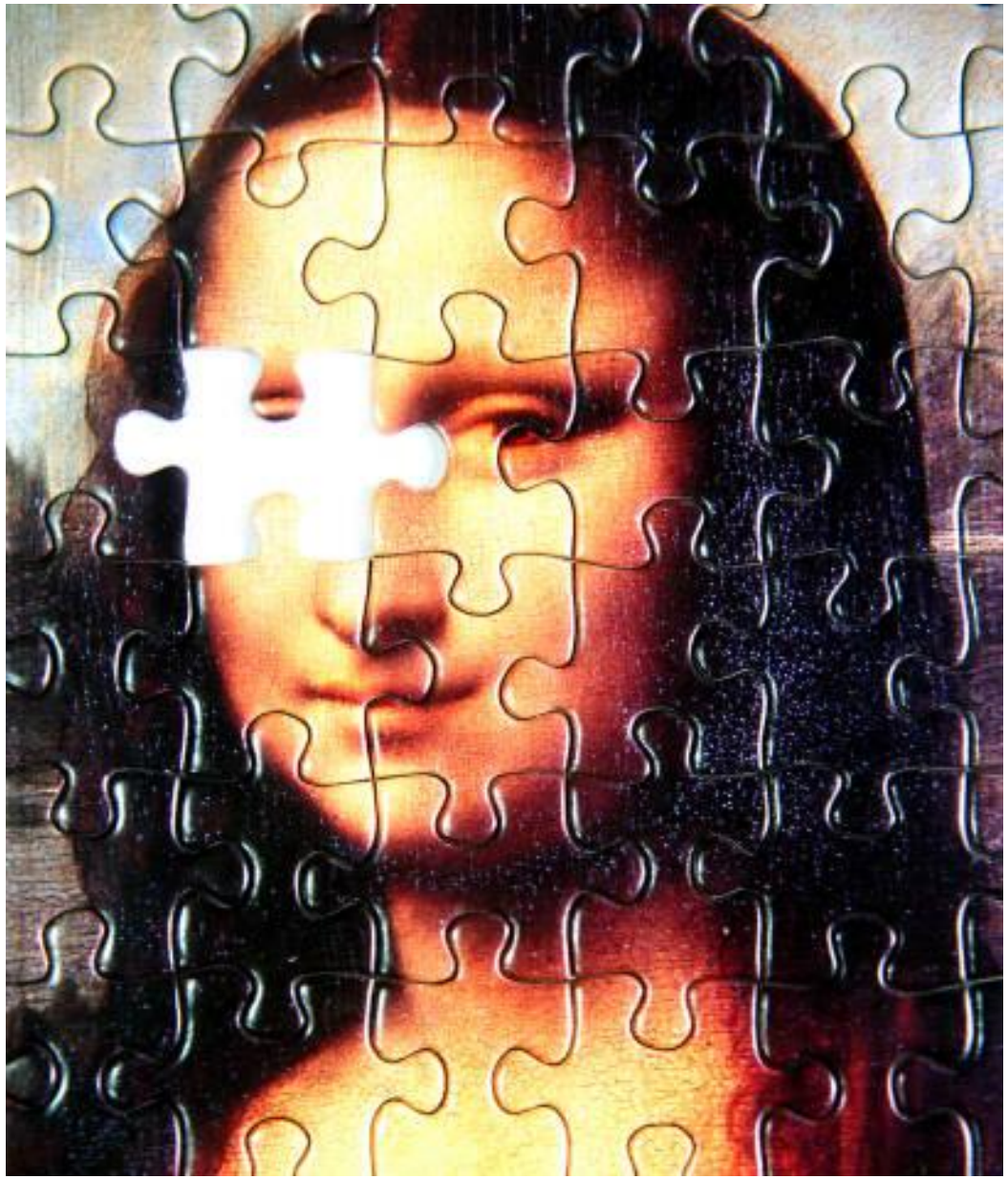
- Given unlabeled data $\boldsymbol{x} \sim p(\boldsymbol{x})$ construct objective

$$L(\theta) = \mathbb{E}_{\boldsymbol{x}} [\ell(f_\theta(\tilde{\boldsymbol{x}}), \boldsymbol{x})]$$

