# CAP 5516
# Medical Image Computing
# (Spring 2025)


**Dr. Chen Chen**
**Associate Professor**
**Center for Research in Computer Vision (CRCV)**
**University of Central Florida**
**Office: HEC 221**
**Email: chen.chen@crcv.ucf.edu**
**Web: https://www.crcv.ucf.edu/chenchen/**

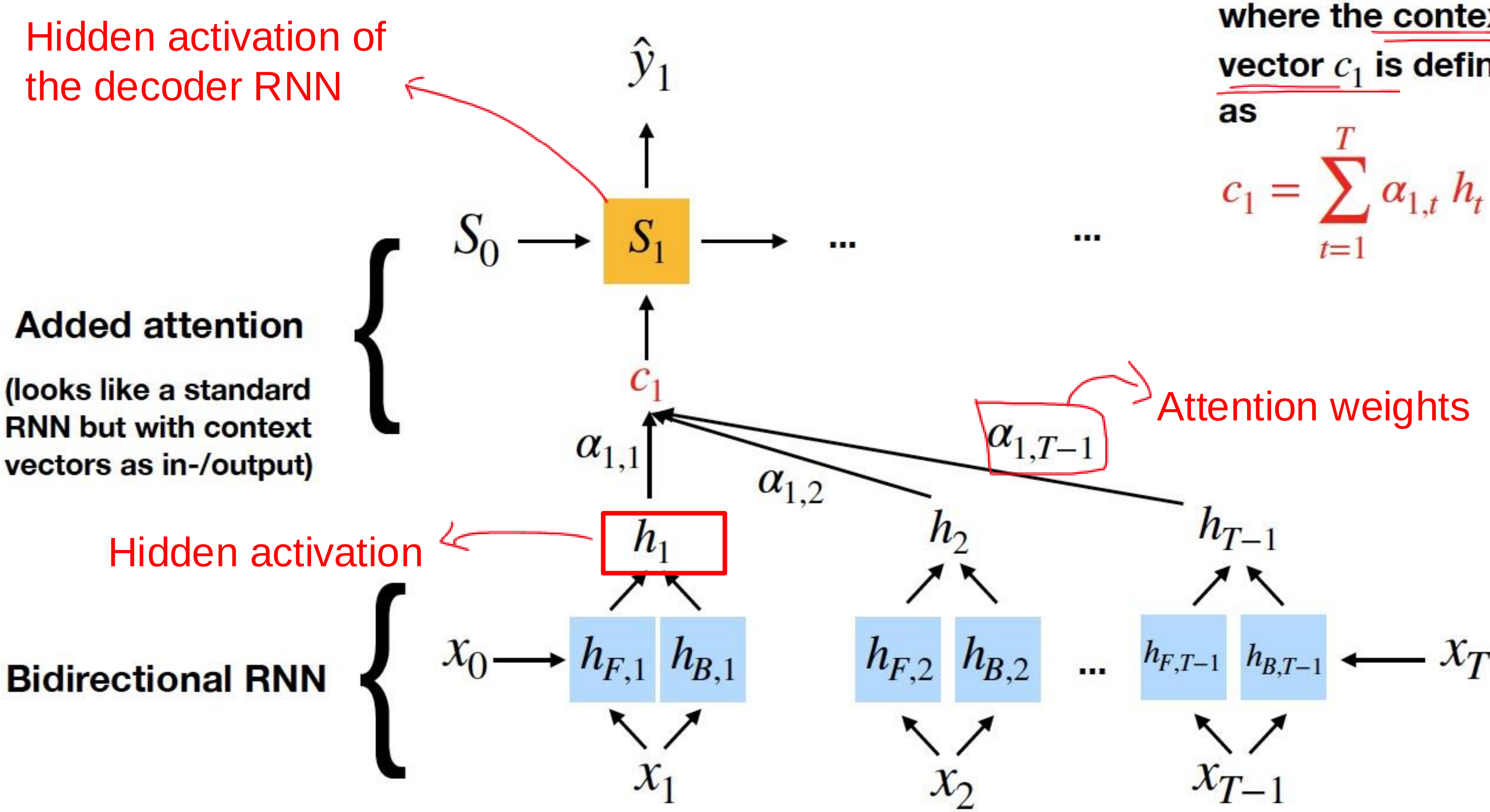

UCF CENTER FOR RESEARCH IN COMPUTER VISION

# Lecture 8: Introduction to Deep Learning (3)

# Attention Mechanism

- The attention mechanism is a technique used to selectively focus on the most relevant parts of an input, rather than using the entire input equally, when producing a prediction.
- It allows a network to dynamically weigh the importance of different elements in the input and focus on the most relevant ones when making a decision.
- The attention mechanism has proven to be particularly effective in tasks such as machine translation and image captioning, where a model must consider different parts of an input when making a prediction.

# RNN with Attention Mechanism

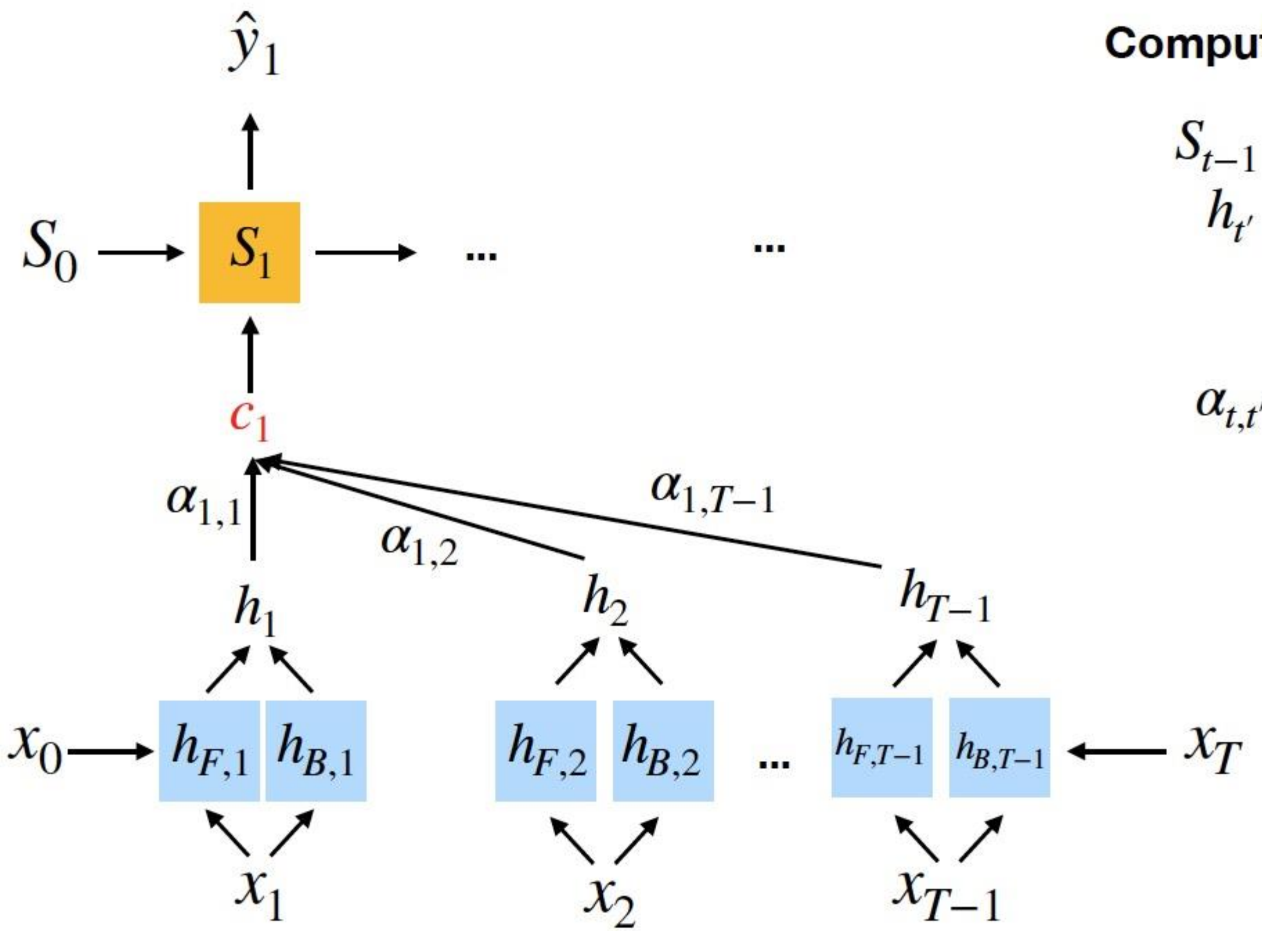



where the context vector $c_1$ is defined as

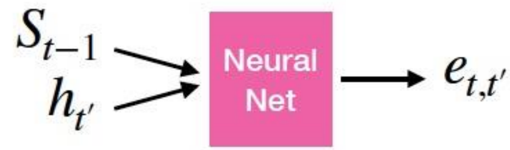
$$c_1 = \sum_{t=1}^{T} \alpha_{1,t}\, h_t$$

Credit: Sebastian Raschka

# RNN with Attention Mechanism




**Computing attention weights**




$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$

Credit: Sebastian Raschka

# RNN with Attention Mechanism

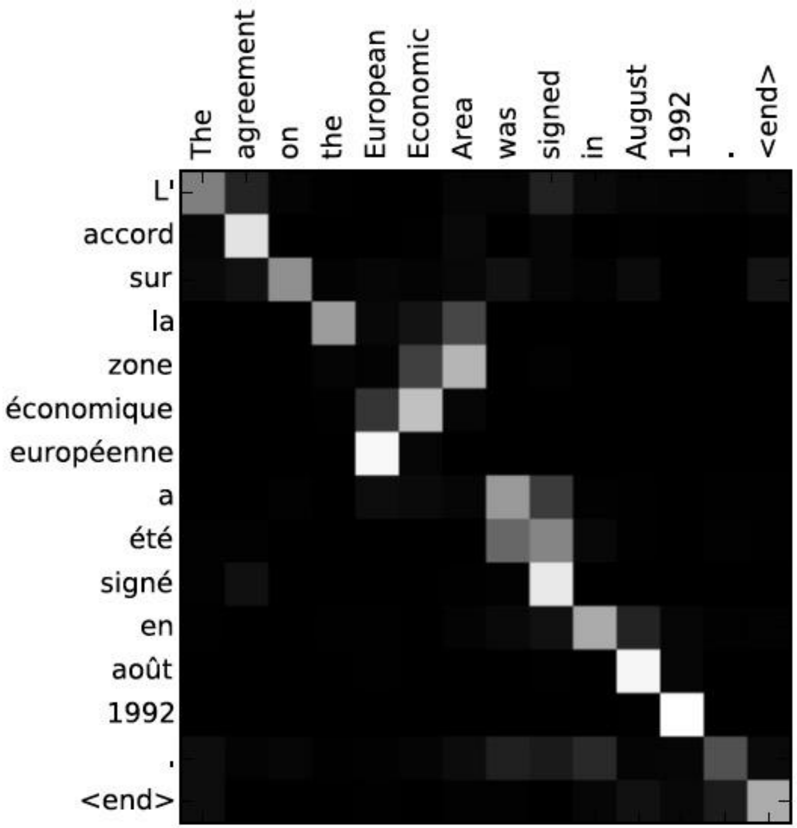
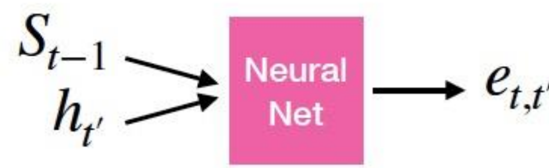



Figure: Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. https://arxiv.org/abs/1409.0473

