Transformers

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

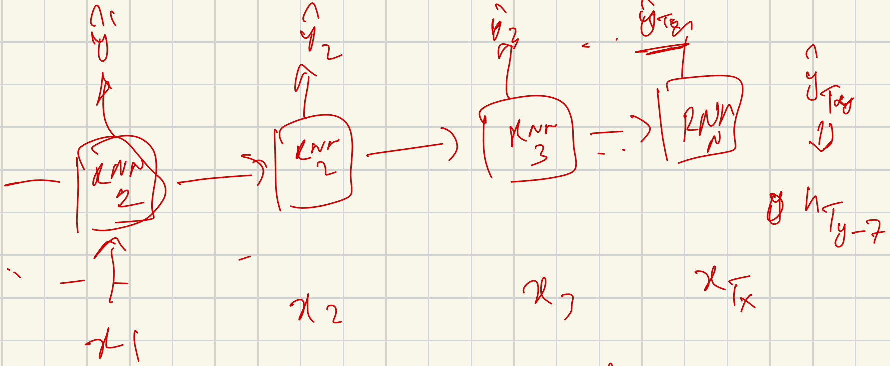
KAUST Academy
King Abdullah University of Science and Technology

June 11, 2025

Table of Contents

Motivation for Transformers

- **Limited Context:**
  - RNNs process sequences step-by-step → Difficult to capture long-range dependencies.
  - CNNs are effective for local patterns → Struggle with global context in sequences.
- **Sequential Bottlenecks:**
  - RNNs process data sequentially.
  - Prevents efficient parallelization.
  - Leads to slow training and inference.
- **Slow Training:**
  - RNNs and CNNs are computationally intensive.
  - Especially challenging for long sequences or large images.
  - Results in increased training times.



Vanilla RNNs → 7 cells

The boy, who was playing football, also loved to watch cricket.

Motivation for Transformers (cont.)

There is a need for models that:

- Capture comprehensive, global context efficiently.
- Allow for parallel computation.
- Accelerate training and inference.

Transformers address these needs by:

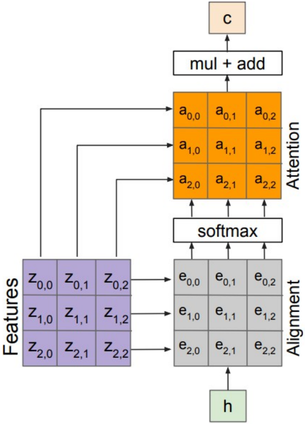
- Leveraging self-attention mechanisms.
- Modeling dependencies across entire sequences.
- Enabling highly parallelizable computation.

Learning Outcomes

After this lecture, you will be able to:

- Explain the motivation behind the development of Transformers.
- Describe the architecture and key components of Transformer models.
- Understand and implement attention and self-attention mechanisms.
- Explain the role of positional encoding in Transformers.
- Distinguish between general attention and self-attention.
- Understand the concept and benefits of multi-head attention.
- Compare CNNs with and without self-attention.
- Summarize the advantages of Transformers over traditional sequence models.

- Previously, we saw how attention helps image captioning models focus on important regions, generating more relevant captions.
- However, we noticed limitations—attention sometimes misses subtle details or struggles with complex scenes.



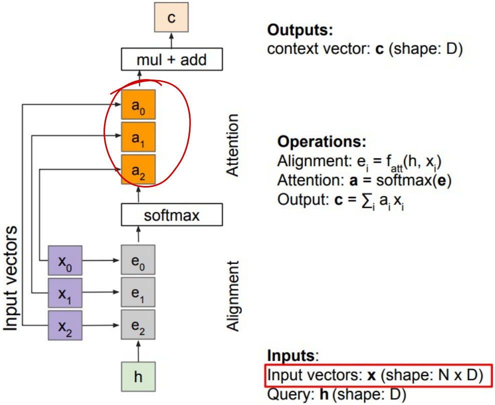
Outputs:
context vector: **c** (shape: D)

Operations:
Alignment: $e_{i,j} = f_{att}(h, z_{i,j})$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{c} = \sum_{i,j} a_{i,j} z_{i,j}$

