NUS | Computing
National University of Singapore

IT5005 Artificial Intelligence

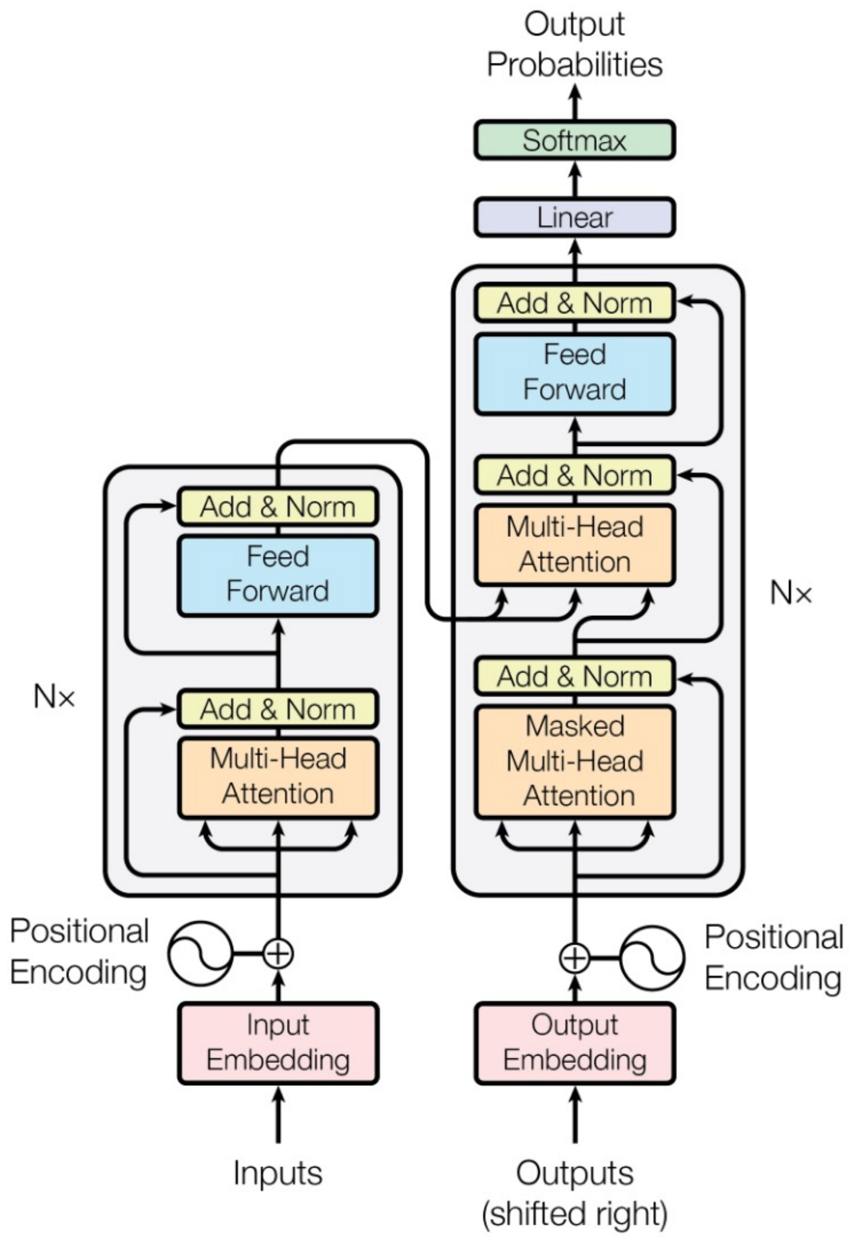
Sirigina Rajendra Prasad
AY2025/2026: Semester 1

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 $\hat{\boldsymbol{y}}_{1:\mathrm{t}}$
    - 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 $t = 1$

Transformatoren
sind
großartig

Raw Data

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

$$\begin{array}{|c|}\hline \boldsymbol{x}_1 \\ \hline x_2 \\ \hline x_3 \\ \hline \end{array}$$

SE+PE

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

$$\begin{array}{|c|}\hline \boldsymbol{x}_1 \\ \hline x_2 \\ \hline x_3 \\ \hline \end{array}$$

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

$$\boxed{[\boldsymbol{w}_1^q \;\; \boldsymbol{w}_2^q\,]}$$

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

$$\begin{array}{|c|}\hline [\boldsymbol{x}_1\boldsymbol{w}_1^q \quad \boldsymbol{x}_1\boldsymbol{w}_2^q] = \boldsymbol{q}_1 \\ \hline [x_2\boldsymbol{w}_1^q \quad x_2\boldsymbol{w}_2^q] = \boldsymbol{q}_2 \\ \hline [x_3\boldsymbol{w}_1^q \quad x_3\boldsymbol{w}_2^q] = \boldsymbol{q}_3 \\ \hline \end{array}$$

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

$$\begin{array}{|c|}\hline \boldsymbol{x}_1 \\ \hline x_2 \\ \hline x_3 \\ \hline \end{array}$$

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

$$\boxed{[\boldsymbol{w}_1^k \;\; \boldsymbol{w}_2^k\,]}$$

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

$$\begin{array}{|c|}\hline [\boldsymbol{x}_1\boldsymbol{w}_1^k \quad \boldsymbol{x}_1\boldsymbol{w}_2^k] = \boldsymbol{k}_1 \\ \hline [x_2\boldsymbol{w}_1^k \quad x_2\boldsymbol{w}_2^k] = \boldsymbol{k}_2 \\ \hline [x_3\boldsymbol{w}_1^k \quad x_3\boldsymbol{w}_2^k] = \boldsymbol{k}_3 \\ \hline \end{array}$$

