



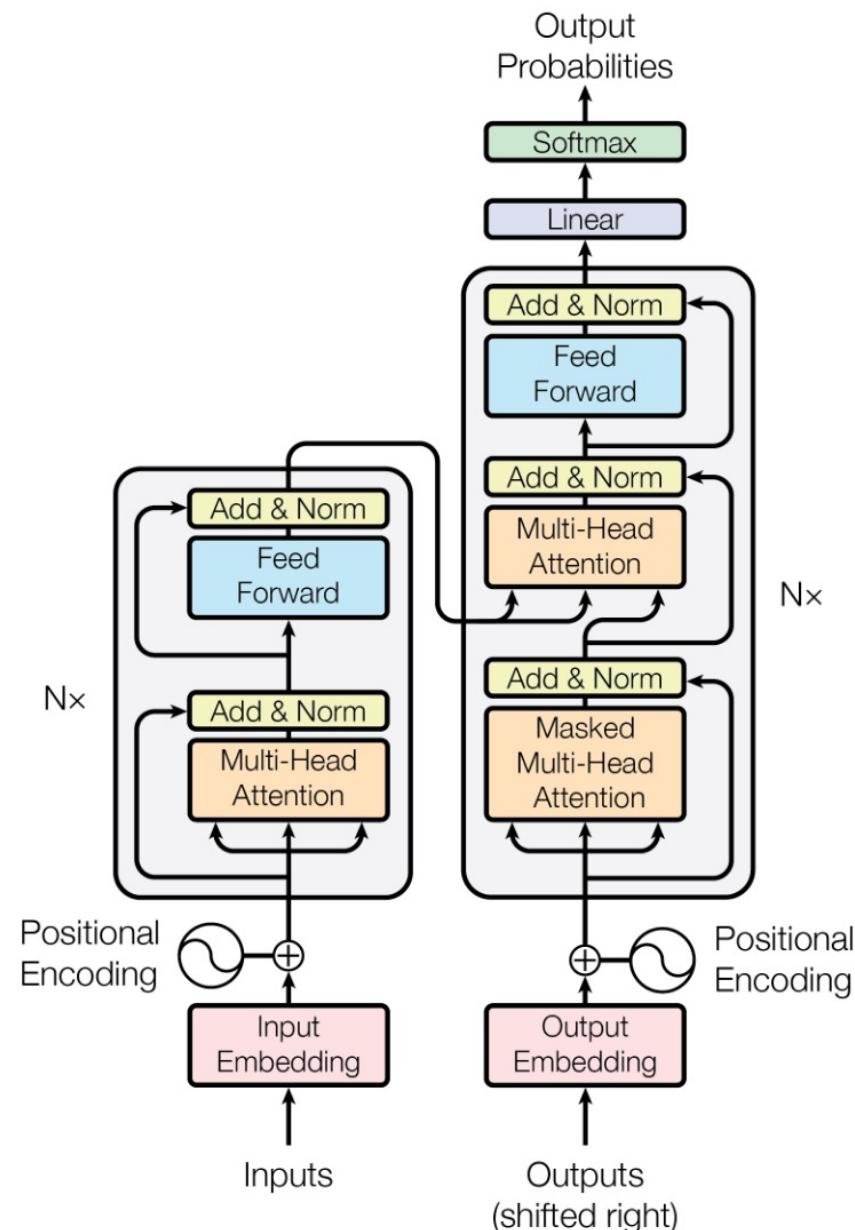
IT5005 Artificial Intelligence

Sirigina Rajendra Prasad
AY2025/2026: Semester 1

Transformers Contd.

Output Layer

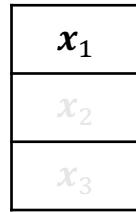
- **Inference:**
 - Auto-regressive and sequential over time
- At each step:
 - Input to the decoder is the tokens generated so far
 - Example: at time $t + 1$, input to decoder is $\hat{y}_{1:t}$
 - Decoder cross-attends to all encoder outputs
 - No masking at cross-attention layer
 - Q , K and V shapes grow with time
 - "KV caching" to save computation time
 - Query only for the newly added position



