GPT (Generative Pre-trained Transformer)

2023. 8

Yongjin Jeong, KwangWoon University

Seq2seq based on RNN

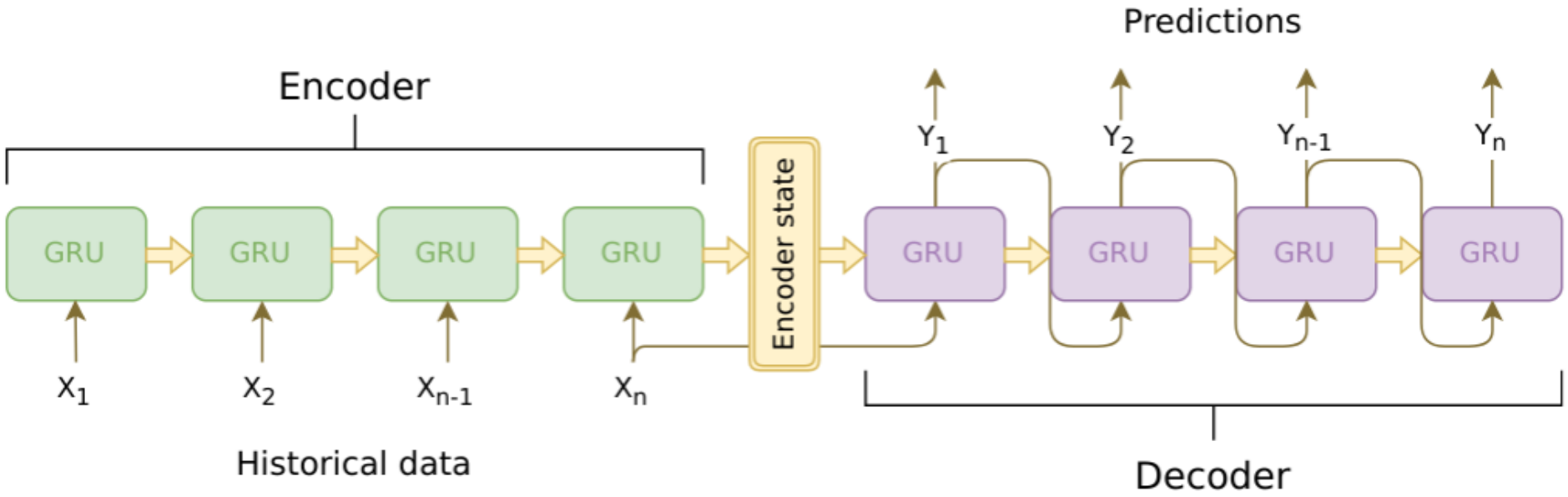


Image source: https://jeddy92.github.io/JEddy92.github.io/ts_seq2seq_intro/

- One fixed context vector (the last hidden state) is transferred to Decoder.

Attention with RNN

simply, weighted average of some values

- **General RNN Encoder-decoder**
  - Encoder

$$h_t = f\left(x_t, h_{t-1}\right)$$

$$c = q\left(\{h_1, \cdots, h_{T_x}\}\right)$$

x=($x_1, \ldots, x_{Tx}$): input sequence
$h_t \in R^n$: a hidden state at time *t*,
c: context vector generated from sequence of hidden states

  - Decoder:
    - trained to predict the next word $y_{t'}$ given the context vector (*c*) and all the previously predicted words $\{y_1, \ldots, y_{t'-1}\}$

$$p(\mathbf{y}) = \prod_{t=1}^{T} p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c)$$

y=($y_1, \ldots, y_{Ty}$): output sequence

with RNN, modeled as

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

S_t : hidden state of RNN

Attention with RNN – Align and Translate

- **Decoder with Attention**

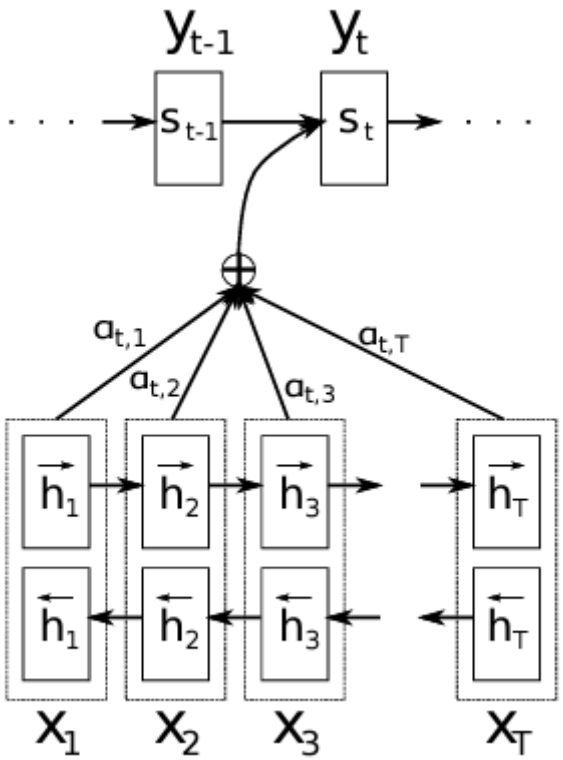
In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1}, s_i, c_i), \qquad (4)$$

where s_i is an RNN hidden state for time i, computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

Unlike the encoder–decoder approach(see Eq.(2)), here the probability is conditioned on a distinct context vector c_i for each target word y_i.