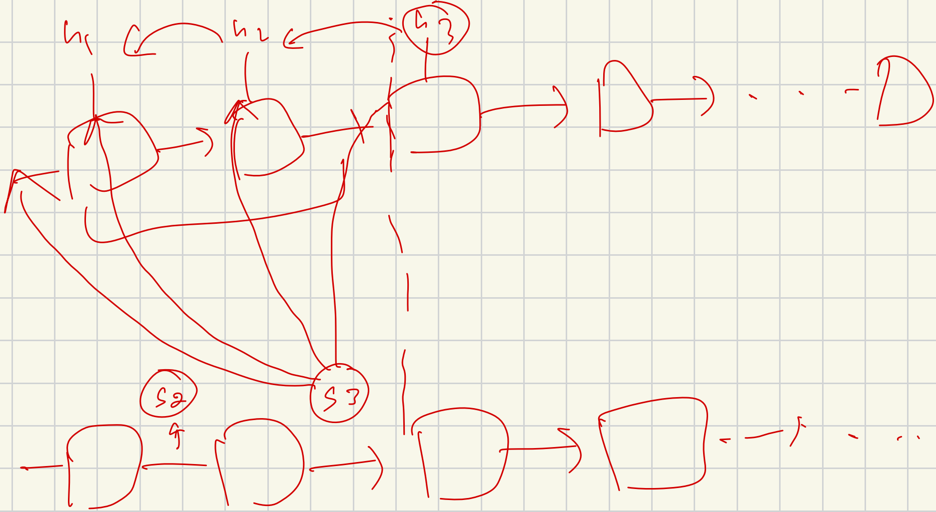
Inputs:
Features: **z** (shape: H x W x D)
Query: **h** (shape: D)

General Attention Layer

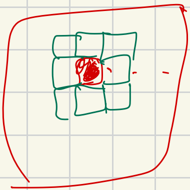
أكاديمية كاوست
KAUST ACADEMY



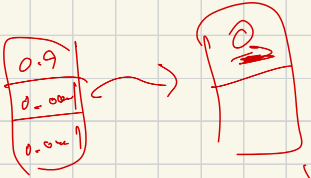
- The **attention** operation is permutation invariant.
- It does not depend on the ordering of features.
- Reshape the input of size $H \times W = N$ into N feature vectors.



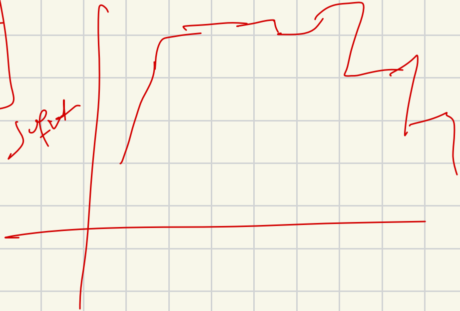
h1
h2
h3
s2
s3

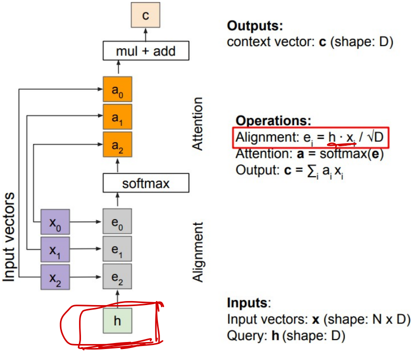


Attention → high

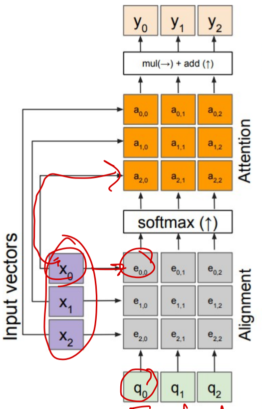


soft





- The original attention mechanism $f_{att}(.)$ uses a simple dot product to compute similarity between queries and keys.
- As the feature dimension D increases $\rightarrow$ the dot product sum involves more terms $\rightarrow$ larger variance in the resulting logits.
- Larger magnitude vectors produce higher logits $\rightarrow$ causing the softmax output to become more peaked (lower entropy, assuming logits are *IID*).
- This means the **model may focus too narrowly**, assigning very **high attention to a few positions** and **almost none to others**.
- To counteract this $\rightarrow$ we scale the dot product by dividing by $\sqrt{D}$.
- This normalization keeps the variance of the logits more consistent, resulting in a softer, more balanced attention distribution.



a_0 x_0

Outputs:
context vectors: **y** (shape: D)

Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} x_i$

$e_2 = q_2 \cdot x_1$

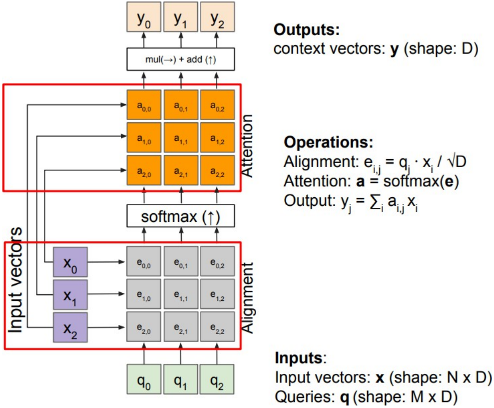
Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D)

- We can use multiple **query vectors** in the attention mechanism.
- Each query attends to the input independently, producing its own output context vector.
- This allows the model to extract different types of information from the same input, enabling richer and more flexible representations.

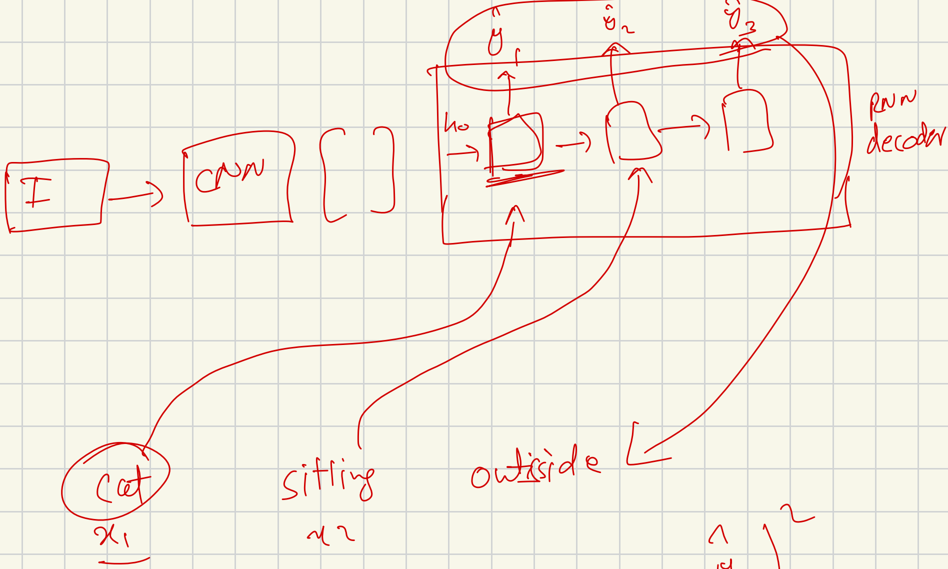
$\rightarrow e_1 = (q_0 \cdot x_0)/\sqrt{D}$

$\rightarrow e_2 = (q_1 \cdot x_1)/\sqrt{D}$

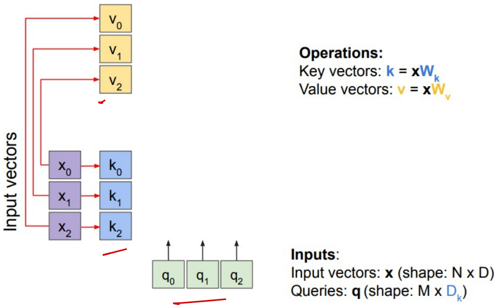
General Attention Layer (cont.)



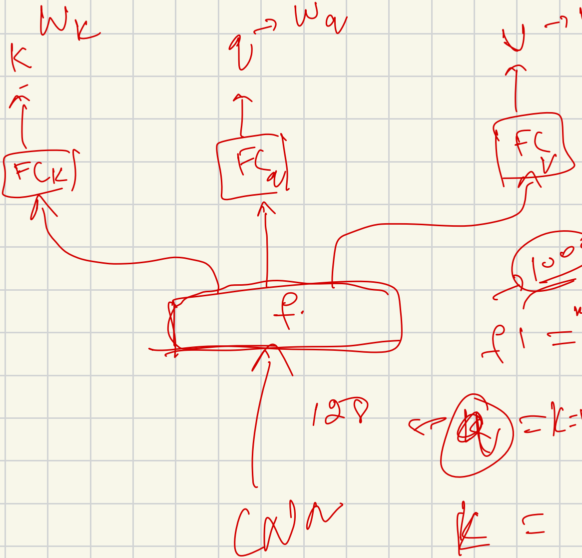
- Observe that the same input vectors are used for both computing the alignment scores (queries and keys) and for generating the attention-weighted output (values).
- This dual use lets the model efficiently use the same features for both attending and aggregating information.