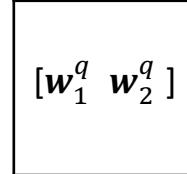
KV caching during inference

Inference: at $t = 1$

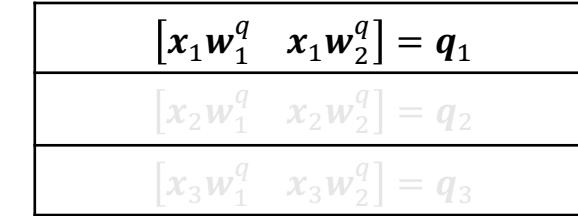
$$Q \in R^{1 \times 2}$$



$$W_i^Q \in R^{2 \times 2}$$



$$QW_i^Q = Q \in R^{1 \times 2}$$



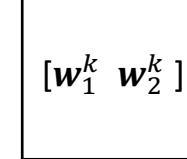
$$X \in R^{1 \times 2}$$



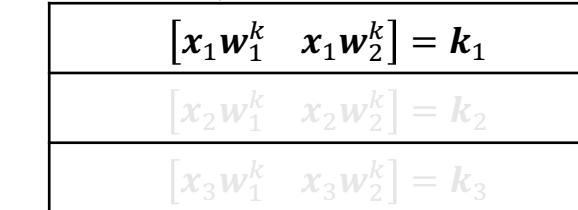
$$K \in R^{1 \times 2}$$



$$W_i^K \in R^{2 \times 2}$$



$$KW_i^K = K \in R^{1 \times 2}$$

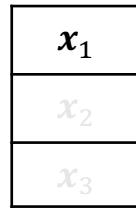


Transformatoren
sind
großartig

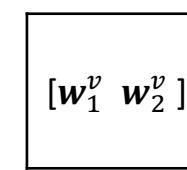
Raw Data

SE+PE

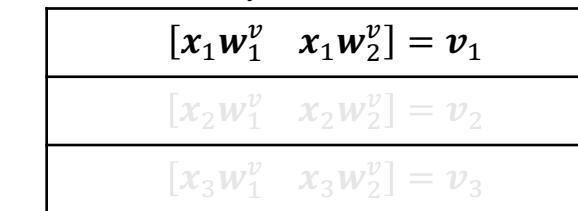
$$V \in R^{1 \times 2}$$



$$W_i^V \in R^{2 \times 2}$$



$$VW_i^V = V \in R^{1 \times 2}$$



$$head_i = softmax \left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}} \circ M \right) (VW_i^V)$$

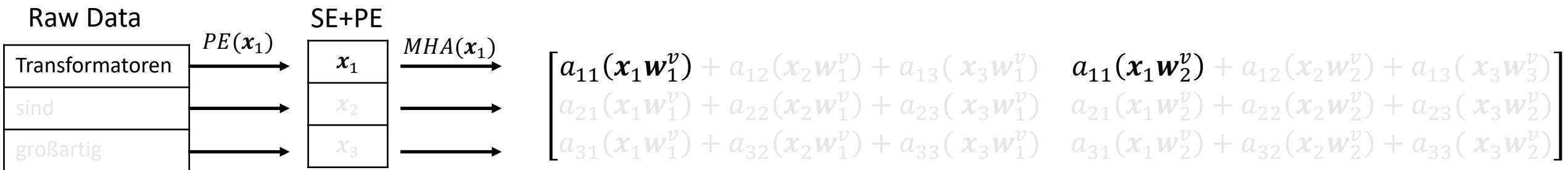
$$\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}} = \begin{bmatrix} \frac{\mathbf{q}_1 \mathbf{k}_1^T}{\sqrt{d_1}} & \frac{\mathbf{q}_1 \mathbf{k}_2^T}{\sqrt{d_1}} & \frac{\mathbf{q}_1 \mathbf{k}_3^T}{\sqrt{d_1}} \\ \frac{\mathbf{q}_2 \mathbf{k}_1^T}{\sqrt{d_2}} & \frac{\mathbf{q}_2 \mathbf{k}_2^T}{\sqrt{d_2}} & \frac{\mathbf{q}_2 \mathbf{k}_3^T}{\sqrt{d_2}} \\ \frac{\mathbf{q}_3 \mathbf{k}_1^T}{\sqrt{d_3}} & \frac{\mathbf{q}_3 \mathbf{k}_2^T}{\sqrt{d_3}} & \frac{\mathbf{q}_3 \mathbf{k}_3^T}{\sqrt{d_3}} \end{bmatrix}$$

| | | | |
|--------------|--------------|----------|----------|
| | Transformers | are | great |
| Transformers | a_{11} | a_{12} | a_{13} |
| are | a_{21} | a_{22} | a_{23} |
| great | a_{31} | a_{32} | a_{33} |

$$softmax \left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}} \circ M \right) = \begin{bmatrix} softmax \left(\begin{bmatrix} \frac{\mathbf{q}_1 \mathbf{k}_1^T}{\sqrt{d_1}} & -\infty & -\infty \end{bmatrix} \right) \\ softmax \left(\begin{bmatrix} \frac{\mathbf{q}_2 \mathbf{k}_1^T}{\sqrt{d_2}} & \frac{\mathbf{q}_2 \mathbf{k}_2^T}{\sqrt{d_2}} & -\infty \end{bmatrix} \right) \\ softmax \left(\begin{bmatrix} \frac{\mathbf{q}_3 \mathbf{k}_1^T}{\sqrt{d_3}} & \frac{\mathbf{q}_3 \mathbf{k}_2^T}{\sqrt{d_3}} & \frac{\mathbf{q}_3 \mathbf{k}_3^T}{\sqrt{d_3}} \end{bmatrix} \right) \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$head_i = softmax \left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}} \circ M \right) (VW_i^V)$$

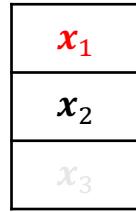
$$softmax \left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}} \circ M \right) = \begin{bmatrix} softmax \left(\begin{bmatrix} \frac{\mathbf{q}_1 \mathbf{k}_1^T}{\sqrt{d_1}} & -\infty & -\infty \end{bmatrix} \right) \\ softmax \left(\begin{bmatrix} \frac{\mathbf{q}_2 \mathbf{k}_1^T}{\sqrt{d_2}} & \frac{\mathbf{q}_2 \mathbf{k}_2^T}{\sqrt{d_2}} & -\infty \end{bmatrix} \right) \\ softmax \left(\begin{bmatrix} \frac{\mathbf{q}_3 \mathbf{k}_1^T}{\sqrt{d_3}} & \frac{\mathbf{q}_3 \mathbf{k}_2^T}{\sqrt{d_3}} & \frac{\mathbf{q}_3 \mathbf{k}_3^T}{\sqrt{d_3}} \end{bmatrix} \right) \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$



Pass this information down the layers and complete the prediction
 Use this predicted word as input for the next time instant