$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$ (weighted sum of annotations h_i)

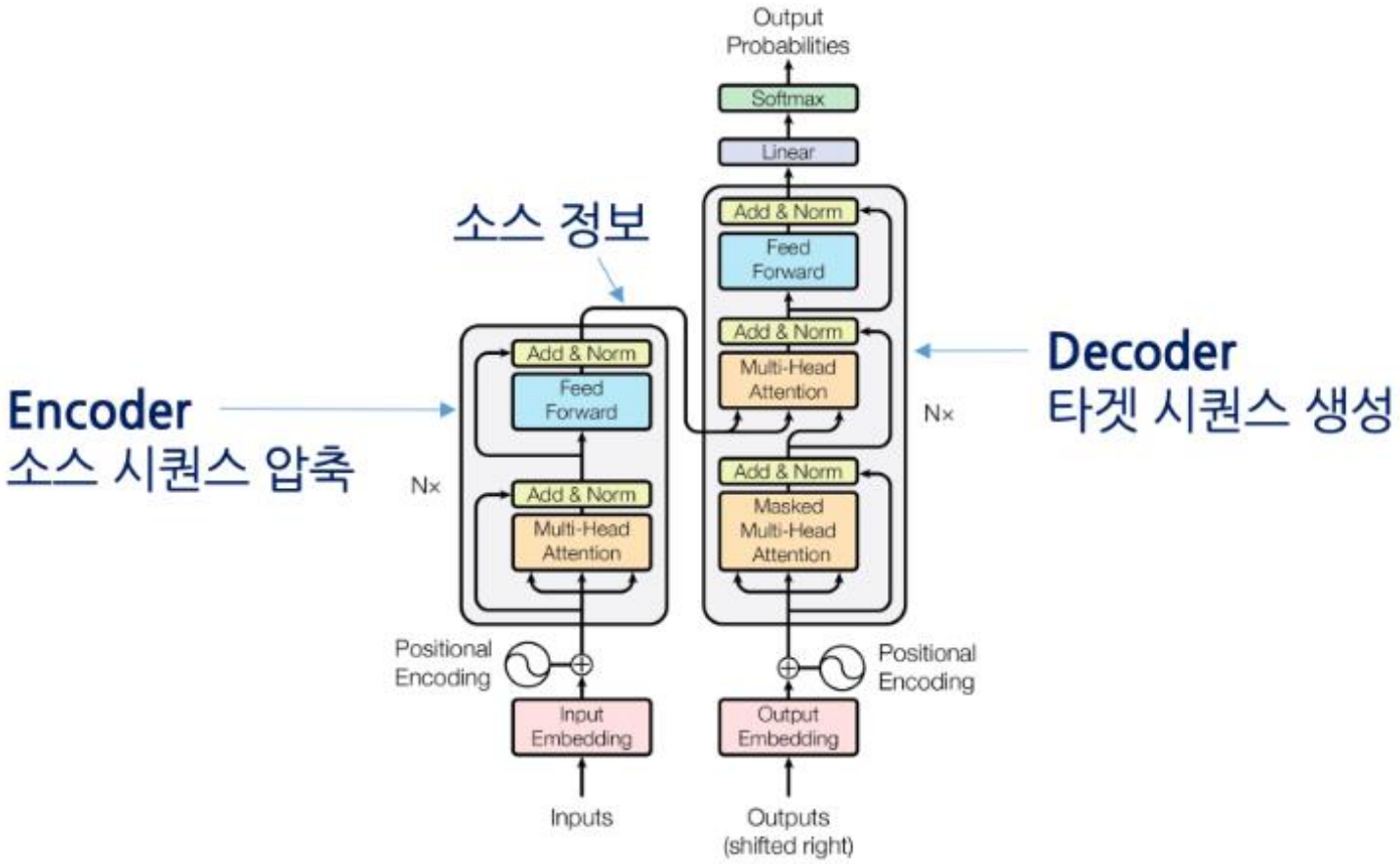
$$h_j = \left[\overrightarrow{h}_j^\top ; \overleftarrow{h}_j^\top\right]^\top$$

weights ⇒ $$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$,where $$e_{ij} = a(s_{i-1}, h_j)$$

alignment model ; it scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden states i−1(just before emitting yi) and the j-th annotation hj of the input sentence. (can be parameterized as a Forward Neural Network)