I
CNN
h_0
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
RNN decoder
cat
sitting
outside
x_1
x_2

$$(y-\hat{y})^2$$



- Notice that the same input vectors are used for both computing alignment scores (as queries and keys) and for generating the attention-weighted output (as values).
- To increase the expressiveness of the attention layer, we can introduce separate fully connected (FC) layers before each step → one for queries, one for keys, and one for values.
- This allows the model to learn different transformations for each role, enabling richer and more flexible representations.



Projection

1000

model.

f_1 = pretrained(vgg16).fn

$Q = K = V = FC_1(f_1)$

$K = FC(f_1)$

FCa

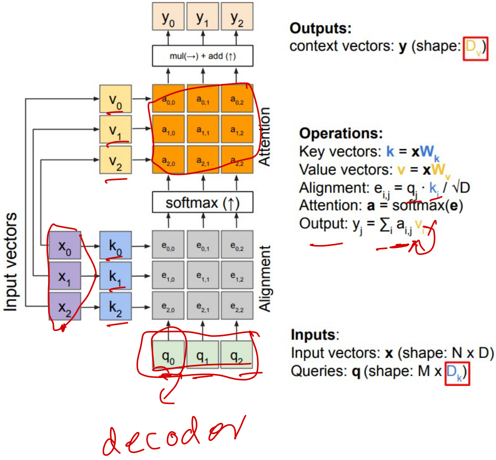
nn.Linear(1000, ~~512~~ 128)

.nn) -- (512, 256)

.nn.line (256, 128)

FCb nn.Line (1000, 128)

General Attention Layer (cont.)



- The input vectors are used for both computing alignment scores (as queries and keys) and for generating the attention-weighted output (as values).
- To enhance the expressiveness of the attention layer, we introduce separate fully connected (FC) layers for queries, keys, and values.
- Each FC layer can learn a different transformation, allowing the model to capture more complex relationships.
- With this setup, the input and output dimensions can differ, depending on the transformations applied by the key and value FC layers.

Attention

Cross attention

↳ query vector → decoder.

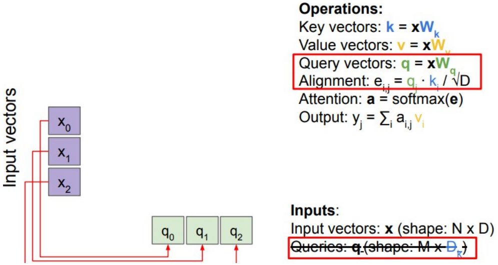
self attention

q k value

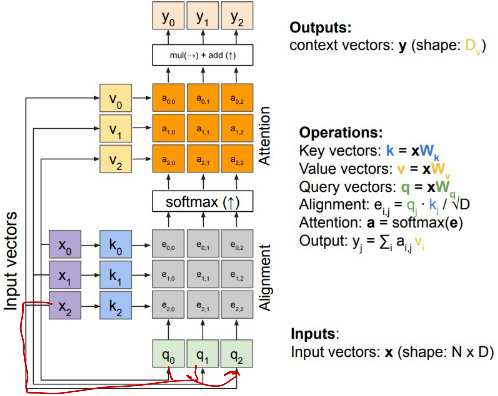
⇓

same

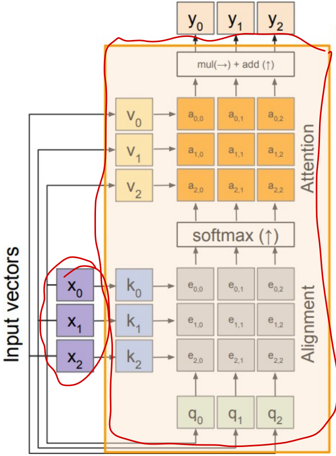
Ali's 2



- Recall: the query vector is derived from the input vectors.
- In a self-attention layer, query, key, and value vectors are all computed from the same input.
- There are no separate input query vectors; instead, they are generated internally.
- Typically, fully connected (FC) layers are used to compute the query, key, and value vectors from the input.
- This allows each position in the input to attend to all other positions, enabling the model to capture contextual relationships.