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

$$\begin{array}{|c|}\hline \boldsymbol{x}_1 \\ \hline x_2 \\ \hline x_3 \\ \hline \end{array}$$

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

$$\boxed{[\boldsymbol{w}_1^v \;\; \boldsymbol{w}_2^v\,]}$$

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

$$\begin{array}{|c|}\hline [\boldsymbol{x}_1\boldsymbol{w}_1^v \quad \boldsymbol{x}_1\boldsymbol{w}_2^v] = \boldsymbol{v}_1 \\ \hline [x_2\boldsymbol{w}_1^v \quad x_2\boldsymbol{w}_2^v] = \boldsymbol{v}_2 \\ \hline [x_3\boldsymbol{w}_1^v \quad x_3\boldsymbol{w}_2^v] = \boldsymbol{v}_3 \\ \hline \end{array}$$

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

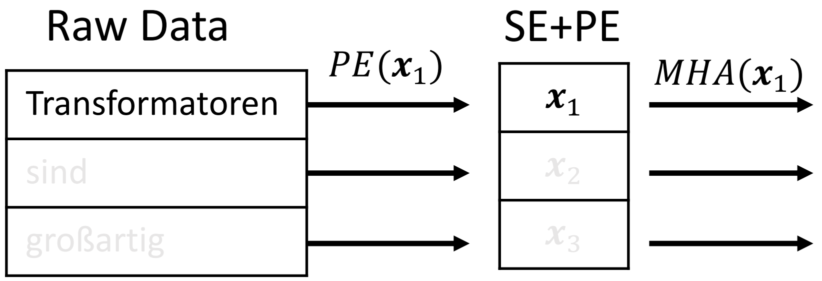
	Transformers	are	great
Transformers	a_{11}	a_{12}	a_{13}
are	a_{21}	a_{22}	a_{23}
great	a_{31}	a_{32}	a_{33}

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

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

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

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



$$\begin{bmatrix} a_{11}(\boldsymbol{x}_1\boldsymbol{w}_1^v) + a_{12}(\boldsymbol{x}_2\boldsymbol{w}_1^v) + a_{13}(\boldsymbol{x}_3\boldsymbol{w}_1^v) & a_{11}(\boldsymbol{x}_1\boldsymbol{w}_2^v) + a_{12}(\boldsymbol{x}_2\boldsymbol{w}_2^v) + a_{13}(\boldsymbol{x}_3\boldsymbol{w}_3^v) \\ a_{21}(\boldsymbol{x}_1\boldsymbol{w}_1^v) + a_{22}(\boldsymbol{x}_2\boldsymbol{w}_1^v) + a_{23}(\boldsymbol{x}_3\boldsymbol{w}_1^v) & a_{21}(\boldsymbol{x}_1\boldsymbol{w}_2^v) + a_{22}(\boldsymbol{x}_2\boldsymbol{w}_2^v) + a_{23}(\boldsymbol{x}_3\boldsymbol{w}_2^v) \\ a_{31}(\boldsymbol{x}_1\boldsymbol{w}_1^v) + a_{32}(\boldsymbol{x}_2\boldsymbol{w}_1^v) + a_{33}(\boldsymbol{x}_3\boldsymbol{w}_1^v) & a_{31}(\boldsymbol{x}_1\boldsymbol{w}_2^v) + a_{32}(\boldsymbol{x}_2\boldsymbol{w}_2^v) + a_{33}(\boldsymbol{x}_3\boldsymbol{w}_2^v) \end{bmatrix}$$

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

Inference: at $t = 2$

Cached
New Data

Transformatoren
sind
großartig

Raw Data

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
x_3

SE+PE

$Q \in R^{3\times2}$

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ x_3 \end{bmatrix} \quad [\boldsymbol{w}_1^q \;\; \boldsymbol{w}_2^q] \quad \begin{array}{l} [\boldsymbol{x}_1\boldsymbol{w}_1^q \quad \boldsymbol{x}_1\boldsymbol{w}_2^q] = \boldsymbol{q}_1 \\ [\boldsymbol{x}_2\boldsymbol{w}_1^q \quad \boldsymbol{x}_2\boldsymbol{w}_2^q] = \boldsymbol{q}_2 \\ [x_3w_1^q \quad x_3w_2^q] = q_3 \end{array}$$

$K \in R^{3\times2}$ $\quad W_i^K \in R^{2\times2}$ $\quad KW_i^K = K \in R^{2\times2\times1}$

$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ x_3 \end{bmatrix} \quad [\boldsymbol{w}_1^k \;\; \boldsymbol{w}_2^k] \quad \begin{array}{l} [\boldsymbol{x}_1\boldsymbol{w}_1^k \quad \boldsymbol{x}_1\boldsymbol{w}_2^k] = \boldsymbol{k}_1 \\ [\boldsymbol{x}_2\boldsymbol{w}_1^k \quad \boldsymbol{x}_2\boldsymbol{w}_2^k] = \boldsymbol{k}_2 \\ [x_3w_1^k \quad x_3w_2^k] = k_3 \end{array}$$

$V \in R^{3\times2}$ $\quad W_i^V \in R^{2\times2}$ $\quad VW_i^V = V \in R^{2\times2\times1}$