## Computing attention weights

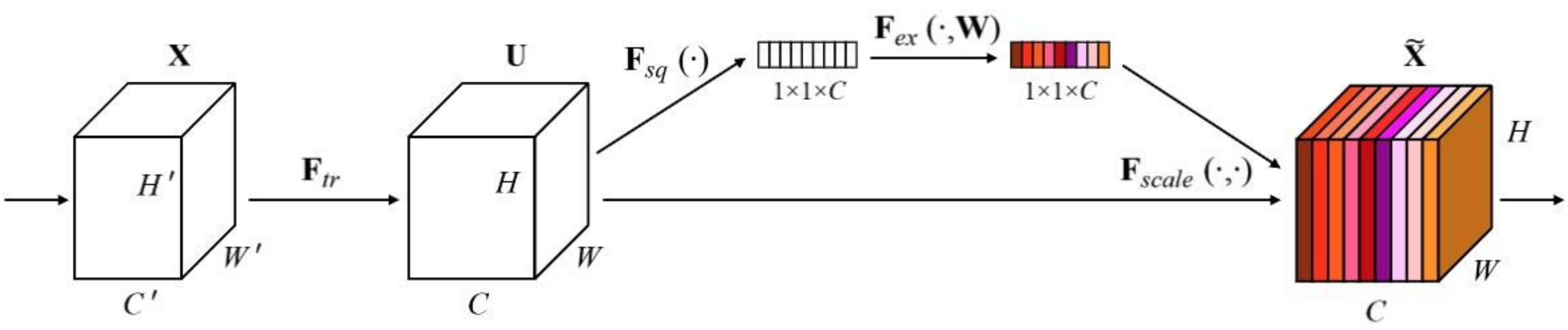



$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$

Credit: Sebastian Raschka

# Attention in CNNs

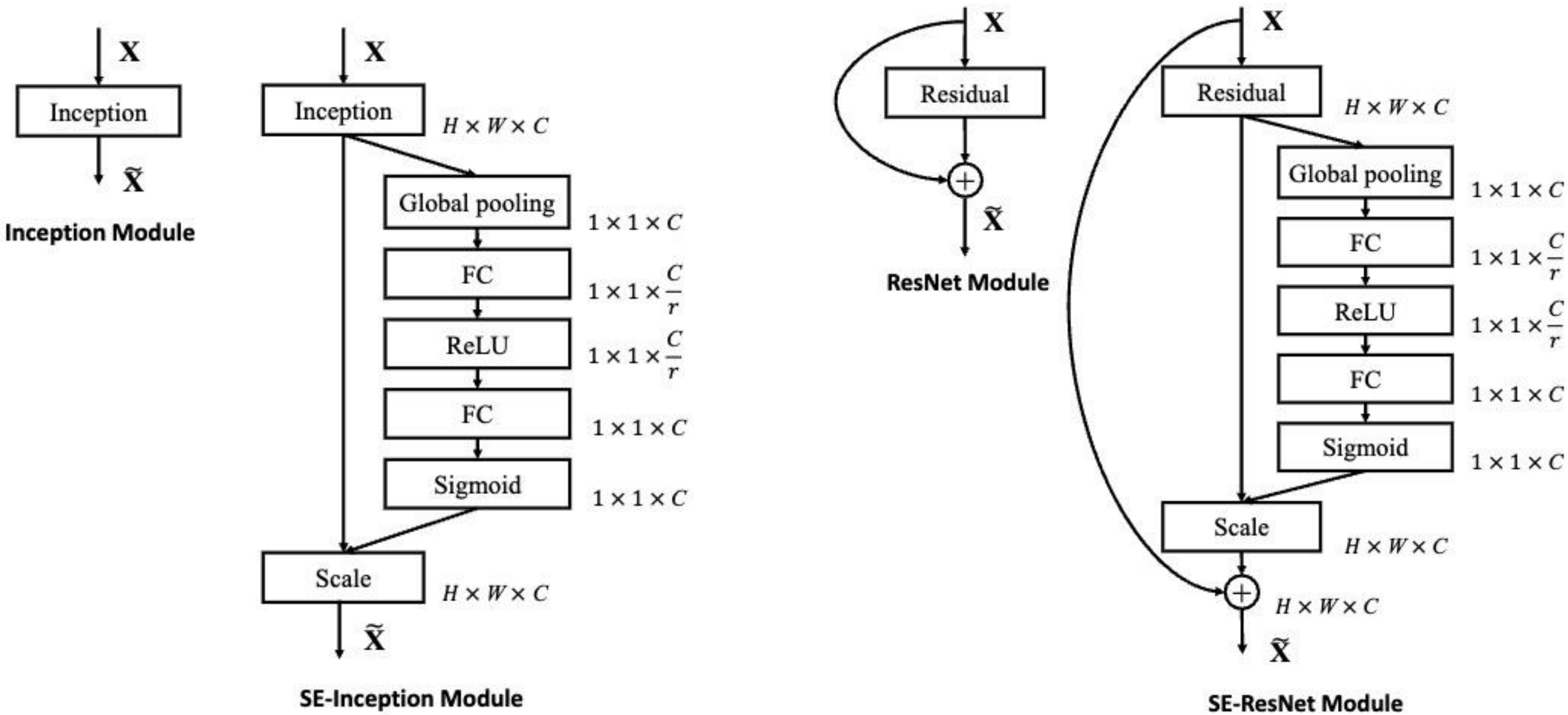
- Squeeze-and-Excitation Networks




Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7132-7141).

# Attention in CNNs

- Squeeze-and-Excitation Networks




Hu, J., Shen, L., & Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7132-7141).

# Transformer

# Attention Is All You Need

**Ashish Vaswani*** Google Brain avaswani@google.com

**Noam Shazeer*** Google Brain noam@google.com

**Niki Parmar*** Google Research nikip@google.com

**Jakob Uszkoreit*** Google Research usz@google.com

**Llion Jones*** Google Research llion@google.com

**Aidan N. Gomez*** † University of Toronto aidan@cs.toronto.edu

**Łukasz Kaiser*** Google Brain lukaszkaiser@google.com

**Illia Polosukhin*** ‡ illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

https://arxiv.org/abs/1706.03762

**Attention** is **all you need**

A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

… to attend to **all** positions in the decoder up to and including that position. **We need** to prevent … **We** implement this inside of scaled dot-product **attention** by masking out (setting to −∞) …

# Since ~2018, Transformers have been growing in popularity ... and size

GPT-3 (**175B parameters**)

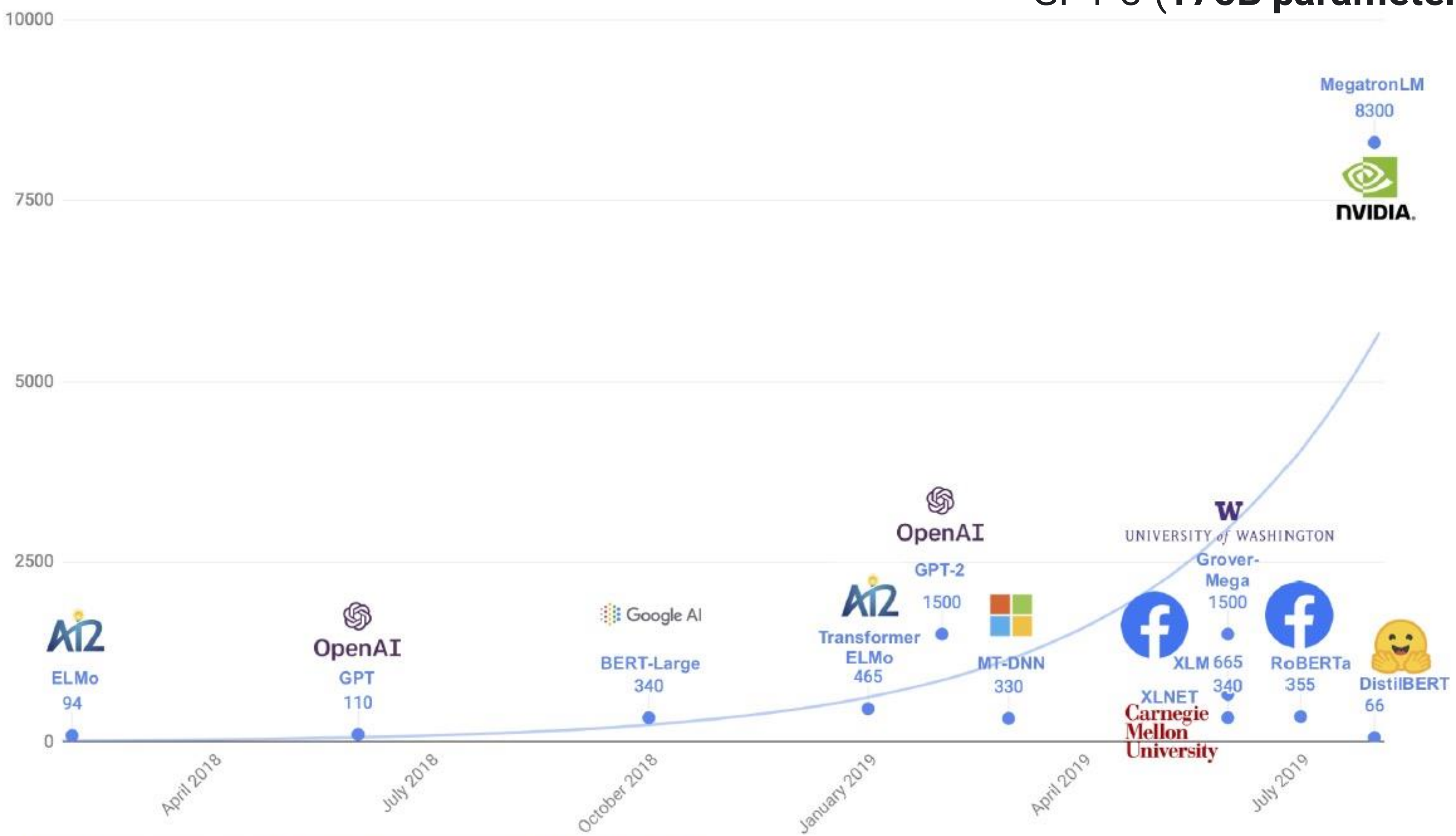
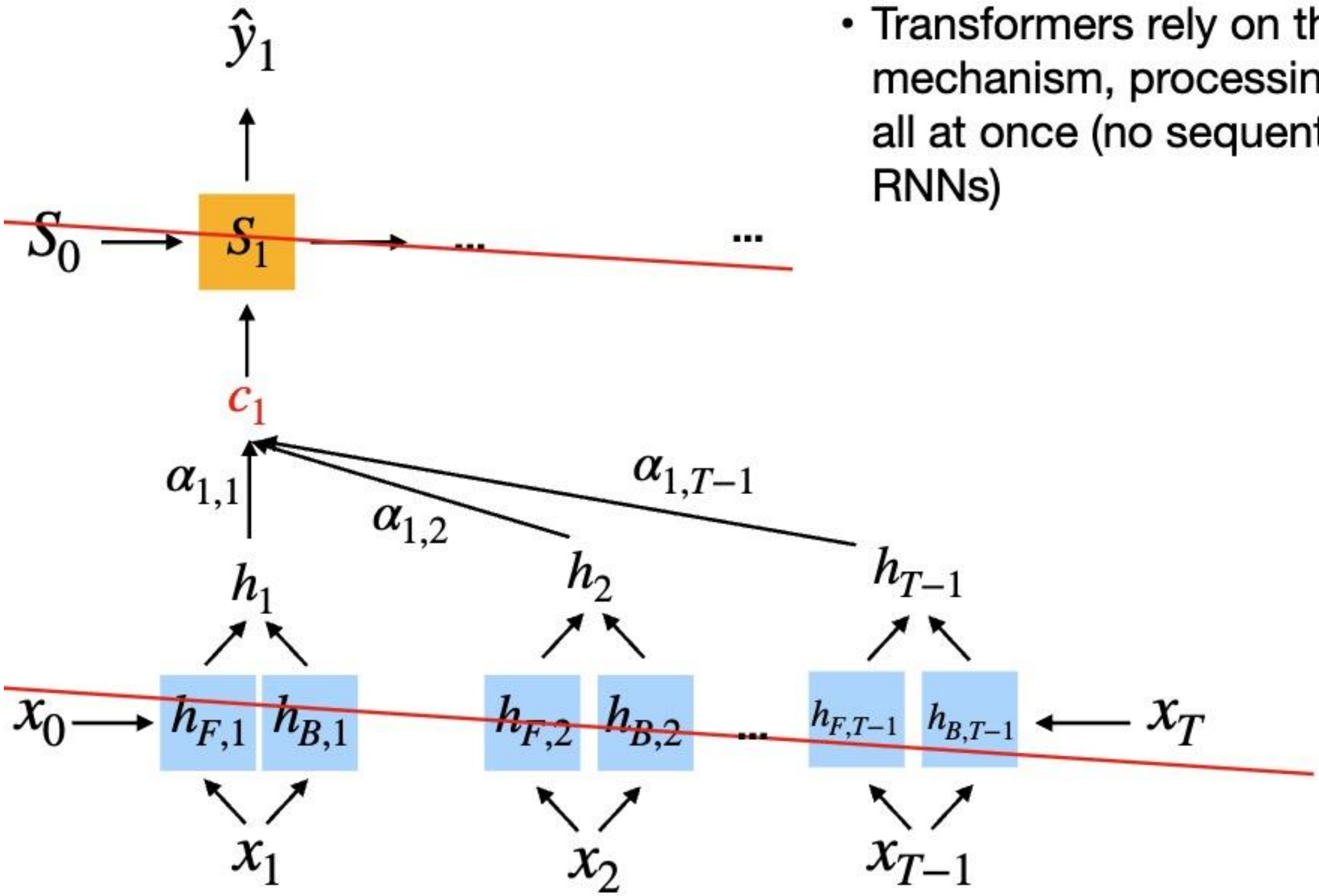



Image Source: https://medium.com/huggingface/distilbert-8cf3380435b5

# Getting rid of the sequential parts




- No recurrence, no convolution
- Transformers rely on the self-attention mechanism, processing the whole sequence all at once (no sequential processing like in RNNs)

Credit: Sebastian Raschka

# Self-Attention Mechanism -- Very Basic Form

**Main procedure:**

1) Derive attention <u>weights</u>: similarity between current input and all other inputs (next slide)
2) Normalize weights via softmax (next slide)
3) Compute attention value from normalized weights and corresponding inputs (below)