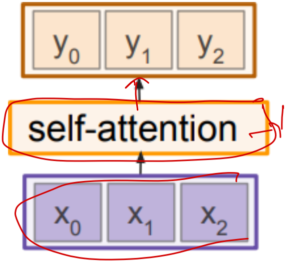
- Recall: the query vector is derived from the input vectors.
- In a self-attention layer, query, key, and value vectors are all computed from the same input.
- There are no separate input query vectors; instead, they are generated internally.
- Typically, fully connected (FC) layers are used to compute the query, key, and value vectors from the input.
- This allows each position in the input to attend to all other positions, enabling the model to capture contextual relationships.



Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$
Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)



$$\underbrace{Q_w = K_w = V_w}_{\text{projection}} = \text{self.FC}(\underline{f_1}).$$

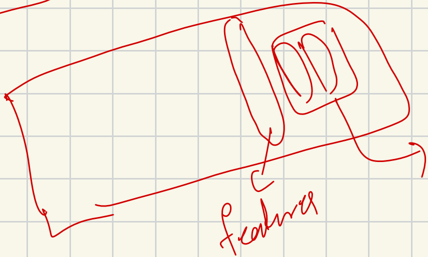
$f_1 = \text{VGG19}(\text{Img})$

$$\text{Alignment} = Q \cdot K^T / \sqrt{D}$$

$$\text{Attention} = \text{Softmax}(\text{Alignment})$$

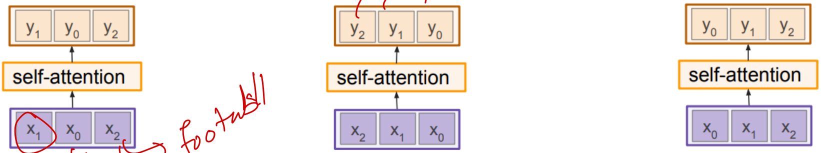
$$\boxed{\text{Output}} = (\text{Attention} \cdot V_w)$$

class CustAtt



Attention mechanism

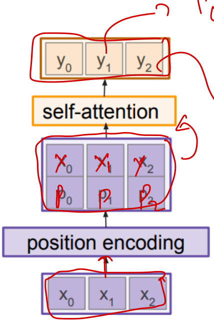
Permutation Invariance



- The self-attention layer is **permutation equivariant**: it produces the same output regardless of the order of the input elements.
- This means the model does not inherently capture the order of the sequence.
- **Challenge:** For tasks involving ordered data, such as language or images, we need a way to encode positional information.
- *How can we enable the model to distinguish between different positions in a sequence?*

Permutation Invariance

- **Positional Encoding:** Supplies sequence order information to models that lack recurrence or convolution, such as Transformers.
- Concatenate or add a special positional encoding p_j to each input vector x_j.
- A function $\mathrm{pos} : \mathbb{N} \to \mathbb{R}^D$ maps the position j of the vector into a D-dimensional vector, i.e., $p_j = \mathrm{pos}(j)$.



love

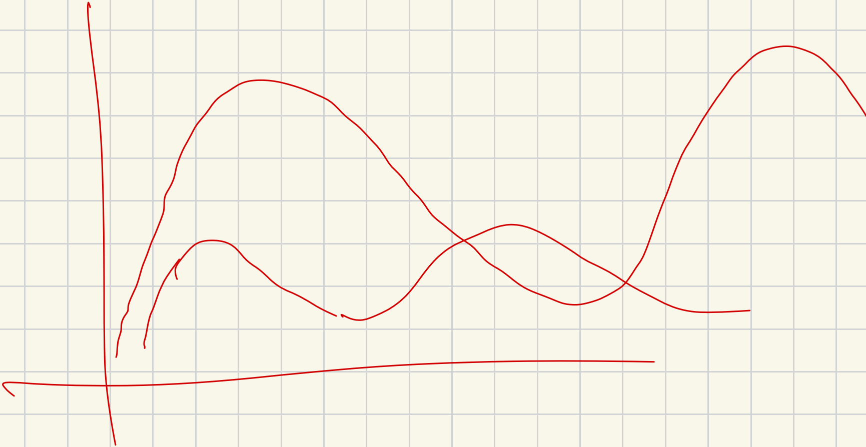
Football

Permutation Invariance (cont.)