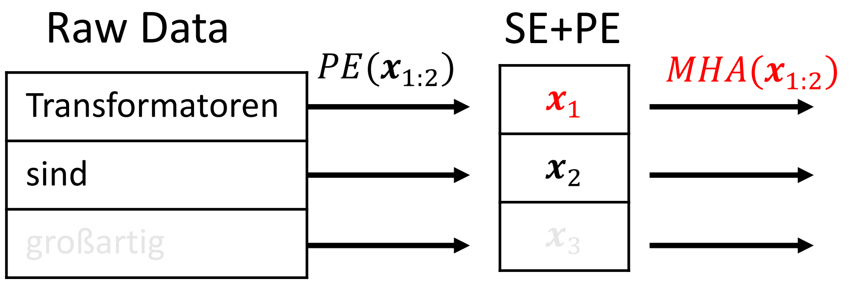
$$\begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ x_3 \end{bmatrix} \quad [\boldsymbol{w}_1^v \;\; \boldsymbol{w}_2^v] \quad \begin{array}{l} [\boldsymbol{x}_1\boldsymbol{w}_1^v \quad \boldsymbol{x}_1\boldsymbol{w}_2^v] = \boldsymbol{v}_1 \\ [\boldsymbol{x}_2\boldsymbol{w}_1^v \quad \boldsymbol{x}_2\boldsymbol{w}_2^v] = \boldsymbol{v}_2 \\ [x_3w_1^v \quad x_3w_2^v] = v_3 \end{array}$$

Now the decoder sees a sequence of length 2

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

$$softmax\left(\frac{(QW_i^Q)(KW_i^K)^T}{\sqrt{d_k}} \circ M\right)\begin{bmatrix} \boldsymbol{x}_1\boldsymbol{w}_1^v & \boldsymbol{x}_1\boldsymbol{w}_2^v \\ \boldsymbol{x}_2\boldsymbol{w}_1^v & \boldsymbol{x}_2\boldsymbol{w}_2^v \\ x_3w_1^v & x_3w_2^v \end{bmatrix} = \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\begin{bmatrix} \boldsymbol{x}_1\boldsymbol{w}_1^v & \boldsymbol{x}_1\boldsymbol{w}_2^v \\ \boldsymbol{x}_2\boldsymbol{w}_1^v & \boldsymbol{x}_2\boldsymbol{w}_2^v \\ x_3w_1^v & x_3w_2^v \end{bmatrix}$$

Cached
New Data



$$\begin{bmatrix} a_{11}(\boldsymbol{x}_1\boldsymbol{w}_1^v) + a_{12}(x_2w_1^v) + a_{13}(x_3w_1^v) & a_{11}(\boldsymbol{x}_1\boldsymbol{w}_2^v) + a_{12}(x_2w_2^v) + a_{13}(x_3w_3^v) \\ a_{21}(\boldsymbol{x}_1\boldsymbol{w}_1^v) + a_{22}(\boldsymbol{x}_2\boldsymbol{w}_1^v) + a_{23}(x_3w_1^v) & a_{21}(\boldsymbol{x}_1\boldsymbol{w}_2^v) + a_{22}(\boldsymbol{x}_2\boldsymbol{w}_2^v) + a_{23}(x_3w_2^v) \\ a_{31}(x_1w_1^v) + a_{32}(x_2w_1^v) + a_{33}(x_3w_1^v) & a_{31}(x_1w_2^v) + a_{32}(x_2w_2^v) + a_{33}(x_3w_2^v) \end{bmatrix}$$

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

Why KV Caching

Raw Data
Transformatoren
sind
großartig

$X \in R^{3\times2}$

SE+PE
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

$Q \in R^{3\times2}$

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

$[\boldsymbol{w}_1^q \ \ \boldsymbol{w}_2^q\,]$

$[\boldsymbol{x}_1\boldsymbol{w}_1^q \quad \boldsymbol{x}_1\boldsymbol{w}_2^q] = \boldsymbol{q}_1$
$[\boldsymbol{x}_2\boldsymbol{w}_1^q \quad \boldsymbol{x}_2\boldsymbol{w}_2^q] = \boldsymbol{q}_2$
$[\boldsymbol{x}_3\boldsymbol{w}_1^q \quad \boldsymbol{x}_3\boldsymbol{w}_2^q] = \boldsymbol{q}_3$

$K \in R^{3\times2}$

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

$W_i^K \in R^{2\times2}$

$[\boldsymbol{w}_1^k \ \ \boldsymbol{w}_2^k\,]$

$KW_i^K = K \in R^{3\times2\times1}$

$[\boldsymbol{x}_1\boldsymbol{w}_1^k \quad \boldsymbol{x}_1\boldsymbol{w}_2^k] = \boldsymbol{k}_1$
$[\boldsymbol{x}_2\boldsymbol{w}_1^k \quad \boldsymbol{x}_2\boldsymbol{w}_2^k] = \boldsymbol{k}_2$
$[\boldsymbol{x}_3\boldsymbol{w}_1^k \quad \boldsymbol{x}_3\boldsymbol{w}_2^k] = \boldsymbol{k}_3$

$V \in R^{3\times2}$

$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$

$W_i^V \in R^{2\times2}$

$[\boldsymbol{w}_1^v \ \ \boldsymbol{w}_2^v\,]$

$VW_i^V = V \in R^{3\times2\times1}$

$[\boldsymbol{x}_1\boldsymbol{w}_1^v \quad \boldsymbol{x}_1\boldsymbol{w}_2^v] = \boldsymbol{v}_1$
$[\boldsymbol{x}_2\boldsymbol{w}_1^v \quad \boldsymbol{x}_2\boldsymbol{w}_2^v] = \boldsymbol{v}_2$
$[\boldsymbol{x}_3\boldsymbol{w}_1^v \quad \boldsymbol{x}_3\boldsymbol{w}_2^v] = \boldsymbol{v}_3$