Inference: at $t = 2$

$$Q \in R^{3 \times 2}$$

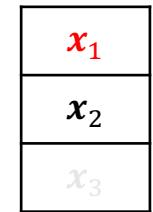


$$[\mathbf{w}_1^q \ \mathbf{w}_2^q]$$

| |
|---|
| $[x_1 \mathbf{w}_1^q \ x_1 \mathbf{w}_2^q] = q_1$ |
| $[x_2 \mathbf{w}_1^q \ x_2 \mathbf{w}_2^q] = q_2$ |
| $[x_3 \mathbf{w}_1^q \ x_3 \mathbf{w}_2^q] = q_3$ |

Cached
New Data

$$K \in R^{3 \times 2}$$



$$W_i^K \in R^{2 \times 2}$$

$$[\mathbf{w}_1^k \ \mathbf{w}_2^k]$$

$$KW_i^K = K \in R^{2 \times 2 \times 1}$$

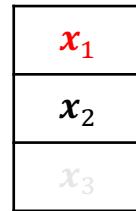
| |
|---|
| $[x_1 \mathbf{w}_1^k \ x_1 \mathbf{w}_2^k] = k_1$ |
| $[x_2 \mathbf{w}_1^k \ x_2 \mathbf{w}_2^k] = k_2$ |
| $[x_3 \mathbf{w}_1^k \ x_3 \mathbf{w}_2^k] = k_3$ |

| |
|-----------------|
| Transformatoren |
| sind |
| großartig |

| |
|-------|
| x_1 |
| x_2 |
| x_3 |

Raw Data

$$V \in R^{3 \times 2}$$



$$W_i^V \in R^{2 \times 2}$$

$$[\mathbf{w}_1^v \ \mathbf{w}_2^v]$$

$$VW_i^V = V \in R^{2 \times 2 \times 1}$$

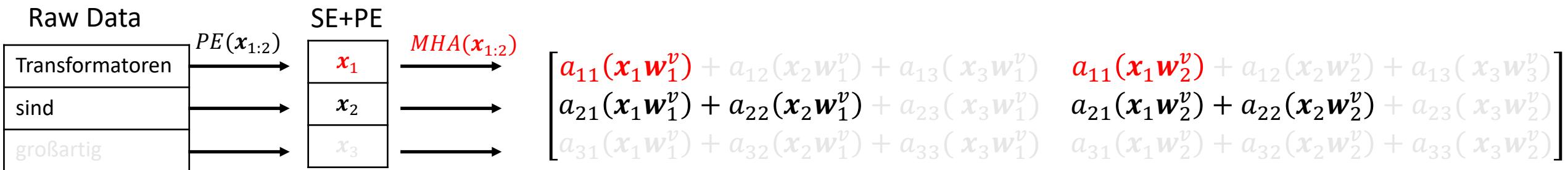
| |
|---|
| $[x_1 \mathbf{w}_1^v \ x_1 \mathbf{w}_2^v] = v_1$ |
| $[x_2 \mathbf{w}_1^v \ x_2 \mathbf{w}_2^v] = v_2$ |
| $[x_3 \mathbf{w}_1^v \ x_3 \mathbf{w}_2^v] = v_3$ |

Now the decoder sees a sequence of length 2

$$head_i = softmax \left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_K}} \circ M \right) (VW_i^V)$$

$$softmax \left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}} \circ M \right) \begin{bmatrix} \mathbf{x}_1 w_1^v & \mathbf{x}_1 w_2^v \\ \mathbf{x}_2 w_1^v & \mathbf{x}_2 w_2^v \\ \mathbf{x}_3 w_1^v & \mathbf{x}_3 w_2^v \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 w_1^v & \mathbf{x}_1 w_2^v \\ \mathbf{x}_2 w_1^v & \mathbf{x}_2 w_2^v \\ \mathbf{x}_3 w_1^v & \mathbf{x}_3 w_2^v \end{bmatrix}$$

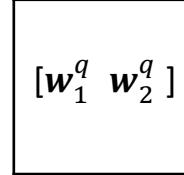
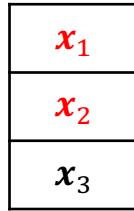
Cached
New Data



Pass this information down the layers and complete the prediction
 Use this predicted word as input for the next time instant

Why KV Caching

$$Q \in R^{3 \times 2}$$



| |
|-------------------------------------|
| $[x_1 w_1^q \quad x_1 w_2^q] = q_1$ |
| $[x_2 w_1^q \quad x_2 w_2^q] = q_2$ |
| $[x_3 w_1^q \quad x_3 w_2^q] = q_3$ |

Cached
New Data

$$X \in R^{3 \times 2}$$

$$K \in R^{3 \times 2}$$

$$W_i^K \in R^{2 \times 2}$$

$$KW_i^K = K \in R^{3 \times 2 \times 1}$$

| |
|-------------------------------------|
| $[x_1 w_1^k \quad x_1 w_2^k] = k_1$ |
| $[x_2 w_1^k \quad x_2 w_2^k] = k_2$ |
| $[x_3 w_1^k \quad x_3 w_2^k] = k_3$ |

Transformatoren
sind
großartig

SE+PE

x_1
 x_2
 x_3

w_1^k
 w_2^k

| |
|-------------------------------------|
| $[x_1 w_1^k \quad x_1 w_2^k] = k_1$ |
| $[x_2 w_1^k \quad x_2 w_2^k] = k_2$ |
| $[x_3 w_1^k \quad x_3 w_2^k] = k_3$ |