**Self-attention as weighted sum:**

$$\boldsymbol{A}_i = \sum_{j=1}^{T} a_{ij} \boldsymbol{x}_j$$

output corresponding to the i-th input

weight based on similarity between current input $\boldsymbol{x}_i$ and all other inputs



Credit: Sebastian Raschka

# Self-Attention Mechanism -- Very Basic Form

**Self-attention as weighted sum:**

$$\mathbf{A}_i = \sum_{j=1}^{T} a_{ij}\boldsymbol{x}_j$$

output corresponding to the i-th input

weight based on similarity between current input $\boldsymbol{x}_i$ and all other inputs

**How to compute the attention weights?**

here as simple dot product:

$$e_{ij} = \boldsymbol{x}_i^{\top}\boldsymbol{x}_j$$

repeat this for all inputs $j \in \{1...T\}$, then normalize

$$a_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{j=1}^{T}\exp\left(e_{ij}\right)} = \operatorname{softmax}\left(\left[e_{ij}\right]_{j=1\ldots.T}\right)$$

# Self-Attention Mechanism -- Very Basic Form

$\vec{a} \cdot \vec{b} = \|a\| \, \|b\| \cos\theta$

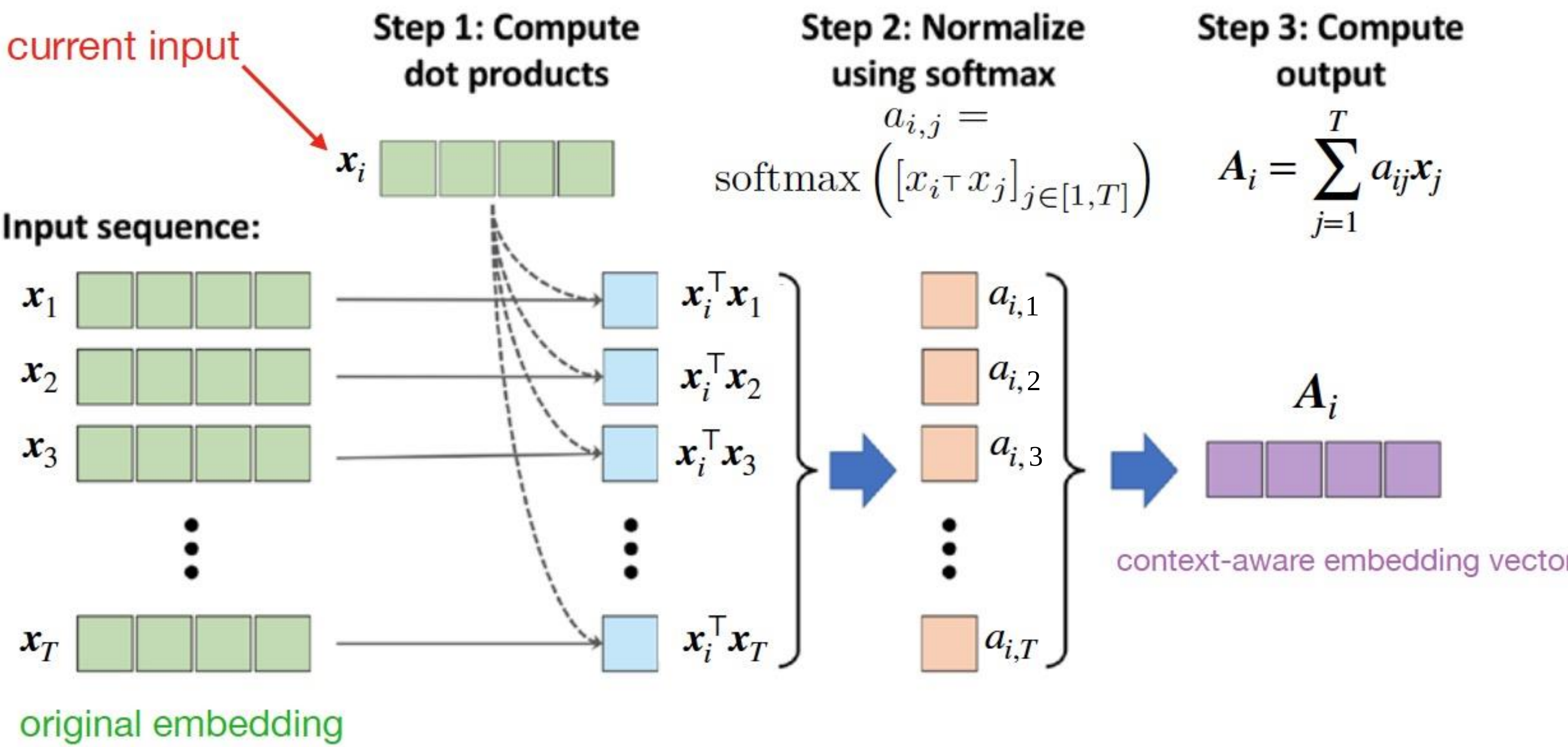



Image source: Raschka & Mirjalili 2019. Python Machine Learning, 3rd edition

# Attention Is All You Need

**Ashish Vaswani*** Google Brain avaswani@google.com

**Noam Shazeer*** Google Brain noam@google.com

**Niki Parmar*** Google Research nikip@google.com

**Jakob Uszkoreit*** Google Research usz@google.com

**Llion Jones*** Google Research llion@google.com

**Aidan N. Gomez*** † University of Toronto aidan@cs.toronto.edu

**Łukasz Kaiser*** Google Brain lukaszkaiser@google.com

**Illia Polosukhin*** ‡ illia.polosukhin@gmail.com

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

https://arxiv.org/abs/1706.03762

# Self-Attention Mechanism

- Previous basic version did not involve any learnable parameters, so not very useful for learning a language model

- We are now adding 3 trainable weight matrices that are multiplied with the input sequence embeddings ($\boldsymbol{x_i}$'s)

$$\text{query} = \underline{\boldsymbol{W}^q}\boldsymbol{x}_i$$
$$\text{key} = \underline{\boldsymbol{W}^k}\boldsymbol{x}_i$$
$$\text{value} = \underline{\boldsymbol{W}^v}\boldsymbol{x}_i$$



Credit: Sebastian Raschka

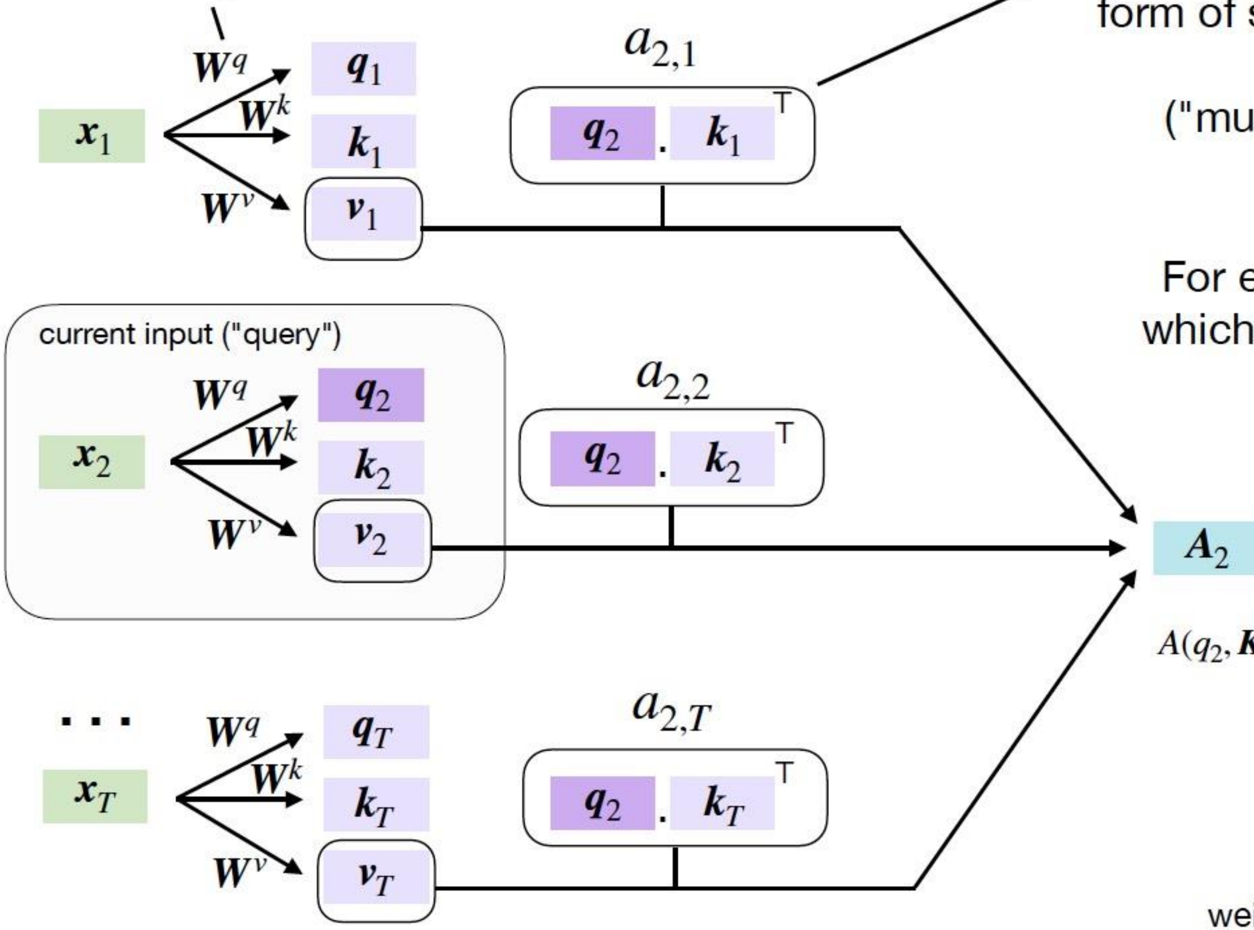
# Self-Attention Mechanism

1. **Query**:
- In the transformer, the **query** vector corresponds to the feature representation of the current token (or position) that is “asking” how much it should attend to other tokens in the input sequence.

2. **Key**:
- In the transformer, the **key** vector represents the features of a token (or position) in the sequence. It is used in conjunction with the query to determine the degree of relevance or attention between tokens.

3. **Value**:
- In the transformer, the **value** vector represents the actual feature content of a token (or position). Once attention weights are calculated using the query and key, these weights are applied to the value vectors to compute the weighted sum, which represents the output of the attention mechanism.

# Self-Attention Mechanism




As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")

For each query, model learns which **k**ey-**v**alue input it should attend to

$$A(q_2, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{T} \Big[ \underbrace{\frac{\exp(q_2 \cdot k_i^\top)}{\sum_j \exp(q_2 \cdot k_j^\top)}}_{\text{softmax}} \times \boldsymbol{v}_i \Big]$$

weighted sum: values weighted by attention weight (softmax score)

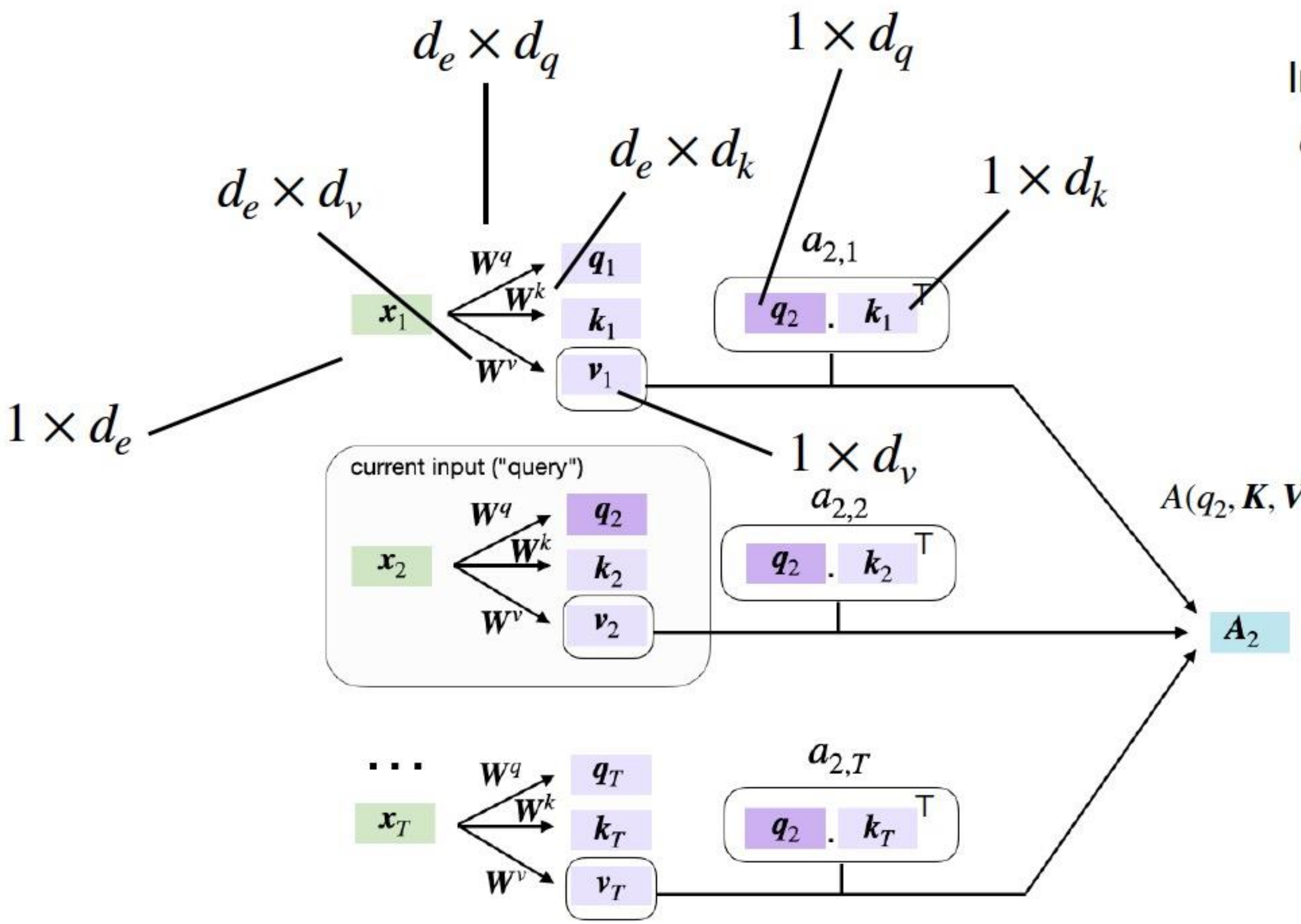
Credit: Sebastian Raschka

# Self-Attention Mechanism

$d_e$ = embedding size (original transformer = 512)