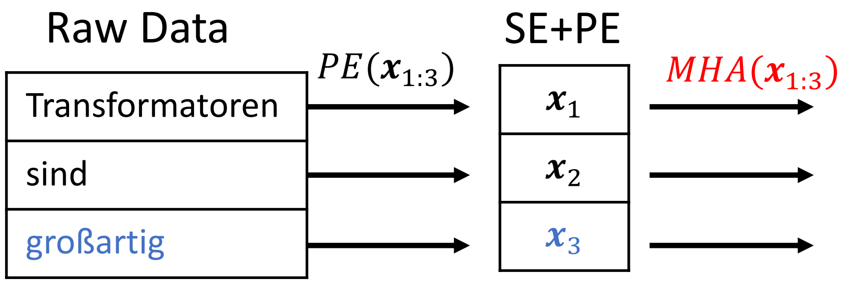
Cached
New Data

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

Attention Scores

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

Cached
New Data



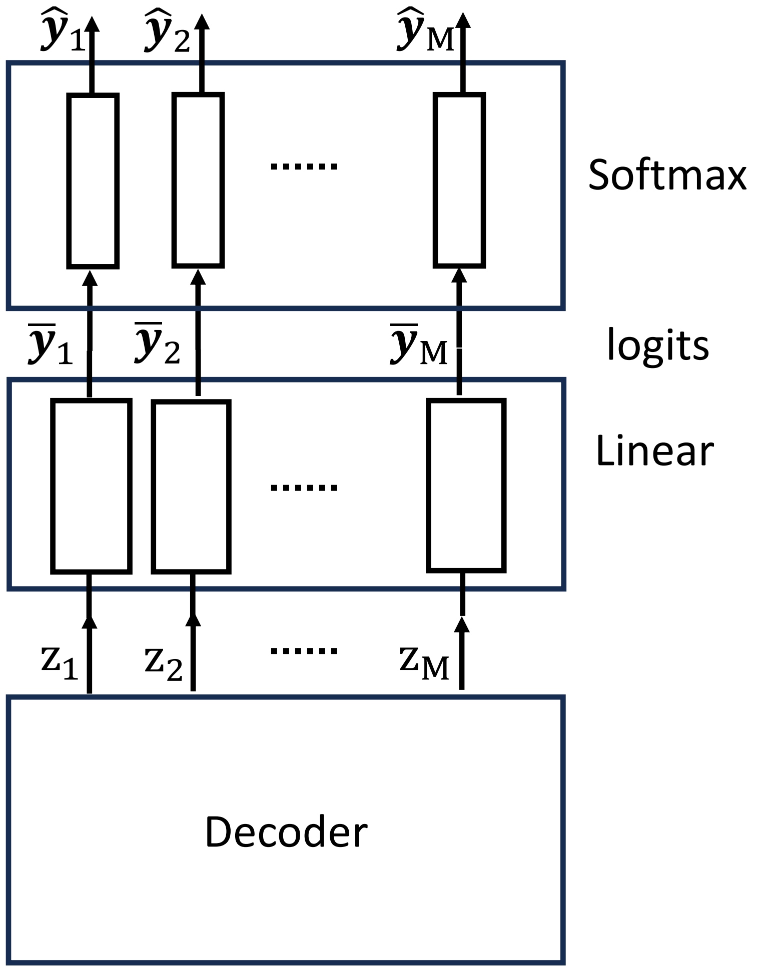
$$\begin{bmatrix} a_{11}(\mathbf{x}_1\mathbf{w}_1^v) + a_{12}(\mathbf{x}_2\mathbf{w}_1^v) + a_{13}(\mathbf{x}_3\mathbf{w}_1^v) & a_{11}(\mathbf{x}_1\mathbf{w}_2^v) + a_{12}(\mathbf{x}_2\mathbf{w}_2^v) + a_{13}(\mathbf{x}_3\mathbf{w}_3^v) \\ a_{21}(\mathbf{x}_1\mathbf{w}_1^v) + a_{22}(\mathbf{x}_2\mathbf{w}_1^v) + a_{23}(\mathbf{x}_3\mathbf{w}_1^v) & a_{21}(\mathbf{x}_1\mathbf{w}_2^v) + a_{22}(\mathbf{x}_2\mathbf{w}_2^v) + a_{23}(\mathbf{x}_3\mathbf{w}_2^v) \\ a_{31}(\mathbf{x}_1\mathbf{w}_1^v) + a_{32}(\mathbf{x}_2\mathbf{w}_1^v) + a_{33}(\mathbf{x}_3\mathbf{w}_1^v) & a_{31}(\mathbf{x}_1\mathbf{w}_2^v) + a_{32}(\mathbf{x}_2\mathbf{w}_2^v) + a_{33}(\mathbf{x}_3\mathbf{w}_2^v) \end{bmatrix}$$

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
  - $Z = [z_1 \quad z_2 \quad \cdots \quad z_M]$, where $z \in R^{d \times 1}$