Raw Data

SE+PE

x_1
 x_2
 x_3

w_1^v
 w_2^v

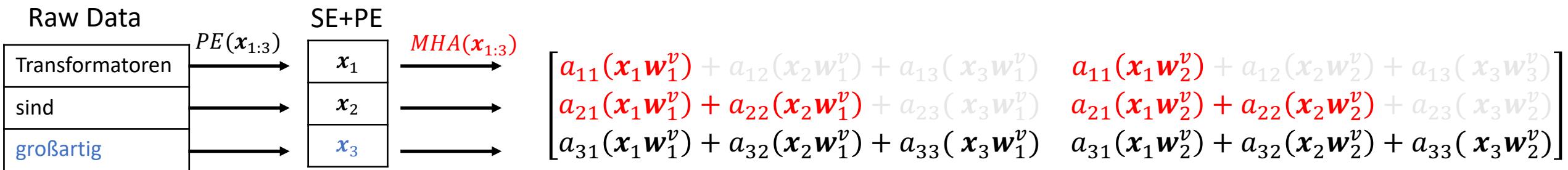
$$VW_i^V = V \in R^{3 \times 2 \times 1}$$

| |
|-------------------------------------|
| $[x_1 w_1^v \quad x_1 w_2^v] = v_1$ |
| $[x_2 w_1^v \quad x_2 w_2^v] = v_2$ |
| $[x_3 w_1^v \quad x_3 w_2^v] = v_3$ |

$$head_i = softmax \left(\frac{(\mathbf{Q}W_i^Q)(\mathbf{K}W_i^K)^T}{\sqrt{d_K}} \circ M \right) (\mathbf{V}W_i^V)$$

$$softmax \left(\frac{(\mathbf{Q}W_i^Q)(\mathbf{K}W_i^K)^T}{\sqrt{d_k}} \circ M \right) \begin{bmatrix} \mathbf{x}_1 \mathbf{w}_1^v & \mathbf{x}_1 \mathbf{w}_2^v \\ \mathbf{x}_2 \mathbf{w}_1^v & \mathbf{x}_2 \mathbf{w}_2^v \\ \mathbf{x}_3 \mathbf{w}_1^v & \mathbf{x}_3 \mathbf{w}_2^v \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \mathbf{w}_1^v & \mathbf{x}_1 \mathbf{w}_2^v \\ \mathbf{x}_2 \mathbf{w}_1^v & \mathbf{x}_2 \mathbf{w}_2^v \\ \mathbf{x}_3 \mathbf{w}_1^v & \mathbf{x}_3 \mathbf{w}_2^v \end{bmatrix}$$

Attention Scores
Cached
New Data

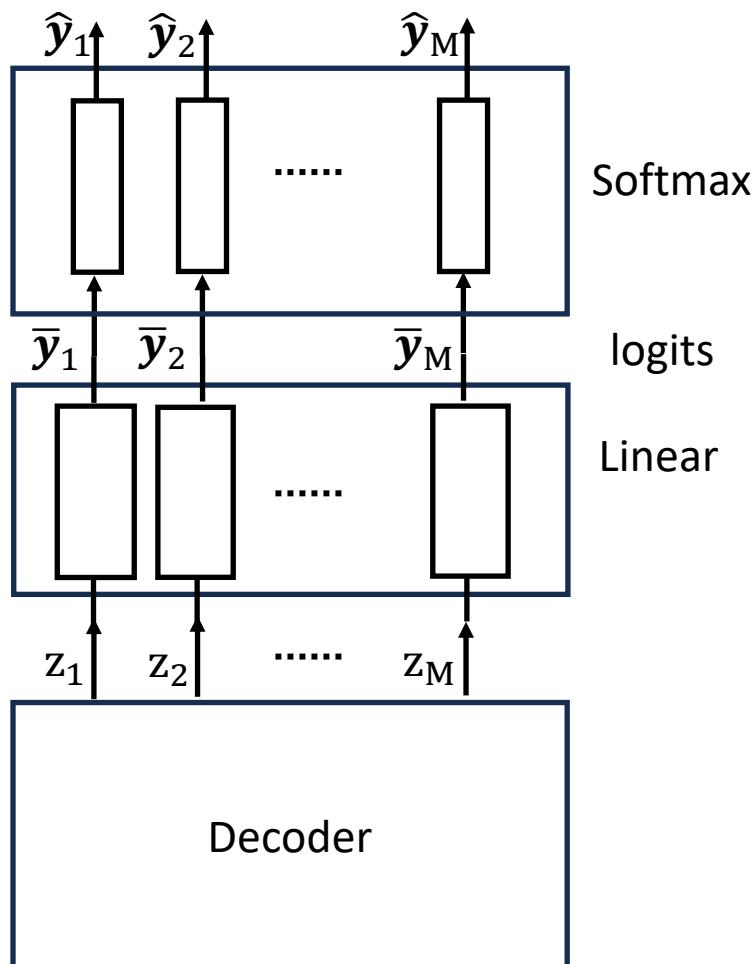


More details here:

<https://huggingface.co/blog/not-lain/kv-caching>

Output Layer

- Let $Z \in R^{M \times d}$ be the output of decoder, where M is the output sequence length and d is the embedding dimension
 - $Z = [z_1 \quad z_2 \quad \dots \quad z_M]$, where $z \in R^{d \times 1}$



$$\hat{y}_i = \text{softmax}(\bar{y}_i)$$

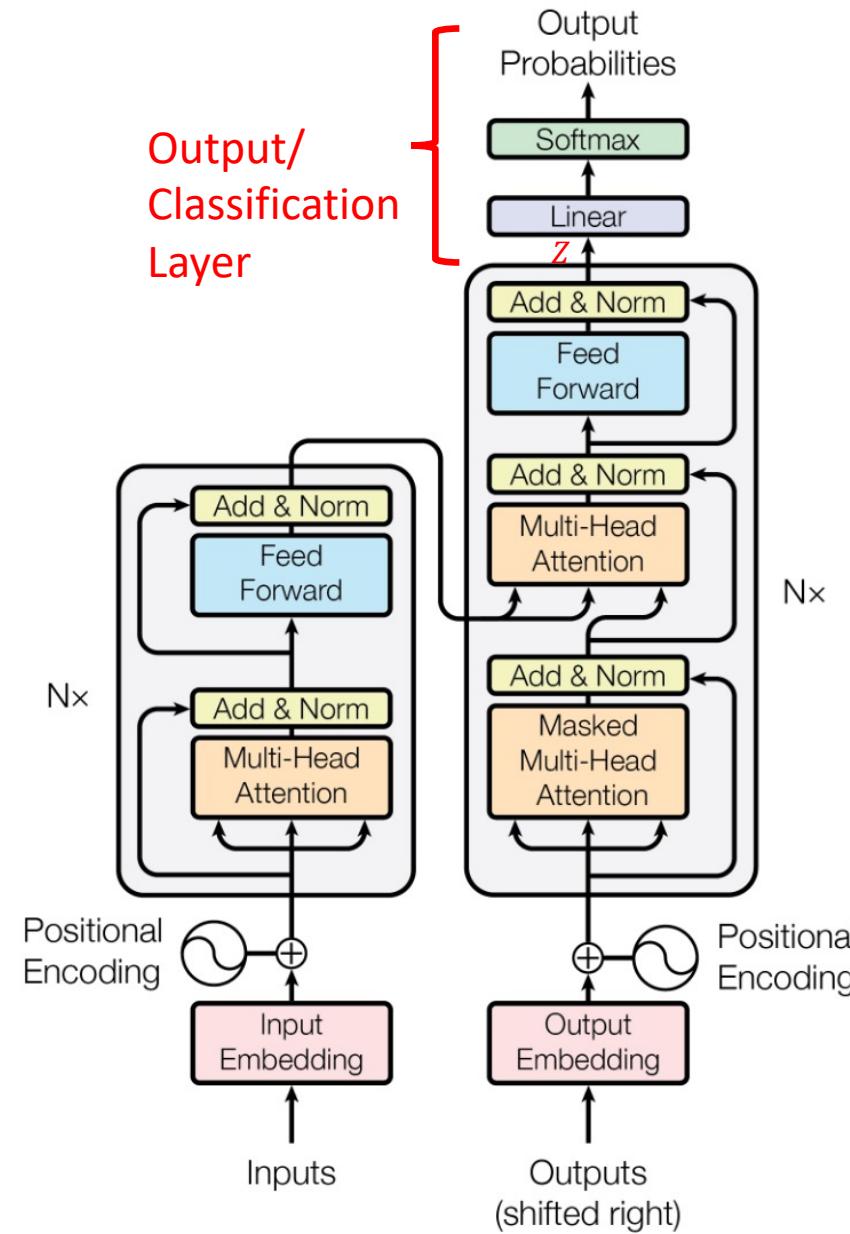
$$\hat{y}_j \in \mathbb{R}^{|V| \times 1}$$

$$\bar{y}_i \in R^{|V| \times 1}$$

$$\bar{y}_i = (W^0)^T z_i$$

$$W^0 \in R^{d \times |V|}$$

Output/ Classification Layer



Output Layer

- Output of the *softmax* is the conditional probability of the next token
- Encoder-Decoder Architecture

$$P(\hat{y}_k | \mathbf{x}_{1:N}, \hat{y}_{1:k-1})$$

- Decoder-only Architecture (GPT)

$$P(\hat{y}_k | prompt_tokens, \hat{y}_{1:k-1})$$

Sampling from Probability Distributions

- If distribution of data is known, samples (data) can be generated
- Example:

$$\begin{bmatrix} P(X = \text{"Chains"}|Y = \text{"Markov"}) \\ P(X = \text{"Markov"}|Y = \text{"Markov"}) \\ P(X = \text{"Rocks"}|Y = \text{"Markov"}) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ 0.5 \end{bmatrix}$$



Most Probable Word:

Chains and *Rocks* are equally probable

```
>>> import numpy as np  
>>> vocabulary = ["Chains", "Markov", "Rocks"]  
>>> np.random.choice(vocabulary, p=[0.5, 0, 0.5])  
['Chains']
```



Generated Sample: *Chains*

```
>>> import random  
>>> random.uniform(0, 1)  
0.2894913983065621
```

Greedy Sampling

- Greedy sampling selects the token with the **highest probability** at each step during generation.
- Pros
 - Simple and fast
 - Deterministic, i.e., always produces the same output for a given input
- Cons
 - Can lead to **repetitive** or **generic** outputs
 - May miss better sequences due to lack of exploration
 - Lacks diversity and creativity
- Use Case
 - Best suited for tasks where **precision** and **consistency** are more important than diversity
 - Eg: factual completions or deterministic templates.