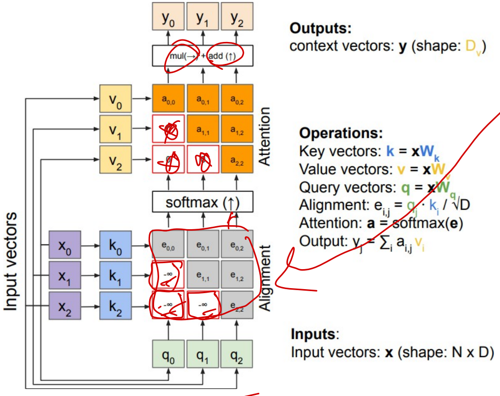
- The positional encoding is added to the input vectors before computing the attention scores.
- This allows the model to incorporate information about the position of each vector in the sequence.
- The positional encoding can be learned or fixed, depending on the implementation.
- The choice of positional encoding can affect the model's ability to capture long-range dependencies and relationships in the data.
- **Sinusoidal encodings:** Common fixed positional encoding uses sine and cosine functions:

$$\mathrm{pos}(j)_{2k} = \sin\left(\frac{j}{10000^{2k/d}}\right)$$

$$\mathrm{pos}(j)_{2k+1} = \cos\left(\frac{j}{10000^{2k/d}}\right)$$



Masked Self-Attention Layer



$$\text{Mask} = \begin{bmatrix} 1 & 1 & 1 \\ -\infty & 1 & 1 \\ -\infty & -\infty & 1 \end{bmatrix}$$

- Prevents each position from attending to subsequent (future) positions.
- Achieved by setting (manually) alignment scores of future tokens to $-\infty$ before softmax.
- Ensures predictions for position i depend only on positions $\leq i$.

I → love

$q_1 \cdot k_1$ (I ↔ I) =

$\boxed{q_2 \cdot k_2}$ (I ↔ love) = 1<00

$q_3 \cdot k_3$ (I ↔ football) =

q_3

k_1

q_1 k_1 v_1 — q_2 k_2 v_2 — q_3 k_3 v_3

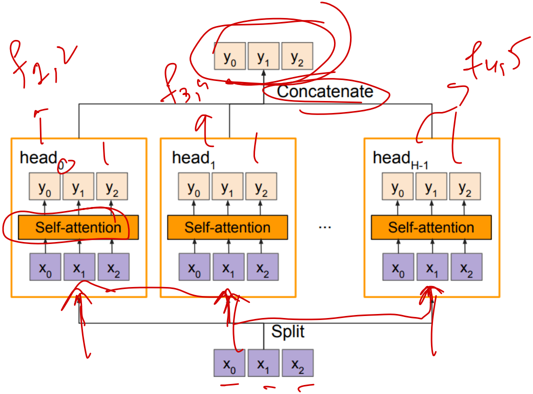
x_1 — x_2 — x_3

I — love — football

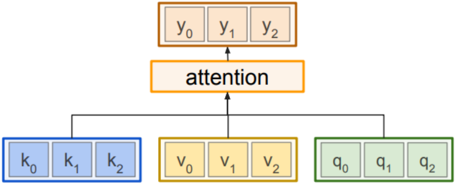
$q_1 \cdot k_1$	$q_2 \cdot k_1$	$q_3 \cdot k_1$
$q_1 \cdot k_2$	$q_2 \cdot k_2$	$q_3 \cdot k_2$
$q_1 \cdot k_3$	$q_2 \cdot k_3$	$q_3 \cdot k_3$

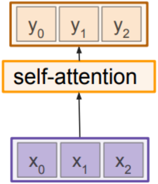
$q_3 k_5$

$q_3 k_5$



- Instead of a single attention mechanism, use multiple “heads” operating in parallel.
- Each head attends to different parts or aspects of the input, capturing diverse relationships (e.g., subject-verb, coreferences).
- The outputs from all heads are concatenated and projected, resulting in a richer and more expressive context representation.
- This modular approach allows the model to learn various types of dependencies simultaneously, improving overall performance.