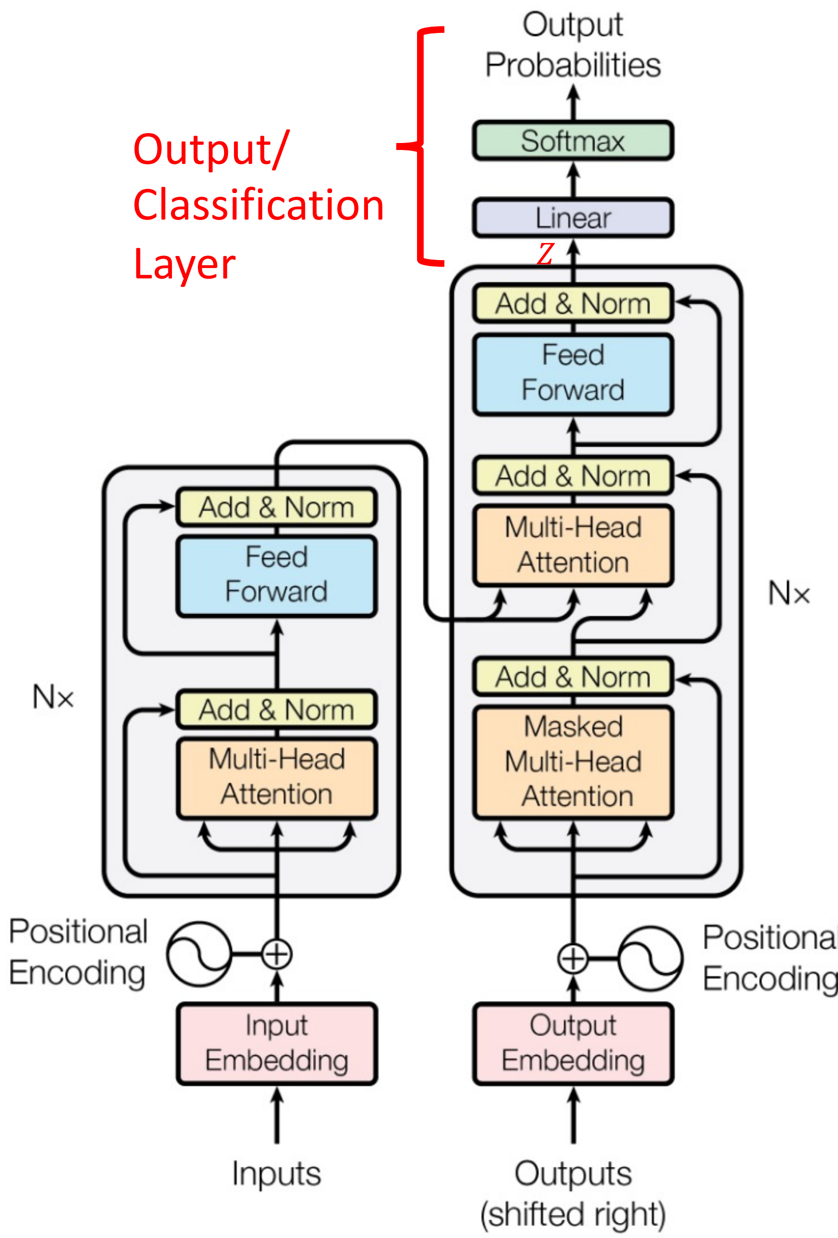
$$\hat{y}_i = \text{softmax}(\bar{y}_i)$$

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

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

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

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



Output Layer

- Output of the $softmax$ is the conditional probability of the next token

- Encoder-Decoder Architecture

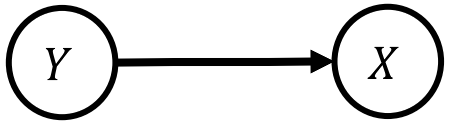
$$P(\hat{y}_k|\boldsymbol{x}_{1:\mathrm{N}}, \hat{y}_{1:k-1})$$

- Decoder-only Architecture (GPT)

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

Sampling from Probability Distributions

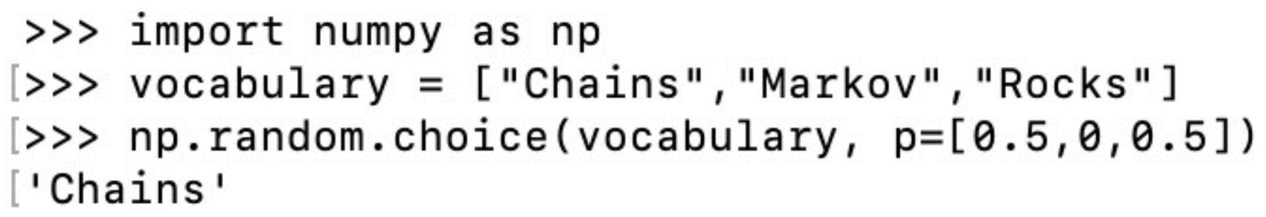
- If distribution of data is known, samples (data) can be generated
- Example:



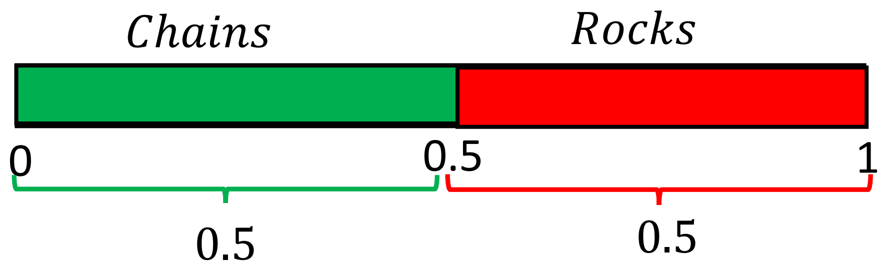
$$\begin{tikzcd} Y \arrow[r] & X \end{tikzcd}$$

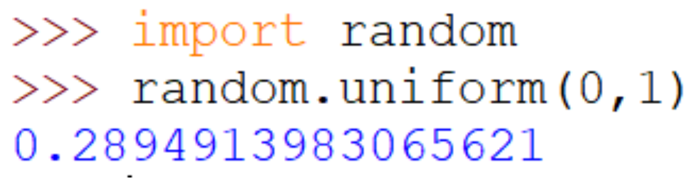
$$\begin{bmatrix} P(X = \text{“}Chains\text{”} | Y = \text{“}Markov\text{”}) \\ P(X = \text{“}Markov\text{”} | Y = \text{“}Markov\text{”}) \\ P(X = \text{“}Rocks\text{”} | Y = \text{“}Markov\text{”}) \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0 \\ 0.5 \end{bmatrix}$$