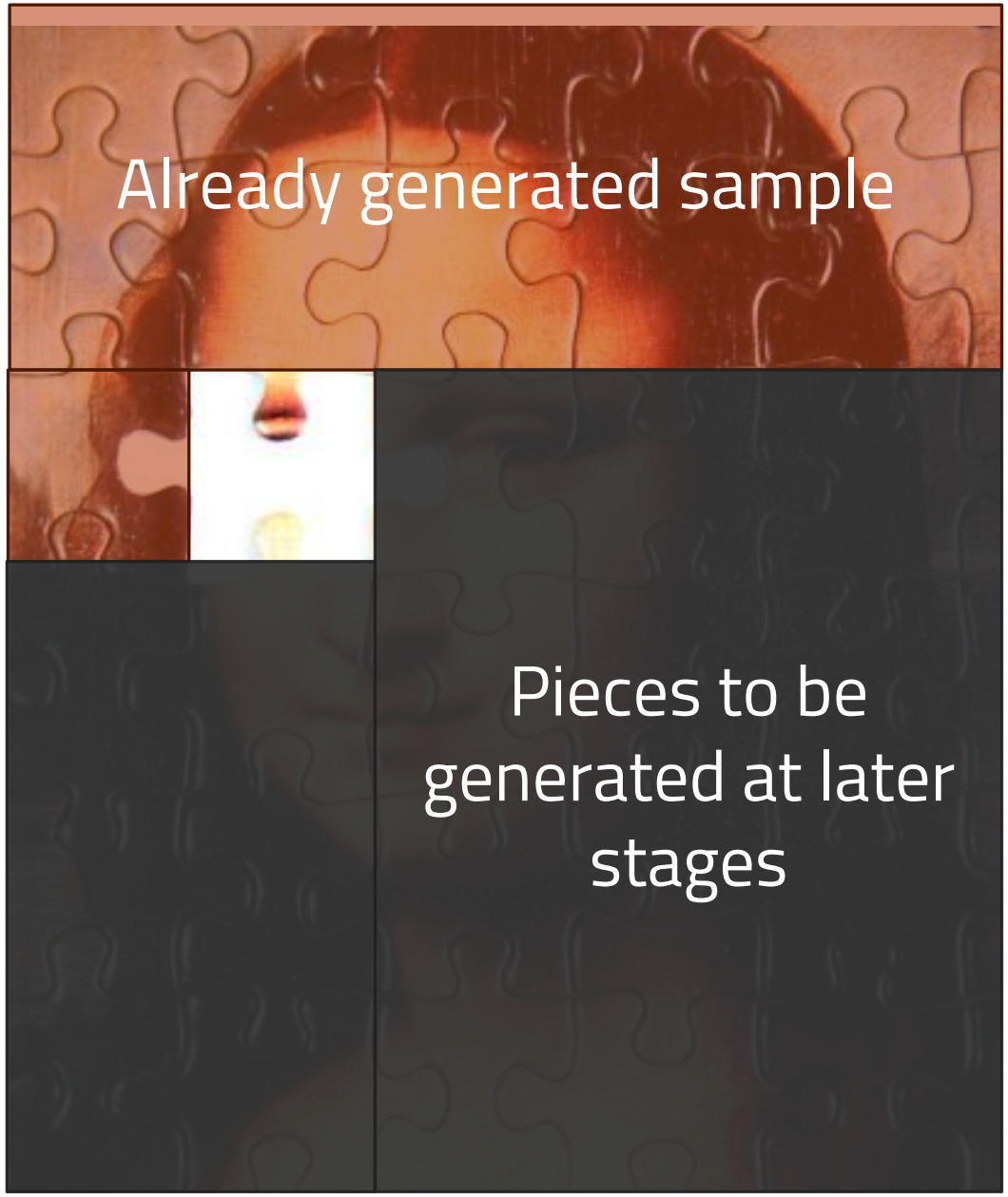
where $\tilde{\boldsymbol{x}}$ is corrupted version of input $\boldsymbol{x}$

- Sample corruption is such that it operates on only a limited portion of $\boldsymbol{x}$ in such a way that the unaltered part of $\tilde{\boldsymbol{x}}$ can be used to reconstruct the corrupted bit
- Most common approach uses masking
  - Generate a binary mask $\boldsymbol{M}_i$ (same size as the input), where i is the (single) position being masked
  - Mask the input $\tilde{\boldsymbol{x}} = \boldsymbol{x} \odot \boldsymbol{M}_i$, where $\odot$ is the elementwise product
  - Train the network to reconstruct $\boldsymbol{x}_i$ based on $\tilde{\boldsymbol{x}}$
  - Repeat sampling different positions i to be masked