where $d_q = d_k$

In original transformer, $d_q = d_v$ as well




$$A(q_2, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{T} \Big[ \underbrace{\frac{\exp(q_2 \cdot k_i^\top)}{\sum_j \exp(q_2 \cdot k_j^\top)}}_{\text{softmax}} \times \boldsymbol{v}_i \Big]$$

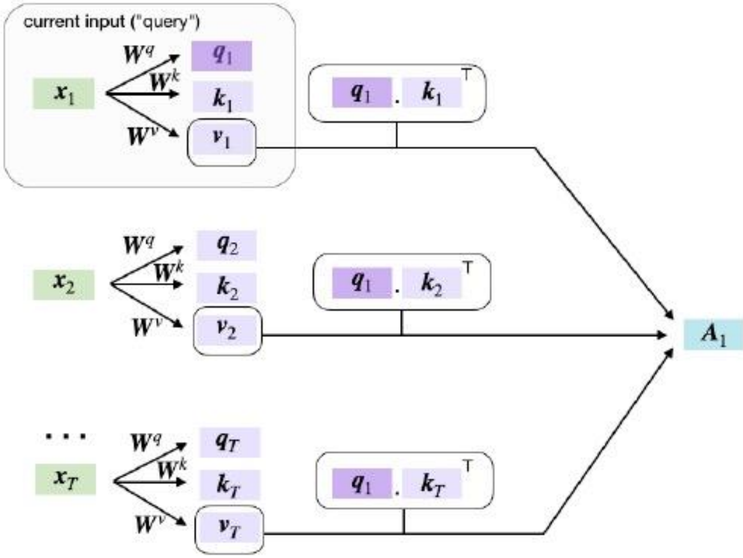
$1 \times 1$

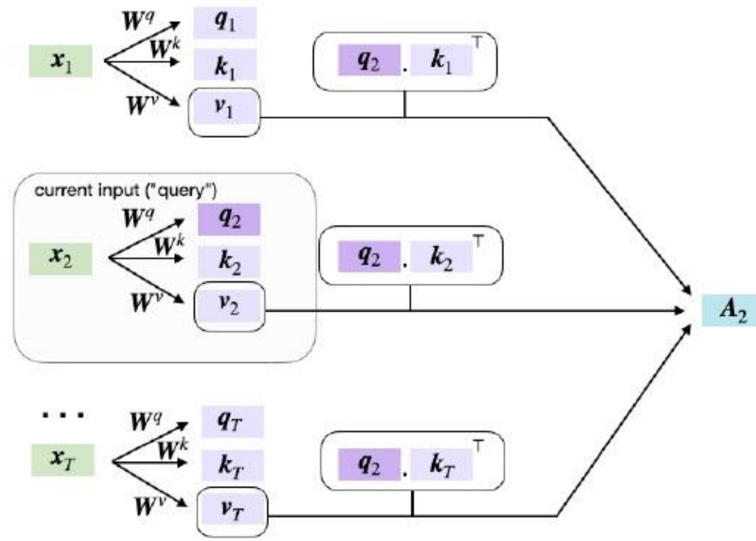
$1 \times d_v$



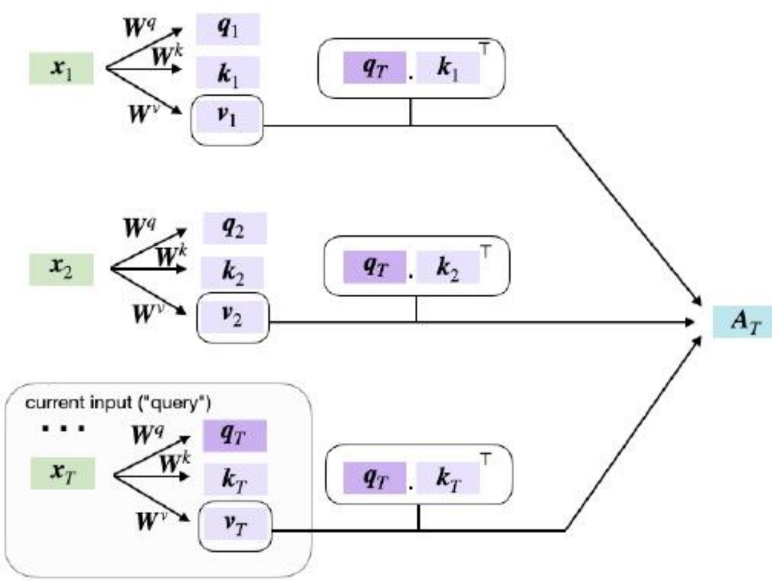
Credit: Sebastian Raschka

# Self-Attention Mechanism




• • •




Computed in parallel

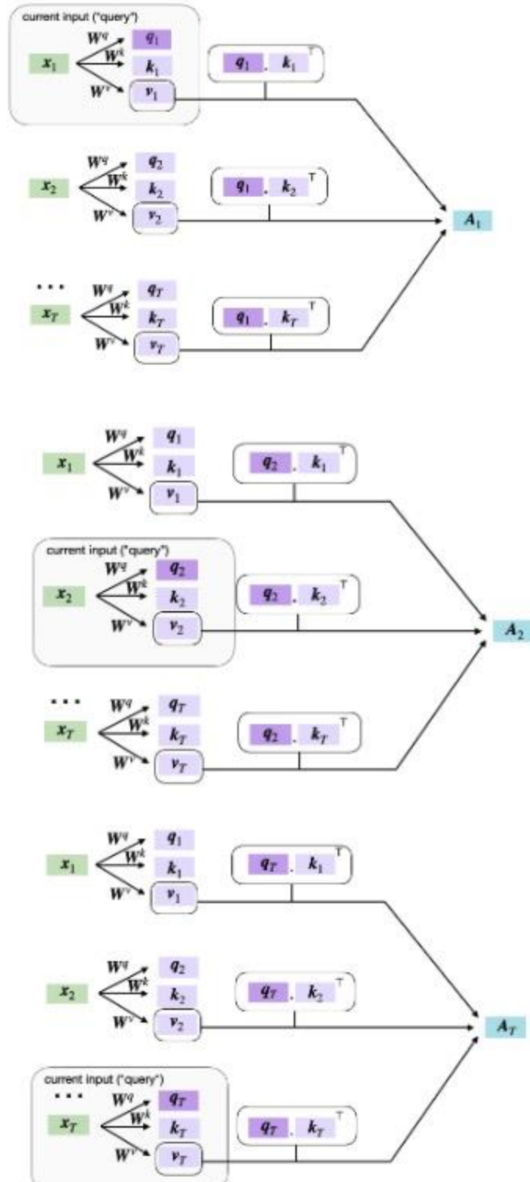
Attention score matrix: $$A = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_T \end{bmatrix}$$

# Self-Attention Mechanism

$d_e$ = embedding size

$T$ = input sequence size

$\boldsymbol{x} \in \mathbb{R}^{T \times d_e}$




$\boldsymbol{Q} \in \mathbb{R}^{T \times d_q}$

$\boldsymbol{K} \in \mathbb{R}^{T \times d_k}$

$\boldsymbol{V} \in \mathbb{R}^{T \times d_v}$

"attention matrix"

$$A(Q, K, V) = \underbrace{\overbrace{\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)}^{T \times T} V}_{T \times d_v}$$

"attention-based embedding"



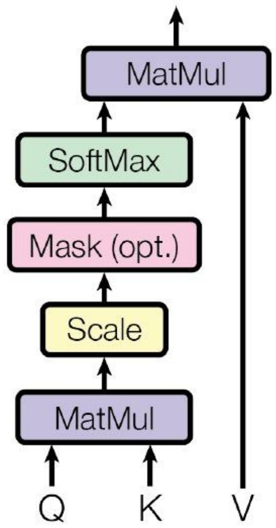
Credit: Sebastian Raschka

# Scaled Dot Product Attention

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V$$

To ensure that the dot-products between query and and key don't grow too large (and softmax gradient become too small) for large $d_k$

Scaled Dot-Product Attention




Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

“ We suspect that for large values of dk, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. ”

# Summary of self-attention mechanism

**Purpose of $W_q$, $W_k$, and $W_v$ :**

- The input embeddings (or features) of tokens in the sequence are often high-dimensional vectors that encode raw information. However, for attention mechanisms to compute meaningful relationships, these embeddings need to be projected into separate subspaces tailored for the specific roles of **query**, **key**, and **value**.

UCF Center for Research in Computer Vision

# Summary of self-attention mechanism

$W_q$ **(Query projection matrix):**

- Projects the input embeddings into the **query** space.
- This determines how much attention a token will “query” (or ask for) from other tokens.

$W_k$ **(Key projection matrix):**

- Projects the input embeddings into the **key** space.
- This determines how the token contributes to the relevance calculation with respect to a query.

$W_v$ **(Value projection matrix):**

- Projects the input embeddings into the **value** space.
- This determines what information a token provides when it is attended to.

# Summary of self-attention mechanism

**Learning Role:**

- The weight matrices $W_q$, $W_k$, and $W_v$ are learned during the training process through backpropagation. Their purpose is to:
  - Capture and encode different aspects of relationships between tokens in the input sequence.
  - Adapt the model to the specific patterns and dependencies present in the data.
- Having separate weight matrices enhances the ability of the model to encode nuanced relationships between tokens.

# Multi-Head Attention

- Apply self-attention multiple times in parallel (similar to multiple kernels for channels in CNNs)
- For each head (self-attention layer), use different $\boldsymbol{W}^q, \boldsymbol{W}^k, \boldsymbol{W}^v$ , then concatenate the results, $\boldsymbol{A}_{(i)}$
- 8 attention heads in the original transformer, i.e.,
$\boldsymbol{W}^q_{(1)}, \boldsymbol{W}^k_{(1)}, \boldsymbol{W}^v_{(1)} \dots \boldsymbol{W}^q_{(8)}, \boldsymbol{W}^k_{(8)}, \boldsymbol{W}^v_{(8)}$

# Multi-Head Attention

- Intuition

**Capturing Different Types of Relationships:**

- **Single-head attention** may focus on a limited set of features or relationships between tokens because it computes a single attention distribution.
- **Multi-head attention** allows the model to learn multiple attention distributions simultaneously, each focusing on different aspects of the input sequence.
- For example, one head might focus on long-term dependencies, while another focuses on local context.

# Multi-Head Attention

- Intuition

**Learning Diverse Representations:**

- Each head has its own set of learnable weight matrices ( $W_q$ , $W_k$ , $W_v$ ), allowing it to project the input embeddings into different subspaces.
- This enables each head to attend to different features or patterns in the input, leading to a richer and more nuanced representation.

# Multi-Head Attention

- Intuition

**Mitigating Information Loss:**

- With a single head, compressing all information into one set of attention scores may lose certain details.
- Multi-head attention distributes the representational capacity across multiple attention heads, reducing the risk of losing important information.

# Multi-Head Attention

- Intuition

**Improving Expressiveness:**

- By allowing multiple independent attention mechanisms to run in parallel, the model can express a broader range of dependencies and capture more complex interactions in the data.
- For instance, in language modeling, one head might focus on syntactic relationships (e.g., subject-verb agreement), while another focuses on semantic relationships (e.g., thematic roles).