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'
```





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

Generated Sample: *Chains*

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.

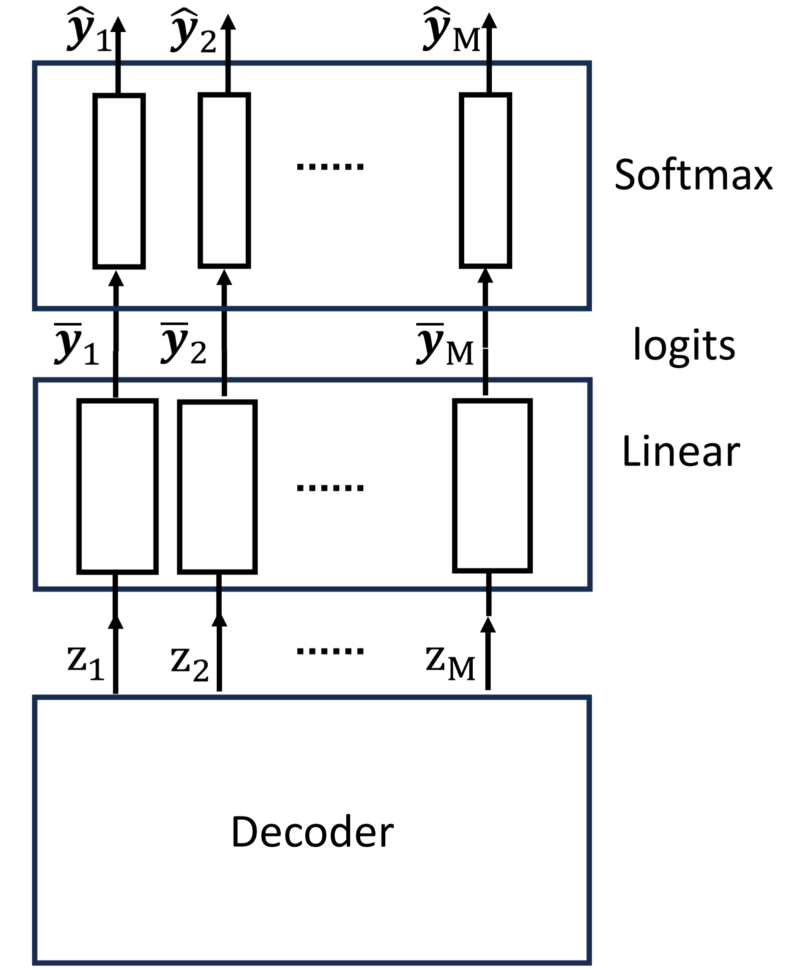
Top-K Sampling

- Restricts the model's output choices to the K most probable tokens at each generation step
- Retains only the top $\boldsymbol{K}$ **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 $top - K$
  - 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?



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

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

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

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

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

How temperature impacts sampling?

- Low temperature ($T \rightarrow 0$)
  - Ditribution becomes sharper (?)
- High temperature ($T \rightarrow \infty$)
  - Distribution becomes flatter

$$T = 0.5 \qquad \frac{\bar{\boldsymbol{y}}_{\mathrm{i}}}{0.5} = \begin{bmatrix} 4 \\ 2 \\ 0 \end{bmatrix} \qquad softmax\left(\frac{\bar{\boldsymbol{y}}_{\mathrm{i}}}{0.5}\right) = \begin{bmatrix} 0.868 \\ 0.117 \\ 0.016 \end{bmatrix}$$

Sharper
Deterministic

$$T = 1 \qquad \bar{\boldsymbol{y}}_{\mathrm{i}} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \qquad softmax(\bar{\boldsymbol{y}}_{\mathrm{i}}) = \begin{bmatrix} 0.665 \\ 0.245 \\ 0.090 \end{bmatrix}$$