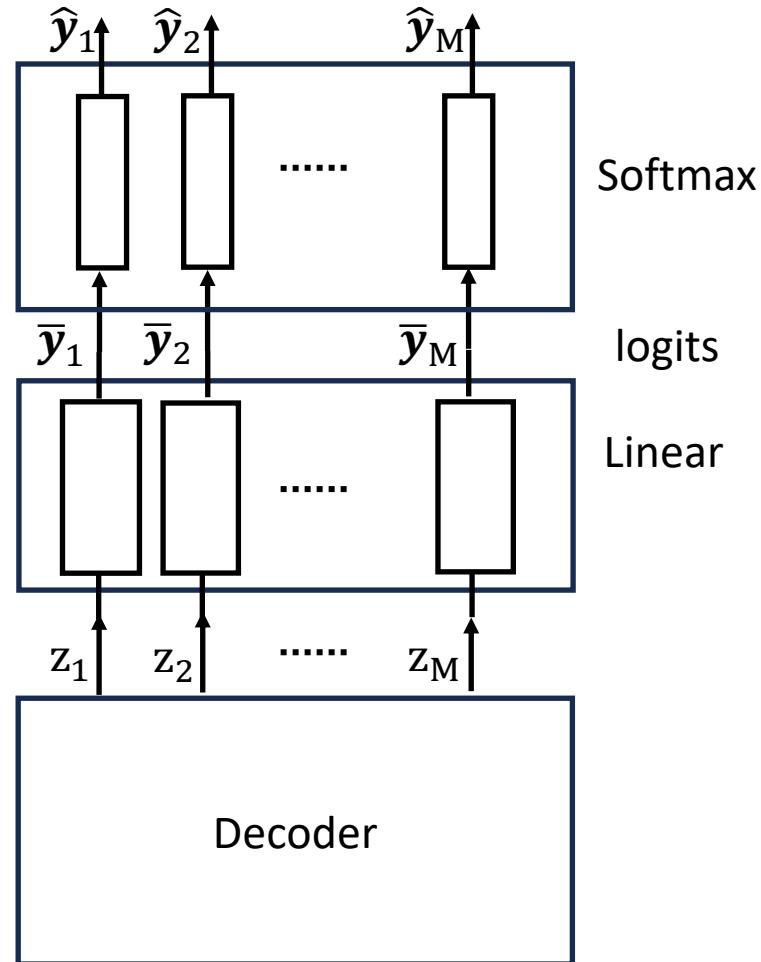
Top- K Sampling

- Restricts the model's output choices to the **K most probable tokens** at each generation step
- Retains only the top **K highest logit values**
- All other logits are set to $-\infty$ (get **zero probability** after *softmax*)
- Pro:
 - Balances diversity and quality
 - Avoids low-probability nonsensical tokens
 - Creative than greedy sampling
- Cons:
 - Choice of K is challenging ($K = 1$ is greedy and large K may lead to nonsensical tokens)

Top- p sampling or Nucleus Sampling

- Dynamically chooses the smallest set of tokens whose cumulative probability exceeds a threshold p
 - Sorts token by the predicted probabilities
 - Accumulate probabilities until the sum exceeds p
 - Sample the token from this subset
- Pros:
 - Adaptive unlike $\text{top} - K$
 - Avoids both repetitive outputs like greedy sampling and nonsensical ones
- Cons:
 - Sensitive to selection of p
- Usecase:
 - Chatbots, story generation and creative writing

Why Temperature in Sampling?



$$\bar{y}_i \in \mathbb{R}^{|V| \times 1} \quad \bar{y}_i = (W^0)^T z_i$$

- Logits (\bar{y}_i) are scaled by $\frac{1}{T}$
- $\frac{\bar{Y}}{T} = \left[\frac{\bar{y}_1}{T}, \frac{\bar{y}_2}{T}, \frac{\bar{y}_3}{T}, \dots, \frac{\bar{y}_M}{T} \right]$

$$\hat{y}_i = \text{softmax}\left(\frac{\bar{y}_i}{T}\right)$$

$$\hat{y}_i = \text{softmax}\left(\frac{\bar{y}_i}{T}\right) = \begin{bmatrix} \exp\left(\frac{\bar{y}_{i,1}}{T}\right) \\ \vdots \\ \exp\left(\frac{\bar{y}_{i,|V|}}{T}\right) \\ \hline \sum_i \exp\left(\frac{\bar{y}_i}{T}\right) \end{bmatrix}$$

$$\hat{y}_i \in \mathbb{R}^{|V| \times 1}$$

How temperature impacts sampling?

- Low temperature ($T \rightarrow 0$)

- Distribution becomes sharper (?)

$$T = 0.5 \quad \frac{\bar{y}_i}{0.5} = \begin{bmatrix} 4 \\ 2 \\ 0 \end{bmatrix} \quad softmax\left(\frac{\bar{y}_i}{0.5}\right) = \begin{bmatrix} 0.868 \\ 0.117 \\ 0.016 \end{bmatrix}$$

Sharper
Deterministic

- High temperature ($T \rightarrow \infty$)
 - Distribution becomes flatter

$$T = 1 \quad \bar{y}_i = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \quad softmax(\bar{y}_i) = \begin{bmatrix} 0.665 \\ 0.245 \\ 0.090 \end{bmatrix}$$

$$T = 2 \quad \frac{\bar{y}_i}{2} = \begin{bmatrix} 4 \\ 2 \\ 0 \end{bmatrix} \quad softmax\left(\frac{\bar{y}_i}{2}\right) = \begin{bmatrix} 0.506 \\ 0.307 \\ 0.186 \end{bmatrix}$$

$$T = \infty \quad \frac{\bar{y}_i}{\infty} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad softmax\left(\frac{\bar{y}_i}{2}\right) = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}$$