# Multi-Head Attention

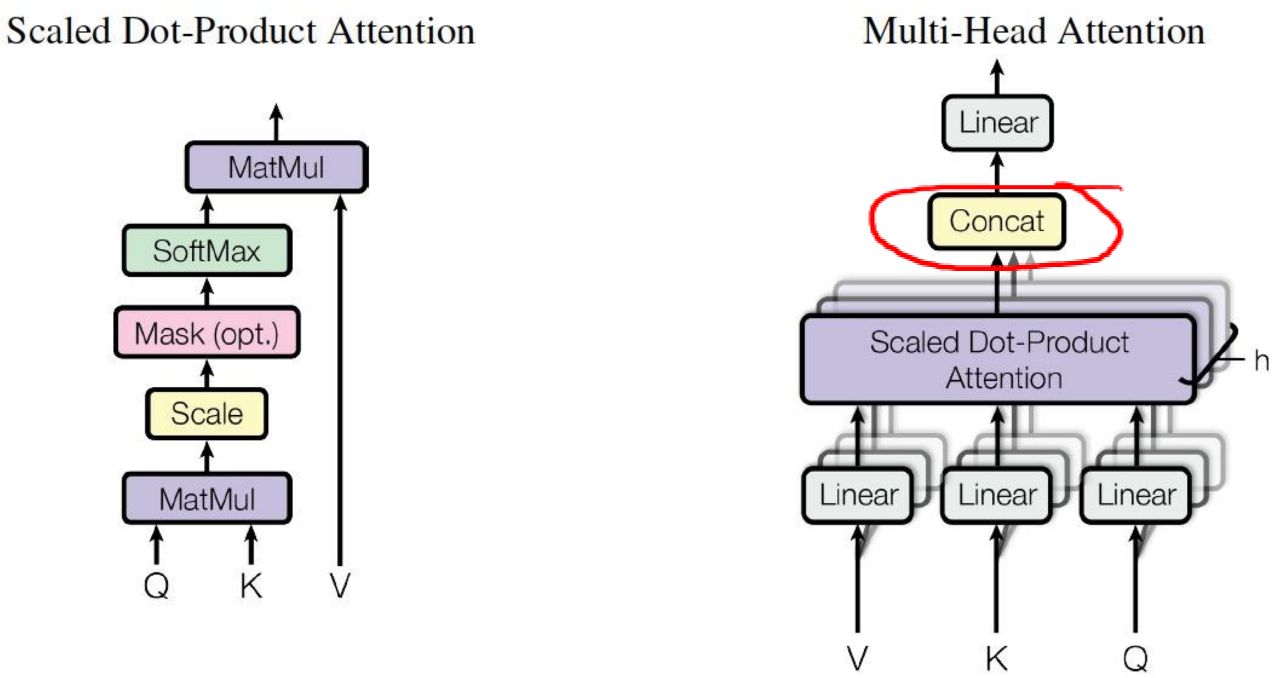
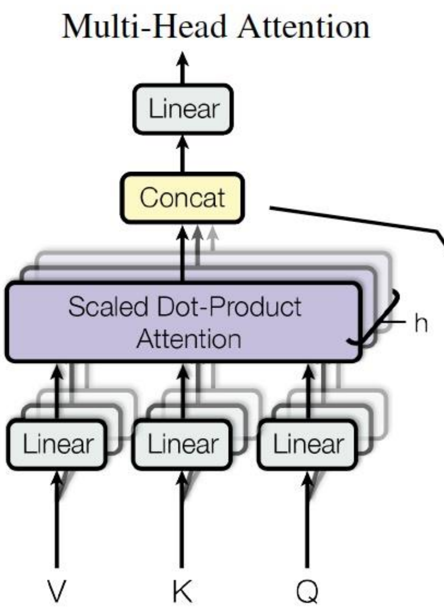



Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

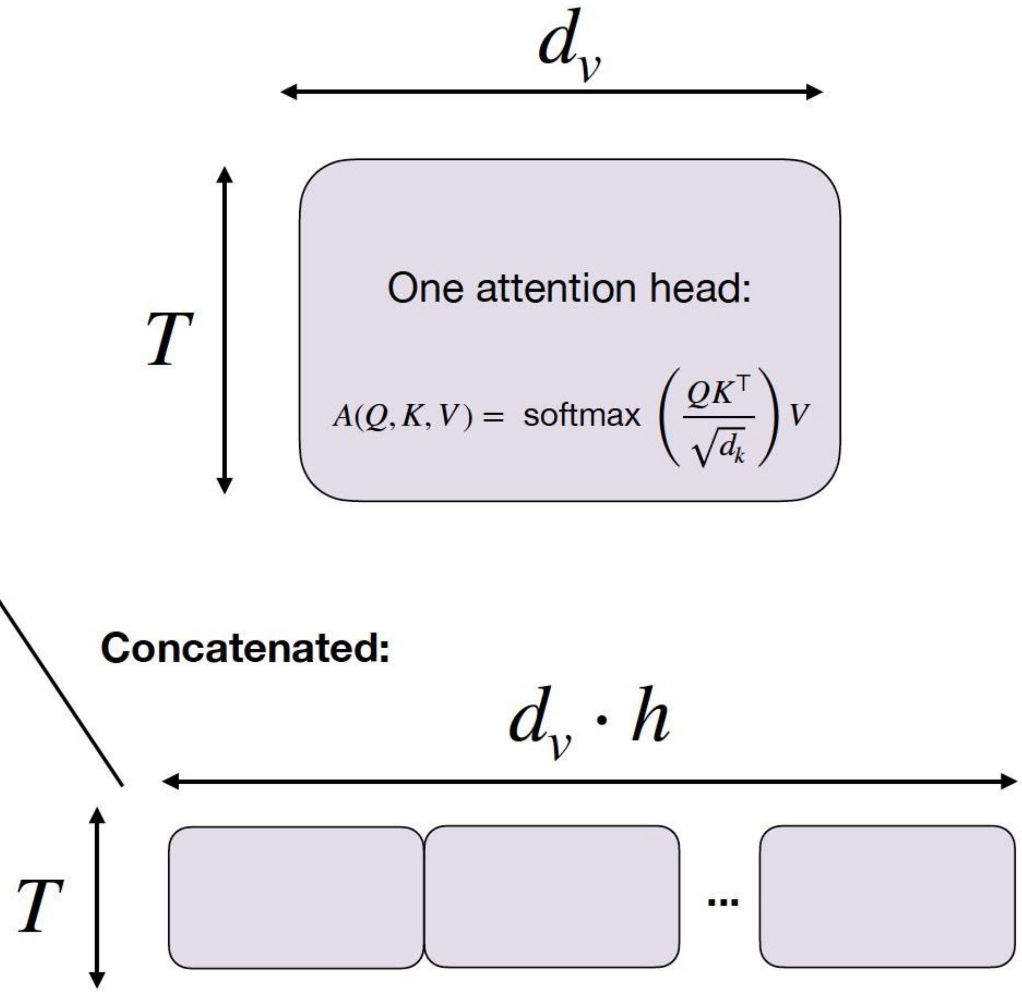
$$\begin{aligned} \mathrm{MultiHead}(Q, K, V) &= \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O \\ \text{where}\ \mathrm{head_i} &= \mathrm{Attention}(QW^Q_i, KW^K_i, VW^V_i) \end{aligned}$$

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

# Multi-Head Attention




Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.




Input sequence dim.
in original transformer:

$T \times d_e = T \times 512$

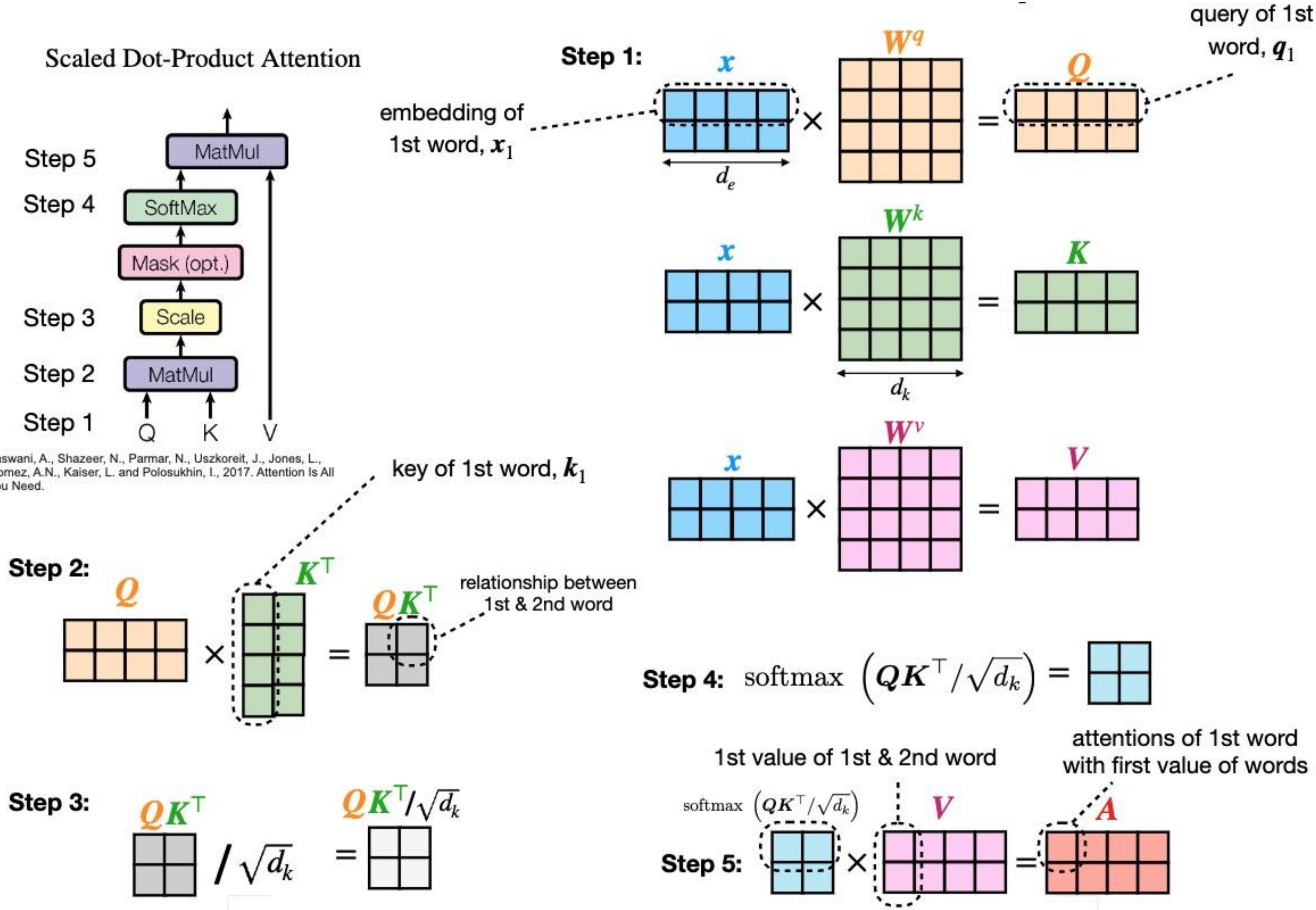
and

$d_v = 512/h = 64$

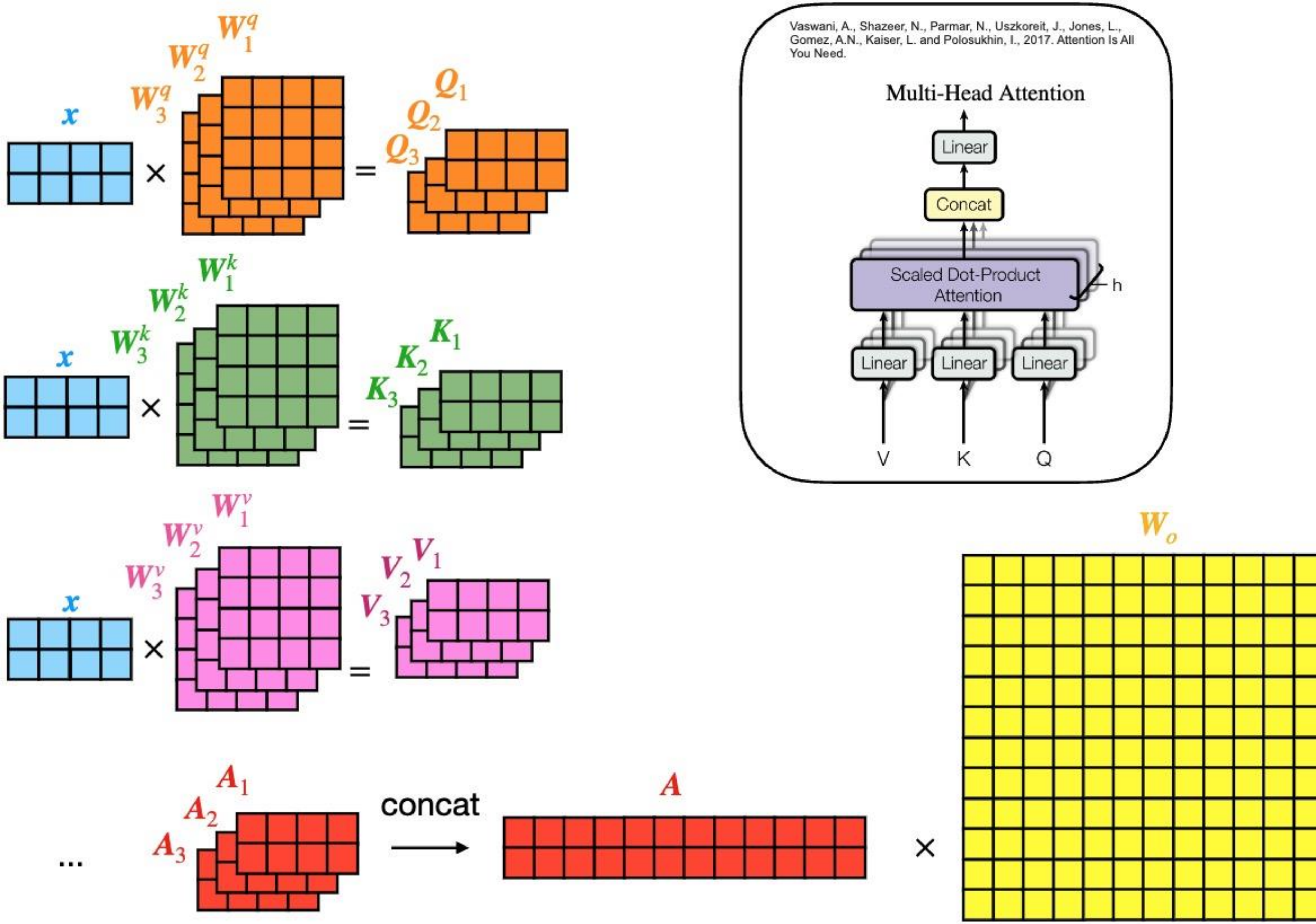


Credit: Sebastian Raschka

## Scaled Dot-Product Attention




Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.



$x$
$W_1^q$
$W_2^q$
$W_3^q$
$Q_1$
$Q_2$
$Q_3$
$W_1^k$
$W_2^k$
$W_3^k$
$K_1$
$K_2$
$K_3$
$W_1^v$
$W_2^v$
$W_3^v$
$V_1$
$V_2$
$V_3$
...
$A_1$
$A_2$
$A_3$
concat
$A$
$W_o$
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.
Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
V
K
Q

# Step-by-Step Walkthrough of Self-Attention in Transformers

## 1. Input Representation

- Consider an **input sequence** of $n$ tokens (e.g., words in a sentence).
- Each token is converted into a **word embedding** of size $d_{\text{model}} = 512$.
- The input sequence is represented as a matrix:

$$X \in \mathbb{R}^{n \times d_{\text{model}}}$$

where:

- $n$ = sequence length (e.g., 10 for a short sentence).
- $d_{\text{model}} = 512$ (embedding dimension).

# Step-by-Step Walkthrough of Self-Attention in Transformers

## 2. Linear Projections: Computing Queries, Keys, and Values

Each token embedding is linearly projected into **query ($Q$), key ($K$), and value ($V$)** vectors using learned weight matrices.

- **Weight Matrices:**

$$W_q \in \mathbb{R}^{d_{\text{model}} \times d_q}, \quad W_k \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_v \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

- **Typical Dimensions:**
  - $d_{\text{model}} = 512$ (input embedding size).
  - $d_k = d_q = d_v = 64$ (smaller subspace for attention calculations).
  - **For multi-head attention**: These projections are done **separately for each attention head**.
- **Computing the Projections:**

$$Q = XW_q, \quad K = XW_k, \quad V = XW_v$$

  - The resulting matrices:

$$Q, K, V \in \mathbb{R}^{n \times d_k}$$

(i.e., for a batch of **10 tokens**, each of **64-dimensional vectors** per head).

# Step-by-Step Walkthrough of Self-Attention in Transformers

### 3. Compute Scaled Dot-Product Attention

The core self-attention mechanism is computed as:

$$A = \frac{QK^T}{\sqrt{d_k}}$$

- **Matrix Dimensions:**
  - $Q \in \mathbb{R}^{n \times d_k}$ and $K^T \in \mathbb{R}^{d_k \times n}$, so:

$$A = QK^T \in \mathbb{R}^{n \times n}$$

  - This forms an **attention score matrix** where each entry $A_{ij}$ represents how much the $i$th token **attends to** the $j$th token.
  - **Why Scale by $\sqrt{d_k}$?** To prevent large dot product values from dominating the softmax.

# Step-by-Step Walkthrough of Self-Attention in Transformers

## 4. Apply Softmax to Compute Attention Weights

We apply the **softmax function** along each row of $A$ to normalize attention scores into probabilities:

$$\text{Attention Scores} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

- **Dimension after Softmax:**
  - $\text{softmax}(A) \in \mathbb{R}^{n \times n}$
  - Ensures each row sums to 1 (probability distribution).

# Step-by-Step Walkthrough of Self-Attention in Transformers

## 5. Compute Weighted Sum of Values

The final step in attention computation:

$$\text{Output} = \text{softmax}(A)V$$

- **Matrix Multiplication Dimensions:**
  - $\text{softmax}(A) \in \mathbb{R}^{n \times n}$
  - $V \in \mathbb{R}^{n \times d_v}$
  - Resulting **attention output**:

$$\text{Output} \in \mathbb{R}^{n \times d_v}$$

  - This produces an updated representation of each token, **contextualized** based on attention.

# Step-by-Step Walkthrough of Self-Attention in Transformers

## 6. Multi-Head Attention

The above steps happen **independently for each attention head.**

- Suppose there are **8 heads**.
- Each head has its **own** $W_q, W_k, W_v$ matrices, projecting into $d_k = d_v = 64$ **per head**.
- **Each head processes the input in parallel**, producing multiple attention outputs:

$$\text{Head}_i \in \mathbb{R}^{n \times d_k}$$

- The outputs of all **8 heads** are concatenated:

$$\text{MultiHead}(X) = \text{Concat}(\text{Head}_1, \ldots, \text{Head}_8)$$

  - **Final Output Dimension:**

$$\mathbb{R}^{n \times (8 \times d_k)} = \mathbb{R}^{n \times 512}$$

  - This ensures that the multi-head attention **preserves the original embedding size**.
- Finally, this concatenated output is **projected back** to $d_{\text{model}}$ using:

$$W_o \in \mathbb{R}^{(h \cdot d_k) \times d_{\text{model}}}$$

# Step-by-Step Walkthrough of Self-Attention in Transformers

## 7. Feedforward Network (FFN)

- Each token's updated representation from attention is passed through a **fully connected feedforward network (FFN)**:

$$\text{FFN}(X) = \max(0, XW_1 + b_1)W_2 + b_2$$

- **Dimensions:**
  - $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ffn}}}$ with $d_{\text{ffn}} = 2048$.
  - $W_2 \in \mathbb{R}^{d_{\text{ffn}} \times d_{\text{model}}}$.

## 8. Add & Norm: Residual Connections

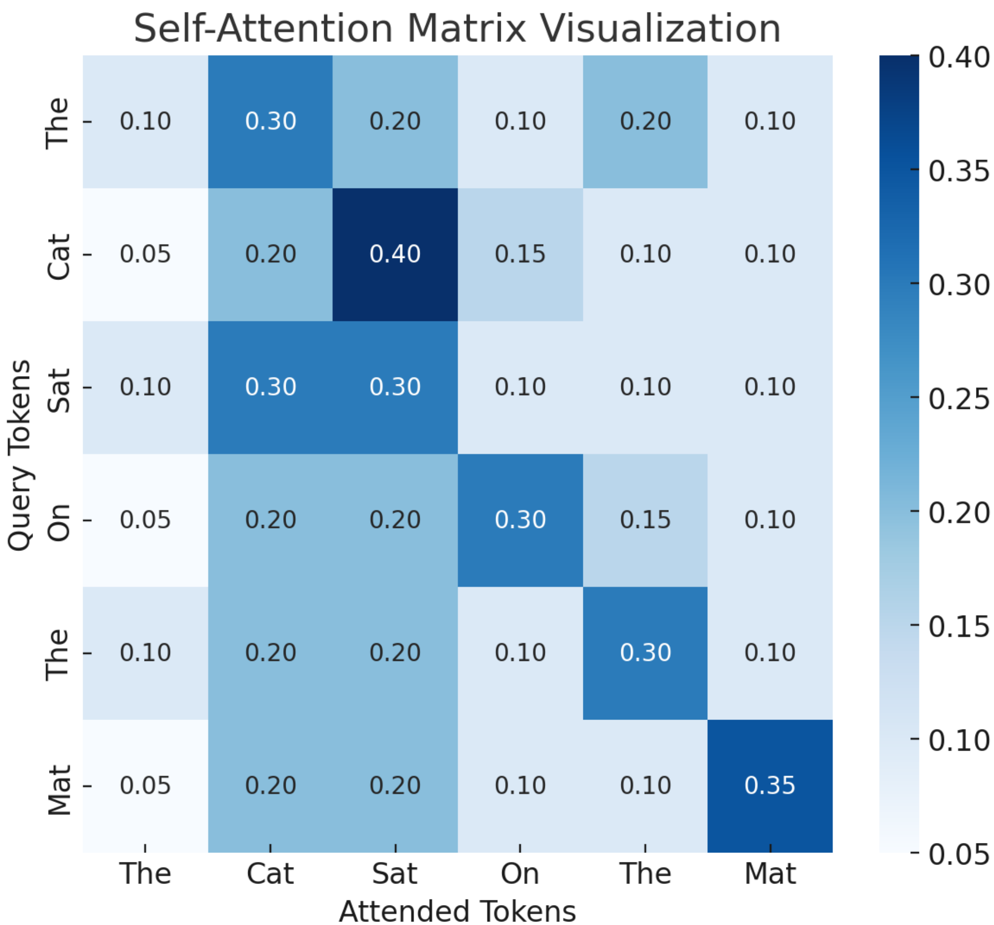
- Both **multi-head attention** and **FFN** have **residual connections**:

$$X = \text{LayerNorm}(X + \text{MultiHead}(X))$$

$$X = \text{LayerNorm}(X + \text{FFN}(X))$$

- These ensure **stability** and **better gradient flow**.

# Self-attention matrix visualization (an example)



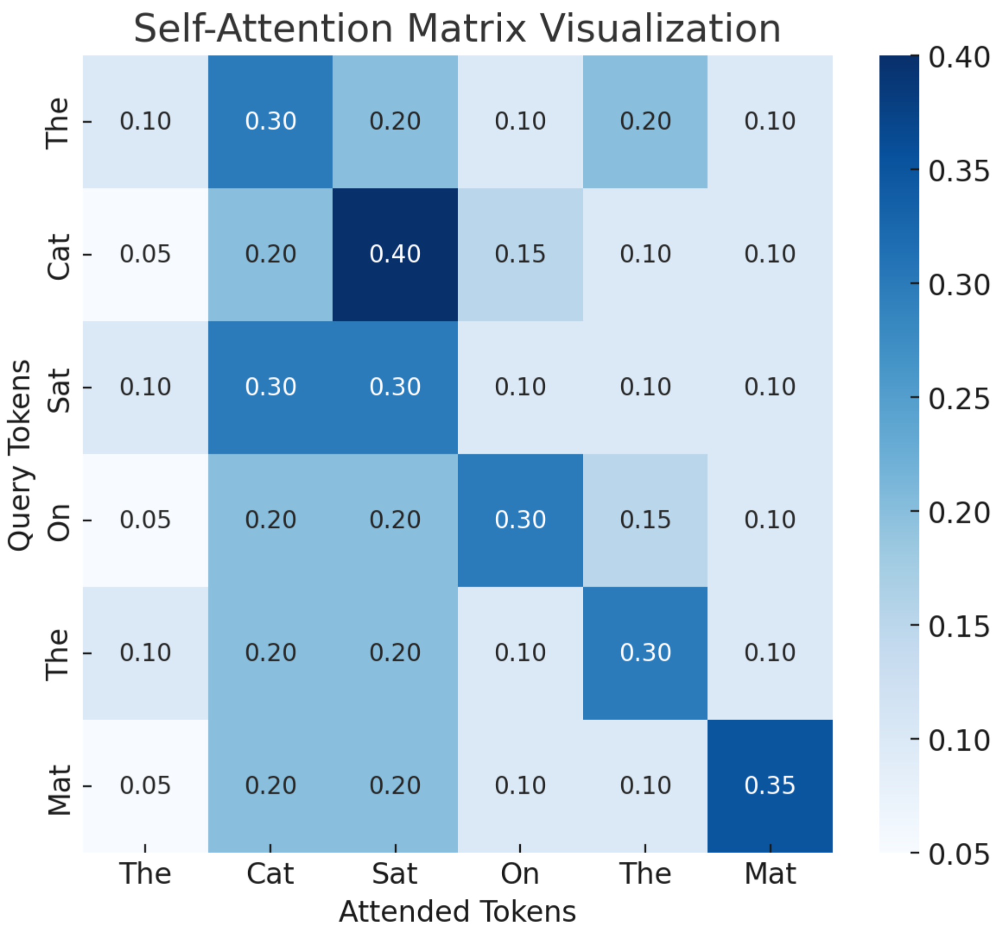

Here is a visual representation of a **self-attention matrix** for the sentence **"The cat sat on the mat."**

- **Rows**: Represent the **query tokens** (i.e., the token currently attending to others).
- **Columns**: Represent the **tokens being attended to**.
- **Values (0.0 to 1.0)**: Represent the attention weight—**higher values (darker shades)** indicate **stronger attention**.