$$T = 2 \qquad \frac{\bar{\boldsymbol{y}}_{\mathrm{i}}}{2} = \begin{bmatrix} 4 \\ 2 \\ 0 \end{bmatrix} \qquad softmax\left(\frac{\bar{\boldsymbol{y}}_{\mathrm{i}}}{2}\right) = \begin{bmatrix} 0.506 \\ 0.307 \\ 0.186 \end{bmatrix}$$

$$T = \infty \qquad \frac{\bar{\boldsymbol{y}}_{\mathrm{i}}}{\infty} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad softmax\left(\frac{\bar{\boldsymbol{y}}_{\mathrm{i}}}{2}\right) = \begin{bmatrix} \frac{1}{3} \\ \frac{1}{3} \\ \frac{1}{3} \end{bmatrix}$$

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) matrix
  - FFN: $2 * d_{model} * d_{FF} + d_{FF} + d_{model}$
  - LN: $2 * 2 * d_{model}$
- Per-Decoder:
  - MHA: $2 * (4 * d_{model} * d_{model} + 4\, d_{model})$ (Q, K, V, W^O) matrix
  - FFN: $2 * d_{model} * d_{FF} + d_{FF} + d_{model}$
  - LN: $3 * 2 * d_{model}$. (no bias terms)
- Output Layer:
  - Projection: $d\_model * |V|$

FLOPs per Layer

- Attention:
  - Q, K, V Projections
    - $QW^Q, KW^K, VW^V: 3 * 2 * L_{seq} * d_{model}^2$
  - Attention Scores (A)
    - $QW^Q(KW^K)^T$: $2 * L_{seq}^2 * d_{model}$
    - $softmax: O(L_{seq}^2)$
  - Attention Values
    - $A(VW^V)$: $2 * L_{seq}^2 * d_{model}$
  - Output Projection
    - $(AVW^V)W_O: 2 * L_{seq} * d_{model}^2$
- FFN