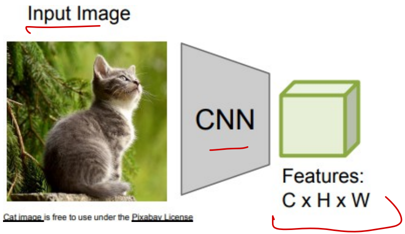


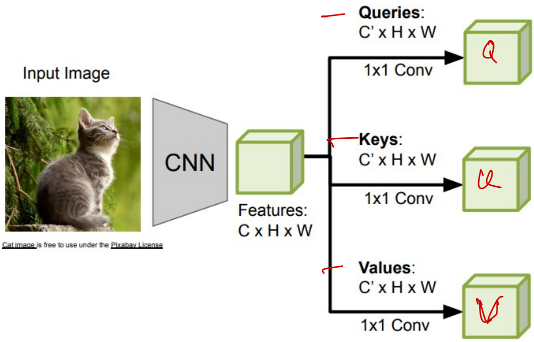


Comparison	**General Attention**	**Self-Attention**
Q, K, V origins	From separate source & target	From same input sequence
Use case	Encoder $\rightarrow$ Decoder cross-attention	Encoder/Decoder internal relation info
Information flow	Across representations	Within single representation

Table 1: General Attention vs. Self-Attention

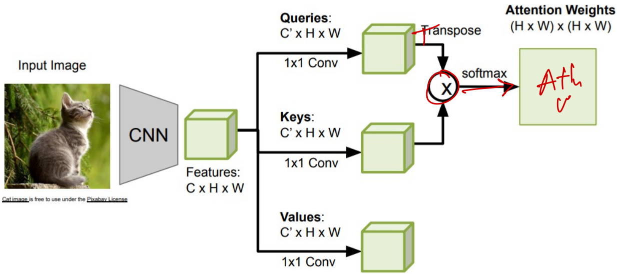
Example: CNN with Self-Attention



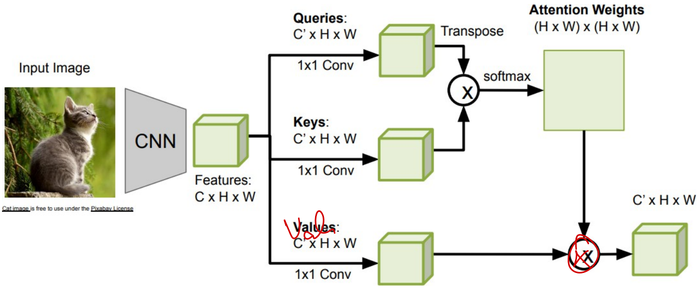


Cat image is free to use under the Pixabay License

Example: CNN with Self-Attention (cont.)

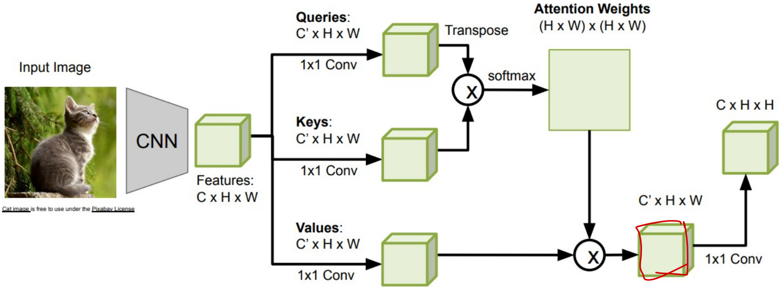


Example: CNN with Self-Attention (cont.)



Cat image is free to use under the Pixabay License

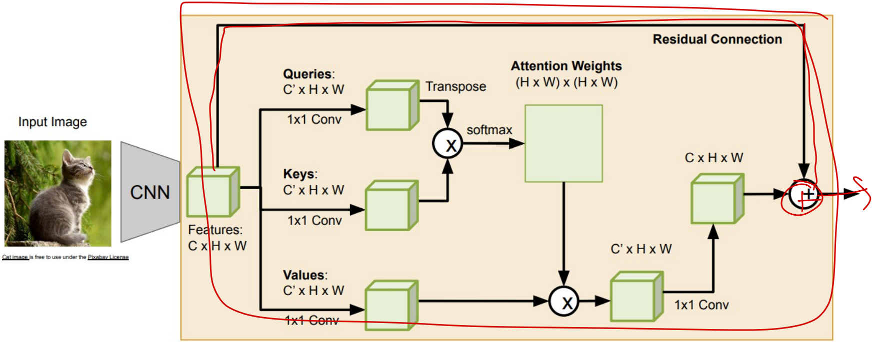
Example: CNN with Self-Attention (cont.)



Cat image is free to use under the Pixabay License

Example: CNN with Self-Attention (cont.)

Mul + Add



Cat image is free to use under the Pixabay License

Self-Attention Module

Attention is all you need

Vaswani et al, NeurIPS 2017

Transformers (cont.)