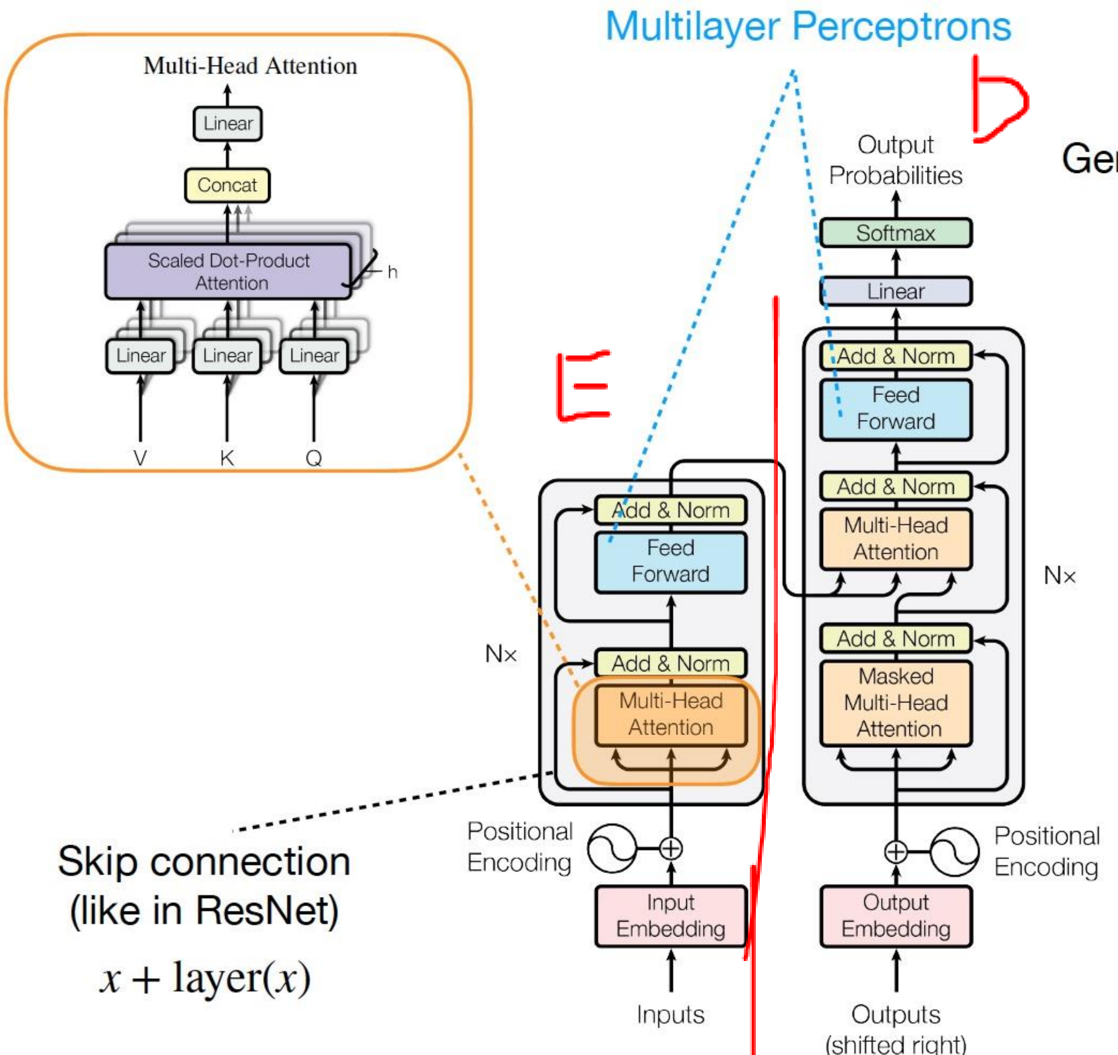
UCF Center for Research in Computer Vision

# Self-attention matrix visualization (an example)




- The **"Cat" row** (2nd row) shows high attention to **"Sat" (0.4)**, meaning "Cat" pays the most attention to "Sat."
- The **"Mat" row** (last row) has a high value (0.35) for **itself**, meaning it primarily focuses on itself.

# The Transformer Architecture




Generates output words one at a time

Skip connection (like in ResNet)

$$x + \text{layer}(x)$$

Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Credit: Sebastian Raschka

# Add Positional Encoding to Word Embedding

- Scaled dot-product and fully-connected layer are permutation invariant
- Sinusoidal positional encoding is a vector of small values (constants) added to the embeddings
- As a result, same word will have slightly different embeddings depending on where they occur in the sentence

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d}\text{model}}\right)$$

$$PE_{pos,2i+1} = \cos\left(\frac{\text{pos}}{10000^{(2i+1)/d}\text{model}}\right)$$

Position encoding can also be learnable

$d_o = 512$

$d_e = 512$

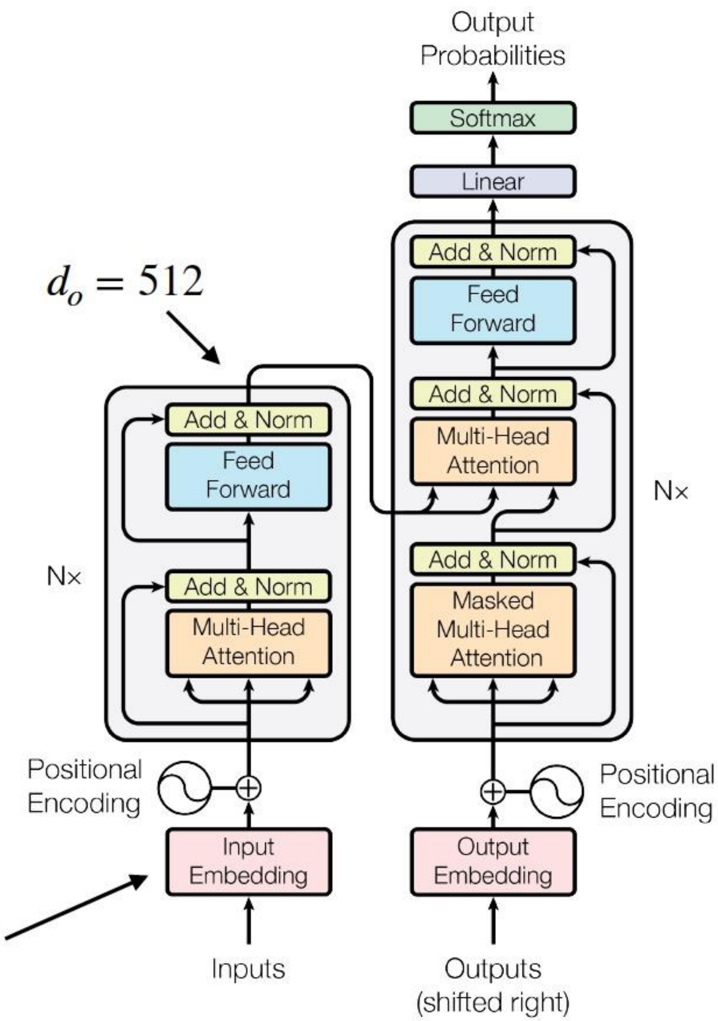



Figure 1: The Transformer - model architecture.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

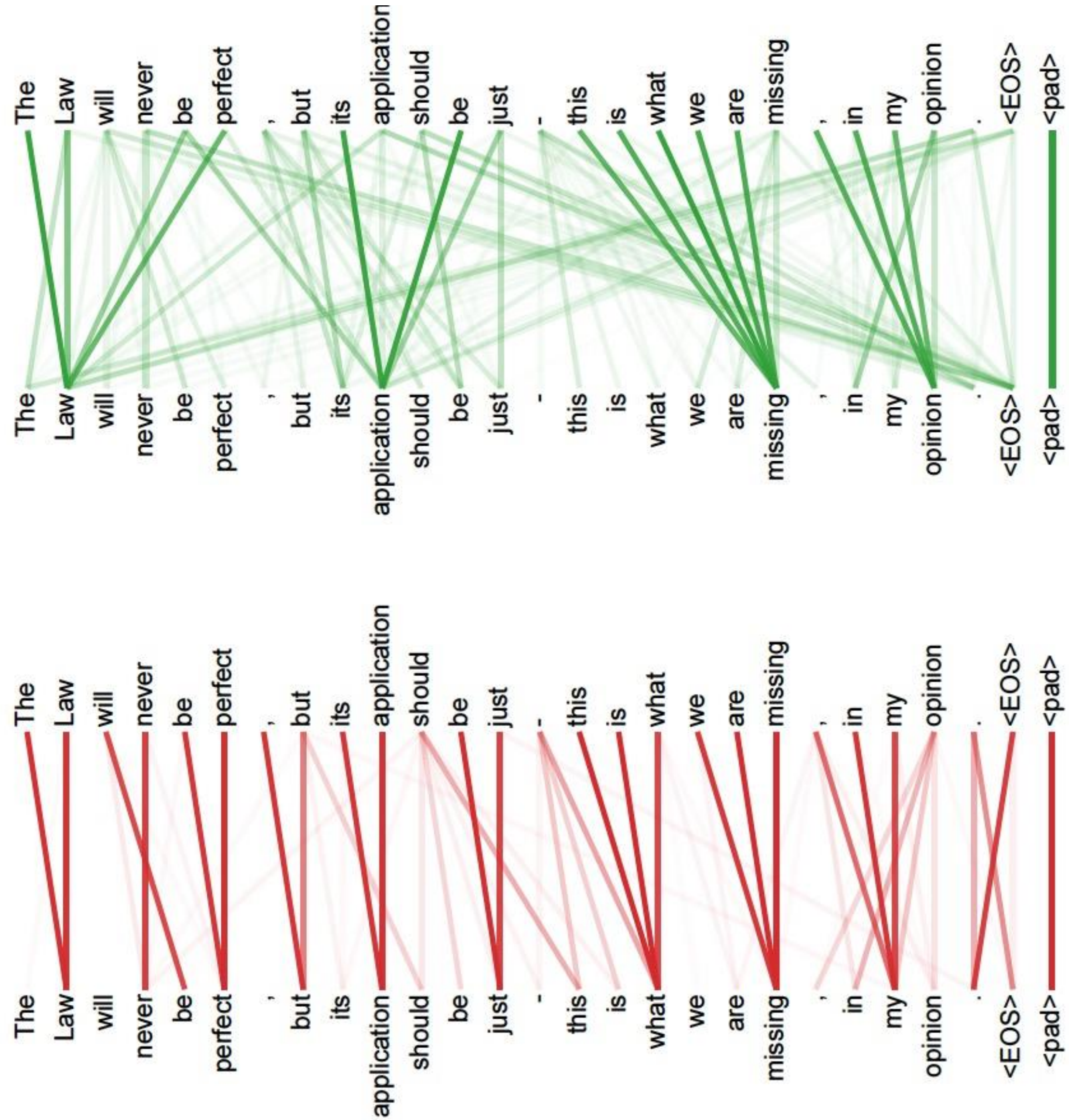



Figure 5: Many of the attention heads exhibit behaviour that seems related to the structure of the sentence. We give two such examples above, from two different heads from the encoder self-attention at layer 5 of 6. The heads clearly learned to perform different tasks.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

- Attention in transformers, step-by-step

https://www.youtube.com/watch?v=eMlx5fFNoYc&t=473s

# Limitations of Self-Attention Mechanism

**Computational Complexity**

- The standard attention mechanism computes a similarity matrix between all pairs of tokens, resulting in:

$$\mathcal{O}(n^2 \cdot d)$$

  - n : Length of the input sequence.
  - d : Embedding dimension.
- This quadratic growth in computation and memory makes it impractical for very long sequences (e.g., documents, long videos, or large medical images).

**Memory Bottleneck**

- The attention matrix scales with $n^2$ , which consumes significant GPU/TPU memory, especially for large input sequences or multi-head setups.
- This limits batch size and model size during training.

# Efficient Self-Attention Mechanisms

- Notable Approaches

**(a) FlashAttention**

- **Key Idea**: Memory-efficient attention computation by processing in a chunked, block-wise manner.
- **Benefits**:
  - Reduces memory complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot d)$.
  - Fused GPU kernels for speed.
  - Scalable to long sequences (e.g., tens of thousands of tokens).

Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, *35*, 16344-16359.
Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

# Efficient Self-Attention Mechanisms

- Notable Approaches

**(b) Sparse Attention**

- **Key Idea**: Leverages the observation that many attention weights are close to zero.
- Only computes attention for a subset of token pairs (sparse patterns like local, random, or strided connections).
- **Examples**:
  - **Longformer**: Uses a mix of local and global attention patterns.
  - **BigBird**: Introduces random, sliding window, and global attention patterns for scalability.
- **Complexity**: Reduces attention computation to $\mathcal{O}(n \cdot \log n)$ or $\mathcal{O}(n \cdot k)$, where $k$ is the number of non-zero entries.

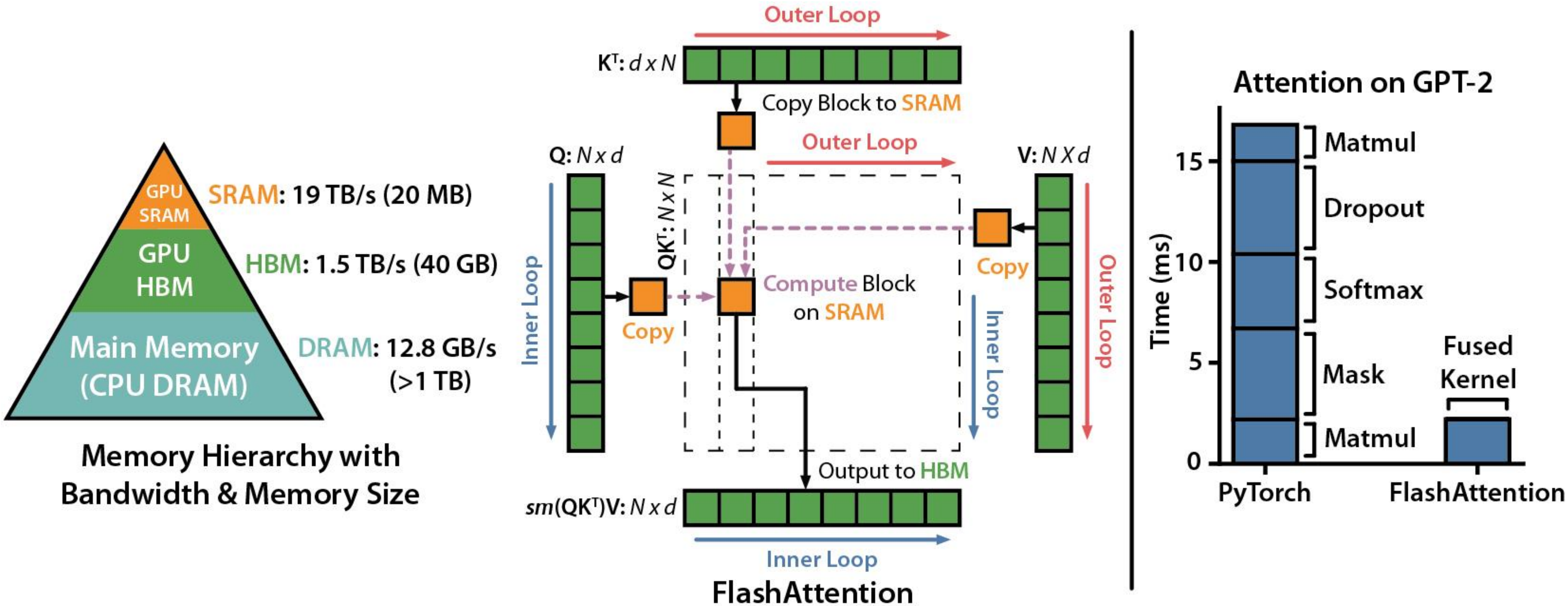
Beltagy, Iz, Matthew E. Peters, and Arman Cohan. "Longformer: The long-document transformer." *arXiv preprint arXiv:2004.05150* (2020).
Zaheer, Manzil, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham et al. "Big bird: Transformers for longer sequences." Advances in neural information processing systems 33 (2020): 17283-17297.

# Efficient Self-Attention Mechanisms

| Method | Complexity | Key Feature | Use Case |
| --- | --- | --- | --- |
| FlashAttention | $\mathcal{O}(n \cdot d)$ | Chunked processing, fused kernels | General-purpose |
| Longformer | $\mathcal{O}(n \cdot k)$ | Local and global attention patterns | Text/document analysis |
| Linformer | $\mathcal{O}(n \cdot d)$ | Low-rank projection of attention | Long text, moderate accuracy |
| Performer | $\mathcal{O}(n \cdot d)$ | Kernelized approximation of softmax | Large-scale tasks |
| Reformer | $\mathcal{O}(n \cdot \log n)$ | LSH-based sparse token grouping | Sparse token interactions |

Choromanski, Krzysztof, et al. "Rethinking attention with performers." *arXiv preprint arXiv:2009.14794* (2020).
Kitaev, Nikita, Łukasz Kaiser, and Anselm Levskaya. "Reformer: The efficient transformer." *arXiv preprint arXiv:2001.04451* (2020).

# FlashAttention




https://github.com/Dao-AILab/flash-attention

**FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness**
Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, Christopher Ré

# Efficient Self-Attention Mechanisms

- Other Approaches
  - Hardware-software co-design (FPGA, etc.)

# Vision Transformer (ViT)

# 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

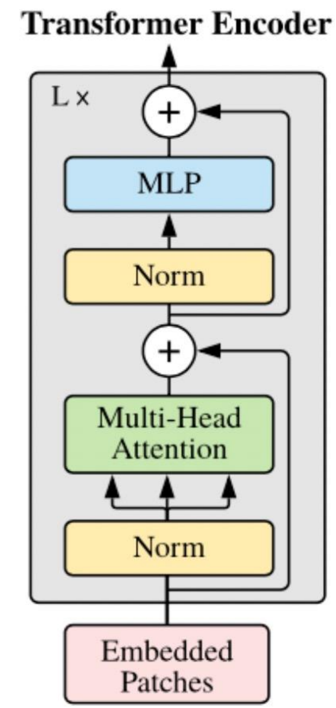
Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

## ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

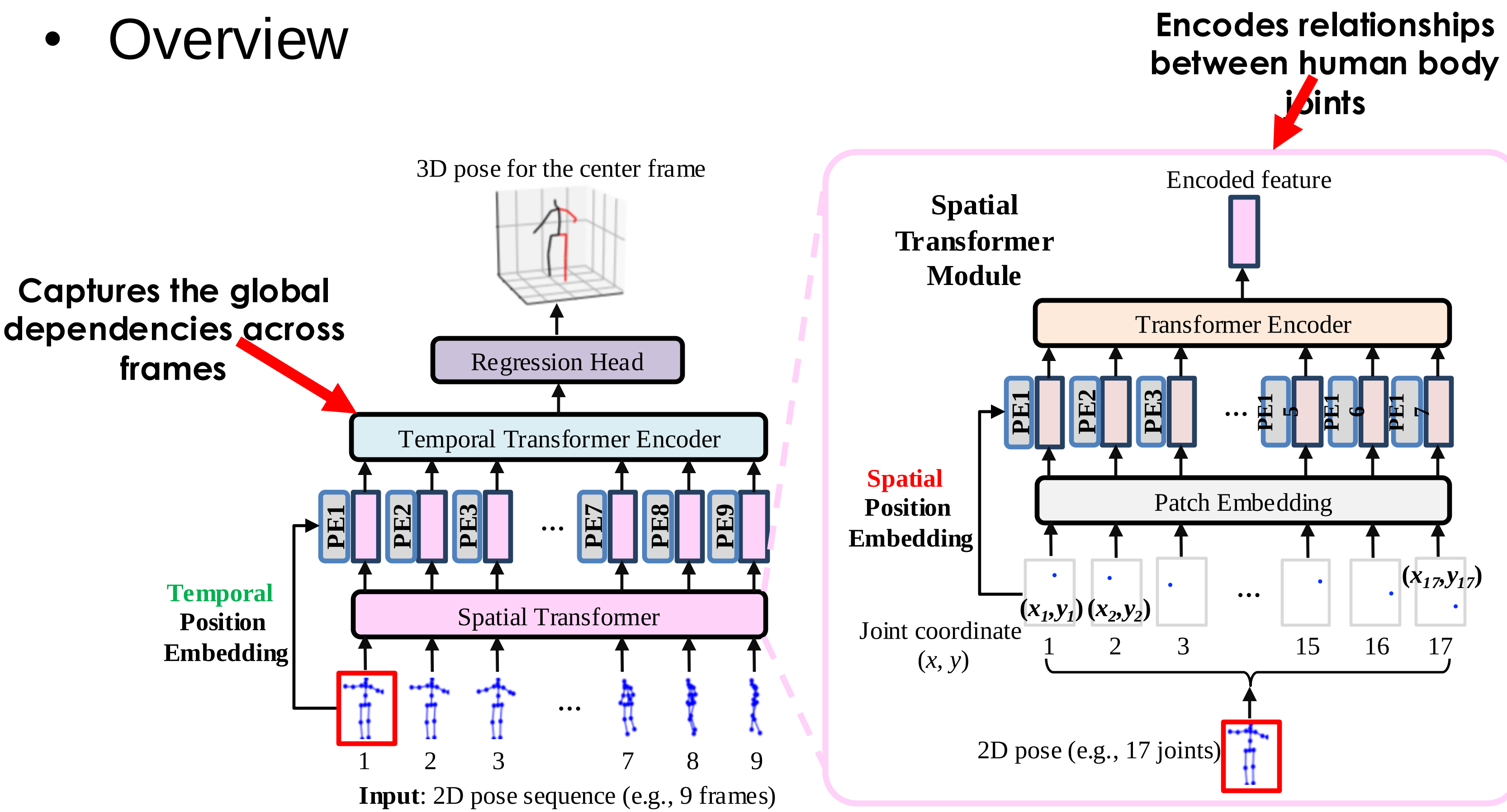
# Vision Transformer




https://github.com/lucidrains/vit-pytorch

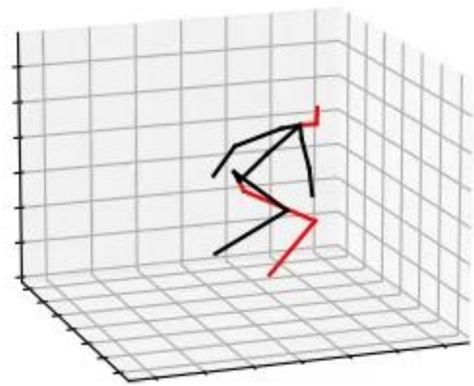
# Other applications (ViT for Human Pose Estimation)

- Overview




Zheng, Ce, et al. "3d human pose estimation with spatial and temporal transformers." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021.

Video with heavy occlusion



# What is Cross-Attention

**1. Self-Attention vs. Cross-Attention**

- **Self-Attention**: The **query (Q), key (K), and value (V)** all come from the **same input** (e.g., a sentence in NLP or patches in ViTs).
- **Cross-Attention**: The **query (Q)** comes from one modality (e.g., **text**) while the **keys (K) and values (V)** come from another modality (e.g., **image features**).

**2. Cross-Attention in Multi-Modal Models**

- Used in models that integrate multiple input types, such as:
  - **Vision-Language Models (e.g., CLIP, Flamingo)** → Image ↔ Text
  - **Video Question Answering** → Video Frames ↔ Text Question
  - **Speech-Text Models (e.g., Whisper)** → Audio ↔ Text

# What is Cross-Attention

**3. Cross-Attention Computation**

$$A = \text{softmax}\left(\frac{Q_T K_I^T}{\sqrt{d_k}}\right)$$

$$\text{CrossAttention Output} = AV_I$$

- $Q_T$ (Queries) → From **Text Tokens**
- $K_I, V_I$ (Keys, Values) → From **Image Features (e.g., CNN/Vision Transformer)**
- Softmax ensures the text tokens **attend to relevant image features**.

# Example - Image Captioning using Cross-Attention

## Use Case: Generating Captions for an Image

- **Input:**
  - **Image** → Processed via **CNN/ViT** → Produces image embeddings.
  - **Text (e.g., partial caption)** → Used to generate **queries (Q)**.

- **Cross-Attention:**
  - The text queries attend to **image features** to determine the most relevant visual information.
  - Used in **image captioning models like BLIP and Flamingo**.

# Resources

- Surveys
  - Transformers in Vision: A Survey
  https://arxiv.org/abs/2101.01169
  - A Survey on Vision Transformer
  https://arxiv.org/abs/2012.12556
  - Transformers in Medical Imaging: A Survey
  https://arxiv.org/pdf/2201.09873.pdf
  - Efficient Transformers: A Survey
  https://arxiv.org/pdf/2009.06732.pdf

the memory and computational complexity required to compute the attention matrix is quadratic in the input sequence length

- Codes
  - https://huggingface.co/docs/transformers/index
  - https://github.com/dk-liang/Awesome-Visual-Transformer
  - https://github.com/fahadshamshad/awesome-transformers-in-medical-imaging

Thank you!

Question?