Flat
Random generation

Number of Parameters

- Embedding Layer: $|V| * d_{model}$. (no bias term in embedding layers)
- Positional Encoding:
 - Fixed PE: None
 - Learned PE: $L_{seq} * d_{model}$ (no bias terms)
- Per-Encoder:
 - MHA: $4 * d_{model} * d_{model} + 4 * d_{model}$ (Q, K, V, W_O matrices and 1 MHA blocks)
 - FFN: $2 * d_{model} * d_{FF} + d_{FF} + d_{model}$ ($W^{[1]}$ and $W^{[2]}$ matrices and 1 FFN block)
 - LN: $2 * 2 * d_{model}$ (γ and β and 2 LN blocks)
- Per-Decoder:
 - MHA: $2 * (4 * d_{model} * d_{model} + 4 * d_{model})$ (Q, K, V, W_O matrices and 2 MHA blocks)
 - FFN: $(2 * d_{model} * d_{FF} + d_{FF} + d_{model})$ ($W^{[1]}$ and $W^{[2]}$ matrices and 1 FFN block)
 - LN: $3 * 2 * d_{model}$ (γ and β and 3 LN blocks)
- Output Layer:
 - Projection: $d_{model} * |V|$ (no bias term in output layer)

FLOPs per Layer with single head

- Attention:

- Q, K, V Projections

- $QW^Q, KW^K, VW^V: 3 * (2 * d_{model}) * (L_{seq} * d_{model})$

↑
FLOPs per element # of elements

Due to multiplications and additions in matrix multiplication

- Attention Scores (A)

- $QW^Q(KW^K)^T: (2 * d_{model}) * (L_{seq} * L_{seq})$
 - softmax: $O(L_{seq}^2)$

- Attention Values

- $A(VW^V): (2 * L_{seq}) * (L_{seq} * d_{model})$

- Output Projection

- $(AVW^V)W_O: (2 * d_{model}) * (L_{seq} * d_{model})$

- FFN

- $(2 * d_{model}) * (L_{seq} * d_{FF}) + (2 * d_{FF}) * (L_{seq} * d_{model})$

- LN:

- $O(L_{seq} * d_{model})$

$$Q, K, V \in R^{L_{seq} \times d_{model}}$$

$$W^Q, W^K, W^V, W_O \in R^{d_{model} \times d_{model}}$$

$$QW^Q, KW^K, VW^V \in R^{L_{seq} \times d_{model}}$$

$$QW^Q(KW^K)^T \in R^{L_{seq} \times L_{seq}}$$

$$A \in R^{L_{seq} \times L_{seq}}: \text{Attention Matrix}$$

$$W^{[1]} \in R^{d_{model} \times d_{FF}}$$

$$W^{[2]} \in R^{d_{FF} \times d_{model}}$$

FLOPS per Layer

- Quadratic in both sequence length and embedding dimension
- For smaller sequences
 - FFN is the bottleneck
- For larger sequences
 - Attention layer is the bottleneck
- KV Caching makes attention cost linear in L_{seq}

Vision Transformers

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

**Alexey Dosovitskiy^{*,†}, Lucas Beyer^{*}, Alexander Kolesnikov^{*}, Dirk Weissenborn^{*},
Xiaohua Zhai^{*}, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer,
Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby^{*,†}**

*equal technical contribution, †equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