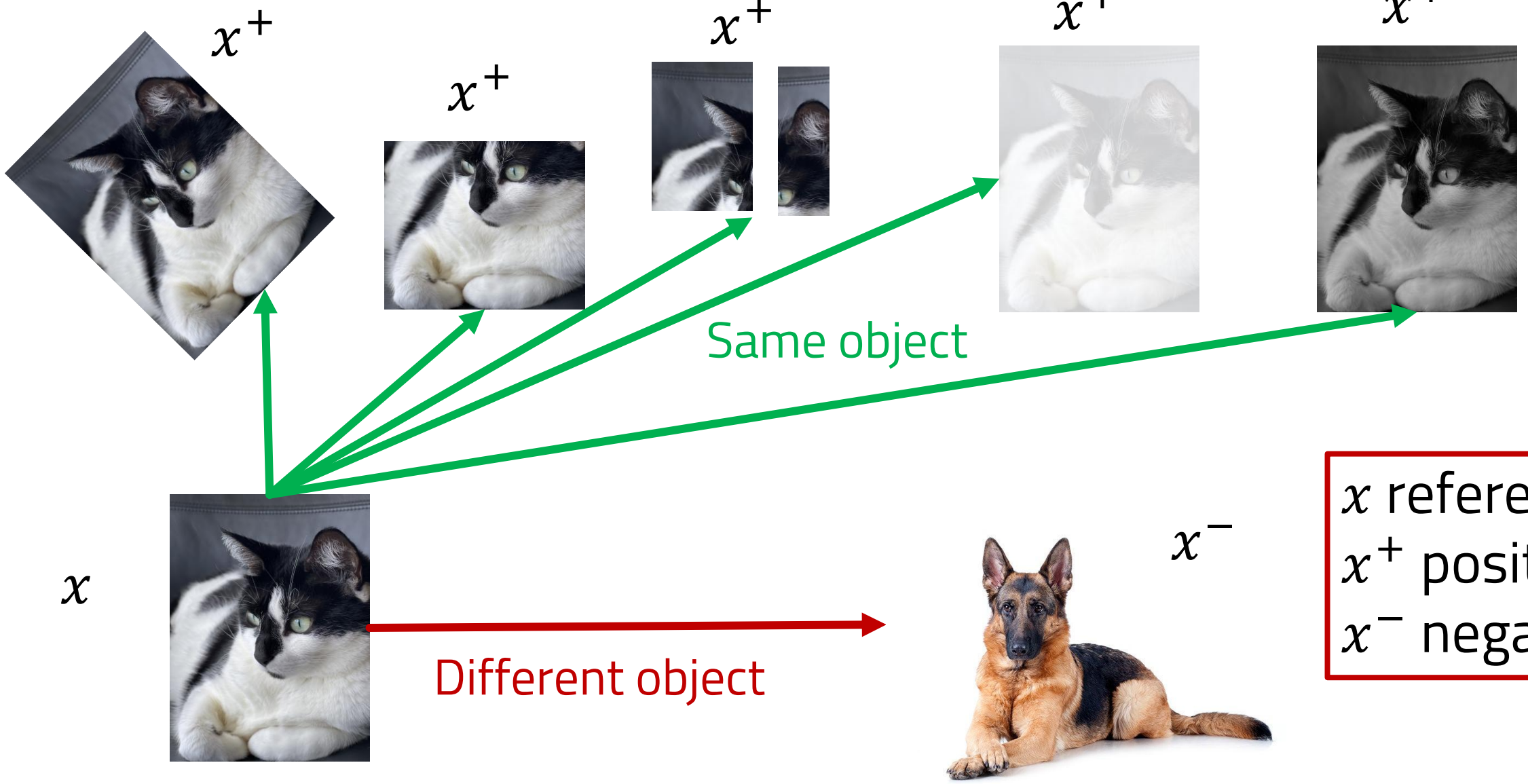
Autoregressive modelling and self-supervised learning



Autoregressive modelling and self-supervised learning



Contrastive learning principle



Contrastive learning

- Leverage weak similarity labelling to train a model to tell apart similar inputs w.r.t. dissimilar inputs
- Formally, we look for

$$sim\big(f_\theta(\boldsymbol{x}), f_\theta(\boldsymbol{x}^+)\big) \gg sim\big(f_\theta(\boldsymbol{x}), f_\theta(\boldsymbol{x}^-)\big)$$

  - $f_\theta(\boldsymbol{x})$ is some form of encoding obtained by a neural network $f_\theta(\cdot)$
  - $sim(\cdot,\cdot)$ is any reasonable similarity function (e.g. dot product)

- Rationale: given a chosen similarity/score function, we aim to learn an encoder function f_θ that yields high score for positive pairs $(\boldsymbol{x}, \boldsymbol{x}^+)$ and low scores for negative pairs $(\boldsymbol{x}, \boldsymbol{x}^-)$.