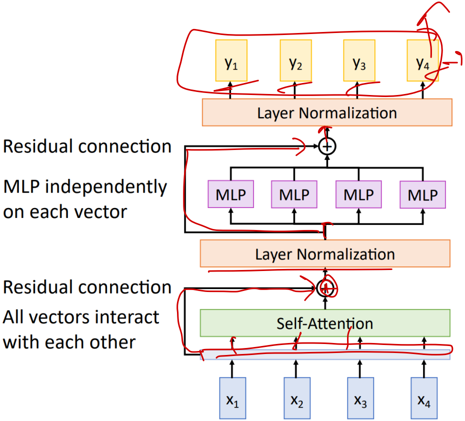


Figure 2: Detailed view of a single Transformer block.

- **Input:** A set of vectors **x**, one per token.
- **Output:** A set of vectors **y**, one per token.
- **Self-Attention:** Allows each token to attend to all others, capturing contextual relationships.
- **Layer Normalization & MLP:** Applied independently to each token, ensuring stability and expressiveness.
- **Key Properties:** Highly scalable and parallelizable due to independent operations across tokens.

Transformers (cont.)

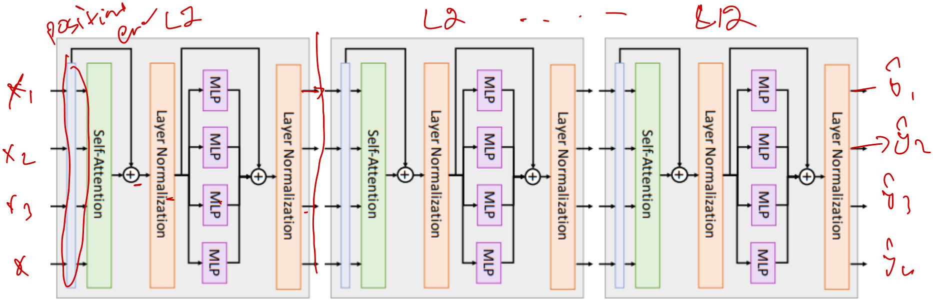


Figure 3: Stacking multiple Transformer blocks.

- A Transformer consists of a sequence of identical blocks, each refining the token representations.
- In the original architecture (Vaswani et al., 2017): 12 blocks, hidden size $D_Q = 512$, and 6 attention heads were used.
- The modular design enables deep stacking and efficient parallel computation.

Summary

- Transformers use self-attention to capture relationships within sequences.
- Self-attention allows each token to attend to all other tokens, enabling context-aware representations.
- Positional encoding is crucial for maintaining the order of tokens in the sequence.
- Masked self-attention prevents future information leakage during training.
- Multi-head attention enhances model capacity by allowing multiple attention mechanisms to learn different aspects of the data.

References

[1] Fei-Fei Li, Yunzhu Li, and Ruohan Gao. *Stanford CS231n: Deep Learning for Computer Vision*. http://cs231n.stanford.edu/index.html

[2] Assaf Shocher, Shai Bagon, Meirav Galun, and Tali Dekel. *WAIC DL4CV Deep Learning for Computer Vision: Fundamentals and Applications*. https://dl4cv.github.io/index.html

[3] Justin Johnson. *UMich EECS 498.008/598.008: Deep Learning for Computer Vision*. https://web.eecs.umich.edu/~justincj/teaching/eecs498/WI2022/

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. Advances in Neural Information Processing Systems, 2017. https://arxiv.org/abs/1706.03762

[5] Stanford CS231n. *Lecture 11: Attention & Transformers.* http://cs231n.stanford.edu/slides/2023/lecture11.pdf

[6] Carnegie Mellon University. *Deep Learning Lecture: Vision Transformers and Transformers.* https://deeplearning.cs.cmu.edu/F23/index.html https://www.youtube.com/watch?v=1gq2y1E5g6M

[7] MIT 6.S191. *Introduction to Deep Learning Resources.* https://introtodeeplearning.com/

[8] Satya Mallick. *LearnOpenCV: A Comprehensive Guide to Attention Mechanisms in Deep Learning.* https://learnopencv.com/attention-mechanism-in-neural-networks/

Credits

Dr. Prashant Aparajeya

Computer Vision Scientist — Director(AISimply Ltd)

p.aparajeya@aisimply.uk

This project benefited from external collaboration, and we acknowledge their contribution with gratitude.