Vision Transformers

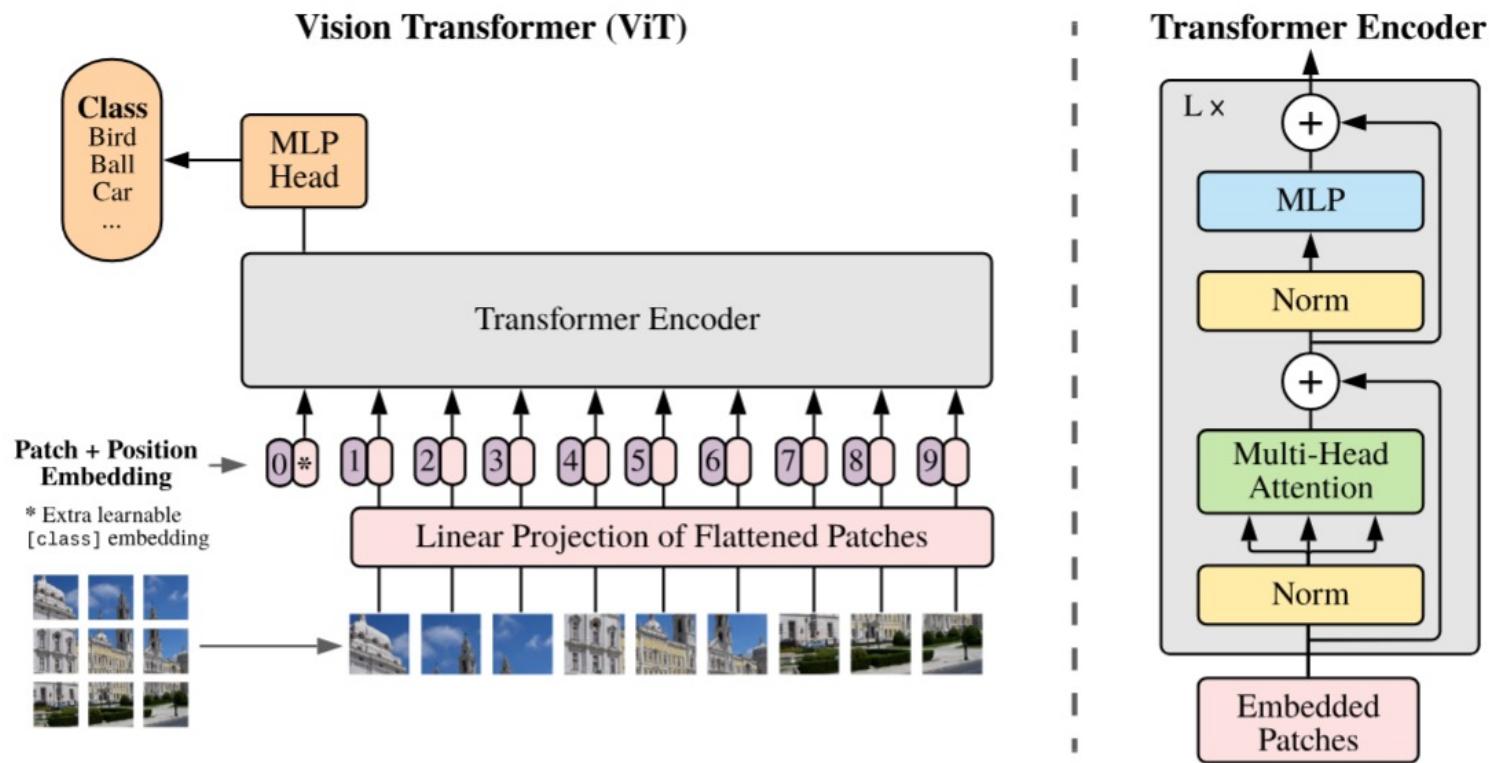
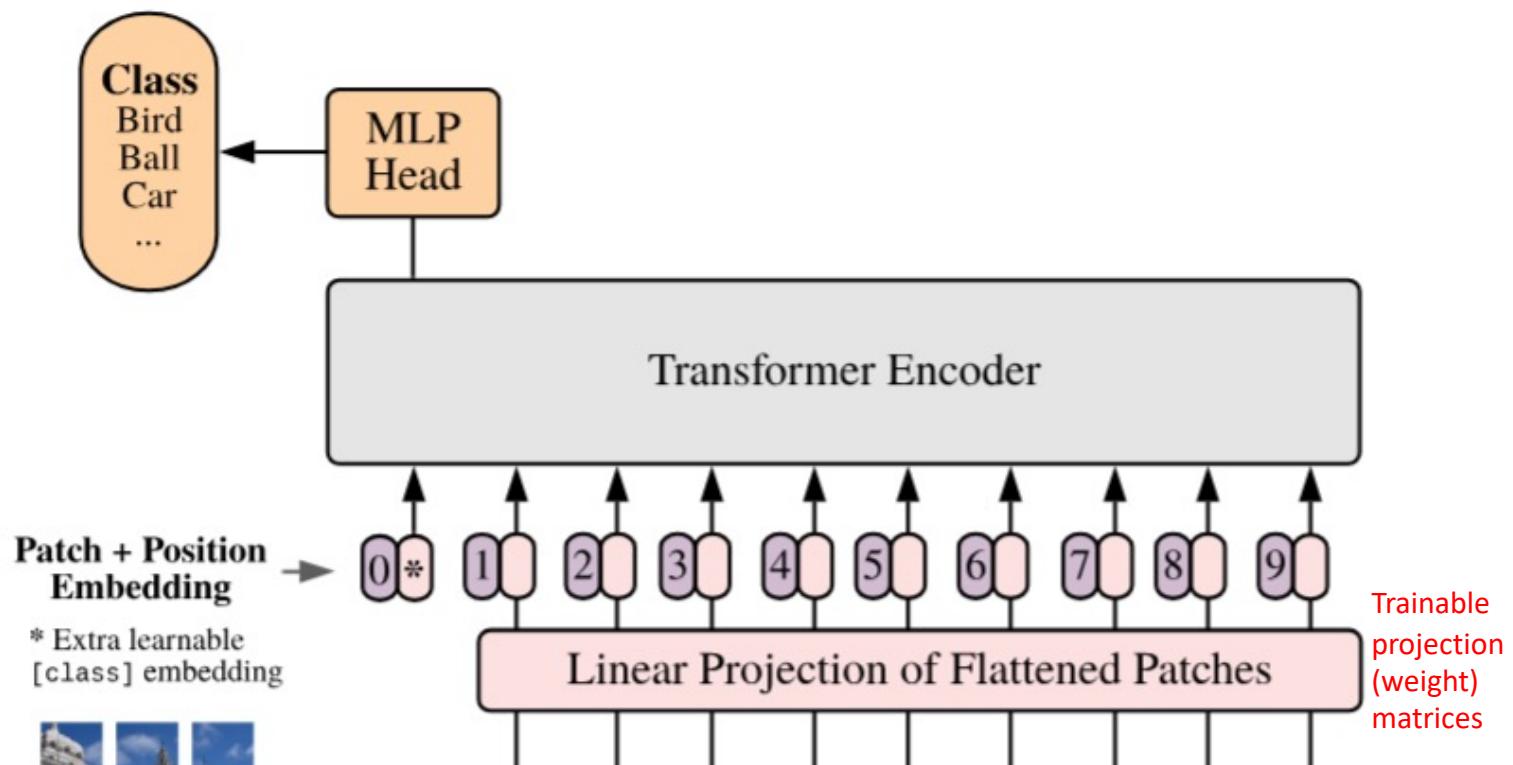


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable “classification token” to the sequence. The illustration of the Transformer encoder was inspired by [Vaswani et al. \(2017\)](#).

Vision Transformer (ViT)



Output embedding vector corresponding to * token represents the whole image (similar to feature vector)