Contrastive learning by InfoNCE

- Given a neural network encoder $f_\theta(\boldsymbol{x})$ build the encoding $\mathbf{z} = f_\theta(\boldsymbol{x})$ and normalize it

$$\boldsymbol{z} = \boldsymbol{z}/\|\boldsymbol{z}\|$$

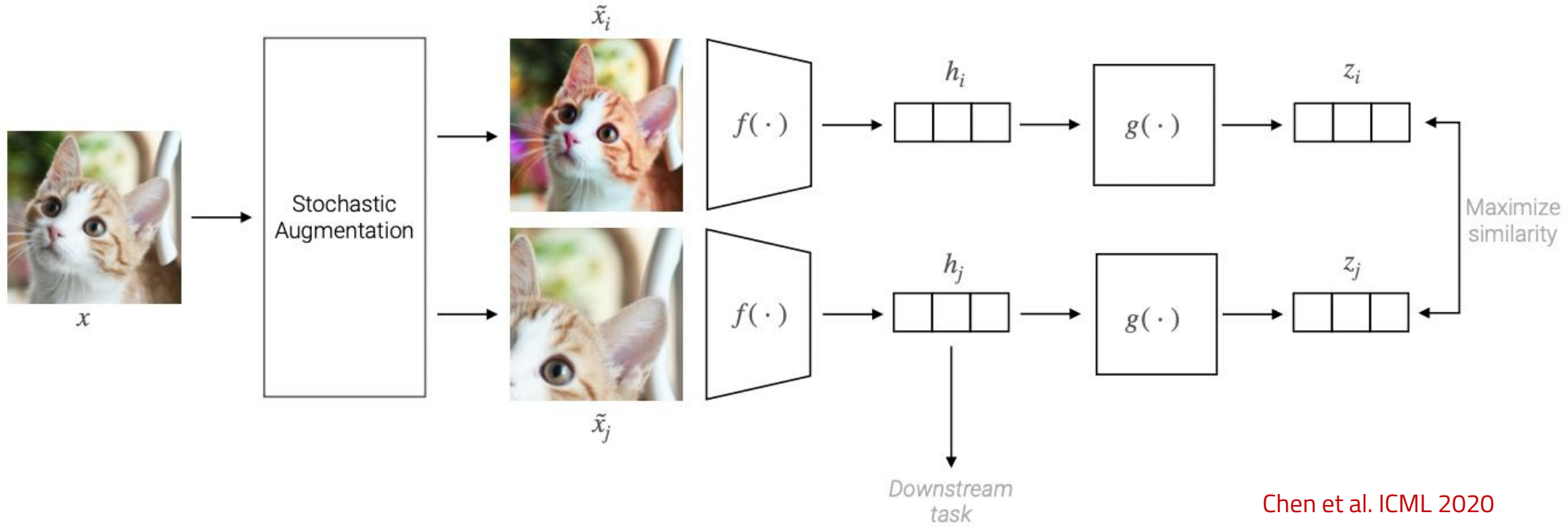
- Given a pair of similar samples $(\boldsymbol{x}, \boldsymbol{x}^+)$ minimize

$$L_{InfoNCE}(x) = -\log\left[\frac{\exp(sim(\boldsymbol{z}, \boldsymbol{z}^+)/\tau)}{\exp(sim(\boldsymbol{z}, \boldsymbol{z}^+)/\tau) + \sum_{\boldsymbol{x}^- \in Neg(\boldsymbol{x})} \exp(sim(\boldsymbol{z}, \boldsymbol{z}^-)/\tau)}\right]$$

- $Neg(\boldsymbol{x})$ is a set of samples that are dissimilar from $\boldsymbol{x}$, and τ is a temperature parameter
- Interpretation: softmax classification among negatives
  - Attraction term pulls positives together
  - Repulsion term pushes negatives apart

SimCLR Contrastive Learning Pipeline

Use parallel stochastic augmentation to produce positive pairs, use other samples as negatives



Chen et al. ICML 2020

Wrap-up

Take Home Messages

- Self-attention
  - Highly parallel neural mechanism to generate contextually informed embeddings of compound data
  - Ideal path length for gradient propagation.. at an N^2 cost
  - A layer for multiset data
- Transformers as low-inductive bias architectures
  - Need huge amounts of data to generalize
  - Self-attention-based: don't even have a native sense of ordering
- Self-supervised learning
  - Training on a pretext task involving correcting and/or understanding a data corruption
- Contrastive learning
  - Training on a pretext task involving separating alike from different data

Next Lectures

- Coding I: Pytorch
- Coding II : Keras/TF

- Guest lectures by Riccardo Massidda