  - $2 * L_{seq} * 2 * d_{model} * d_{FF} \approx 2 * L_{seq} * 2 * d_{model}^2$
- LN:
  - $O(L_{seq} * d_{model})$

Due to multiplications and additions in matrix multiplication

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

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

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

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

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

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

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

$d_{FF} = 4d_{model}$

FLOPS per Layer

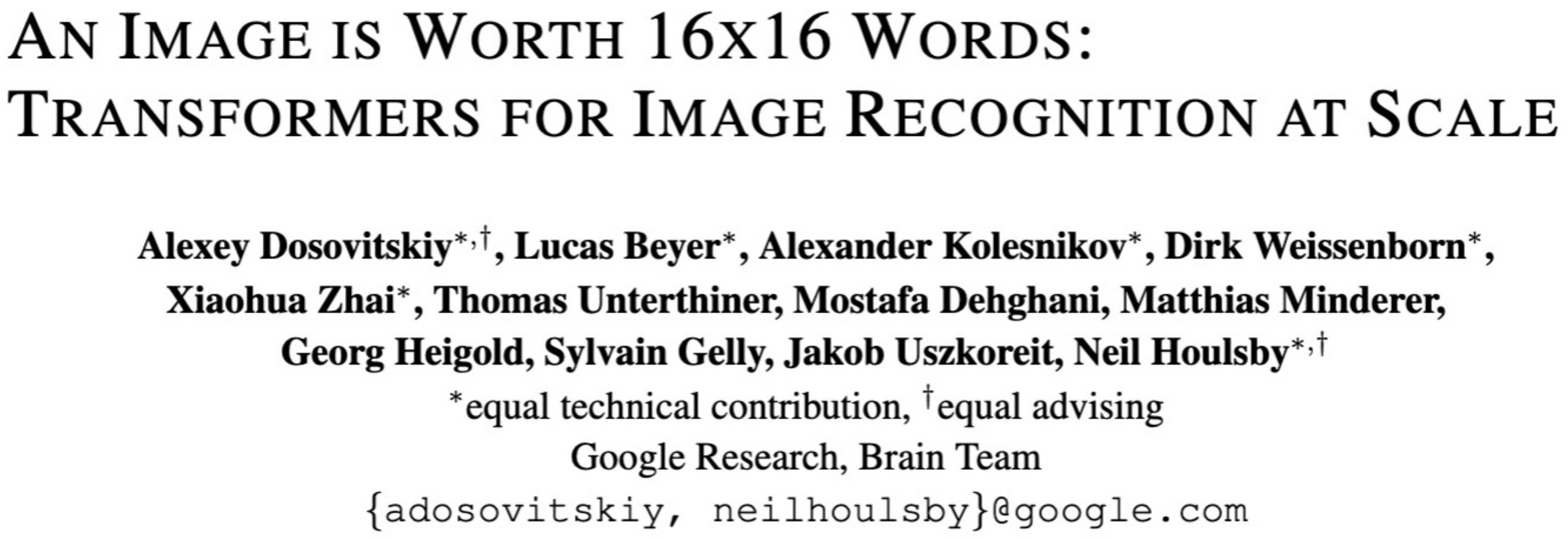
- Quadratic in both sequence length and embedding dimension

- 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

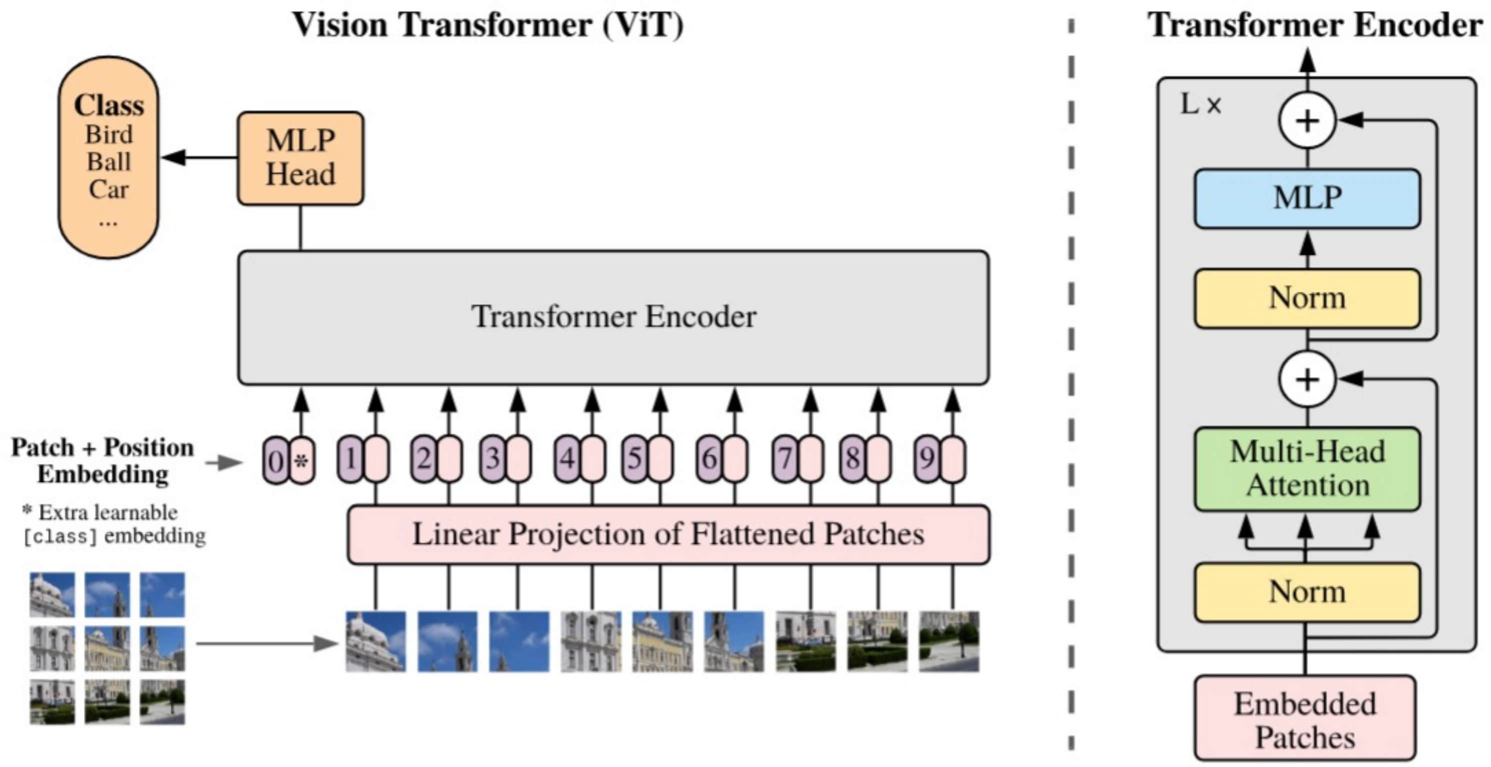
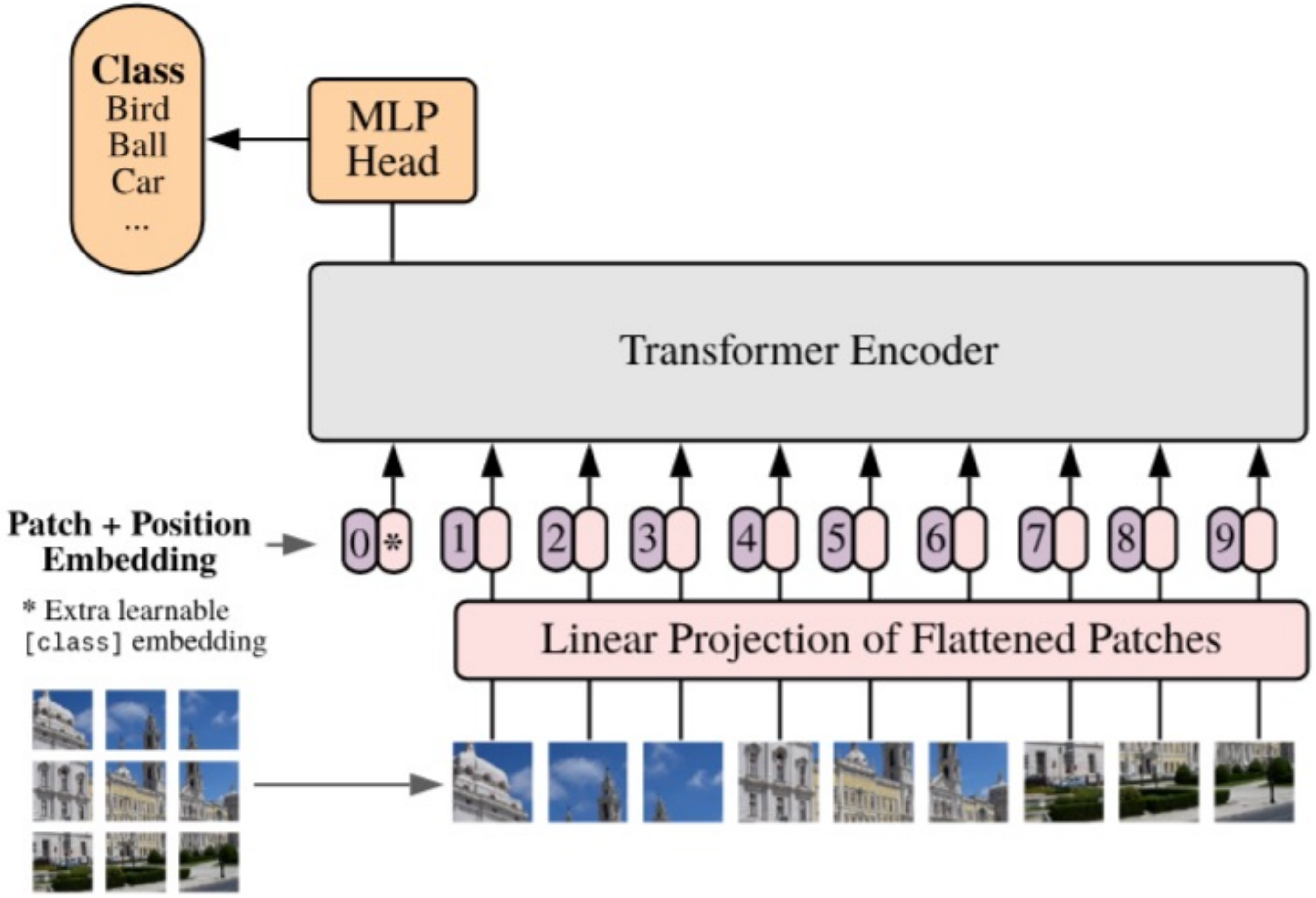


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification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

Vision Transformer (ViT)


Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0 * 1 2 3 4 5 6 7 8 9
Linear Projection of Flattened Patches

Trainable projection (weight) matrices

$\boldsymbol{x} \in R^{H \times W \times 3}$

$\boldsymbol{x} \in R^{N \times (3P^2)}$

$\boldsymbol{N} = \frac{HW}{P^2}$: Number of patches

P: Patch height (width)

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