The Transformer

- **Without RNN**



Encoder:

- Multi-head self-attention mechanism
- A simple position-wise fully connected feed-forward network

Decoder:

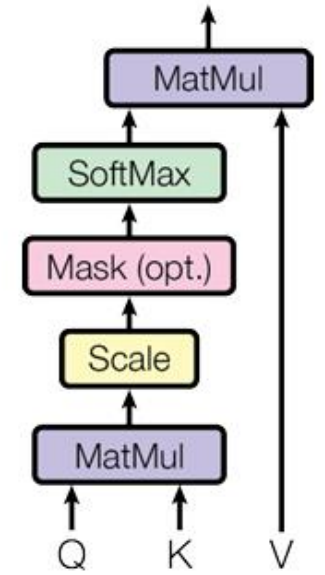
- **Masked** Multi-head self-attention mechanism (ensures that the predictions for position i can depend only on the known outputs at positions less than i)
- A simple position-wise fully connected feed-forward network
- Multi-head attention over the output of the encoder stack

https://ratsgo.github.io/nlpbook/docs/language_model/tr_self_attention/

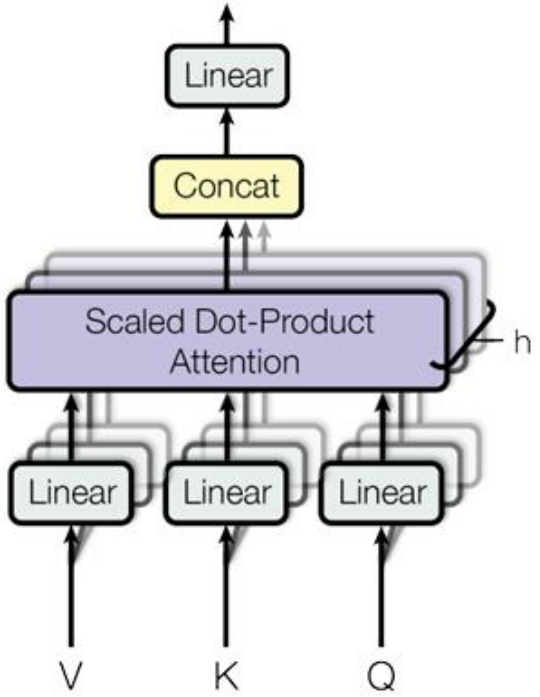
Multi-Head Attention

- **Attention**
  - Mapping a query and a set of key-value pairs to an output
  - The output is computed as a weighted sum of the values, where the weight is computed by a compatibility function of the query with the corresponding key
  - Query, key, and value are packed into matrices Q, K, and V respectively.
  - Usually, $d_k = d_q$, and $d_v = d_{model} = d_{emb}$

Scaled Dot-Product Attention



Multi-Head Attention



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

*attention(q,**k**,**v**) = $\sum_i$ similarity(q,k_i) * v_i*

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

$$W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$

The Transformer - Attention

- **Multi-head attention**
  - In "**encoder-decoder attention**“ layers, the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. (This allows every position in the decoder to attend overall positions in the input sequences. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models.)
  - **Self-attention layers in Encoder**: all of the keys, values, and queries come from the same place (the output of the previous layer in the encoder). Each position in the encoder can attend to all positions in the previous layer of the encoder.
  - **Self-attention layers in Decoder**: we allow each position in the decoder to attend to all positions in the decoder up to and including that position. We need to prevent leftward information flow in the decoder to preserve the auto-regressive property. We implement this inside of scaled dot-product attention by masking out(setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.

Attention by Query, Key, and Value

- Query, Key, Value
  - Simply generated with three different matrices (linear **transformation**) <- why?
  - **Query**: the input or prompt provided by the user.
  - **Key**: a piece of information used to retrieve relevant knowledge. (In the case of ChatGPT, the key is typically generated from the query.)
  - **Value**: the information retrieved by using the key
- What is the **Weight** for Value (Query * Key):
  - to calculate the (cosine) similarity or dot product between two vectors (a scalar value that represents the similarity or relevance between the key and the query)
  - Multiplying the key and query is a common mathematical operation used to calculate the similarity or **dot product** between two vectors.

❖ Intuition: for the sentence "Jane visits Africa."
- When your eyes see ***jane***, your brain looks for **the most related word** in the rest of the sentence to understand what **jane** is about (query). Your brain focuses or attends to the word **visit** (key).

https://stats.stackexchange.com/questions/421935/what-exactly-are-keys-queries-and-values-in-attention-mechanisms

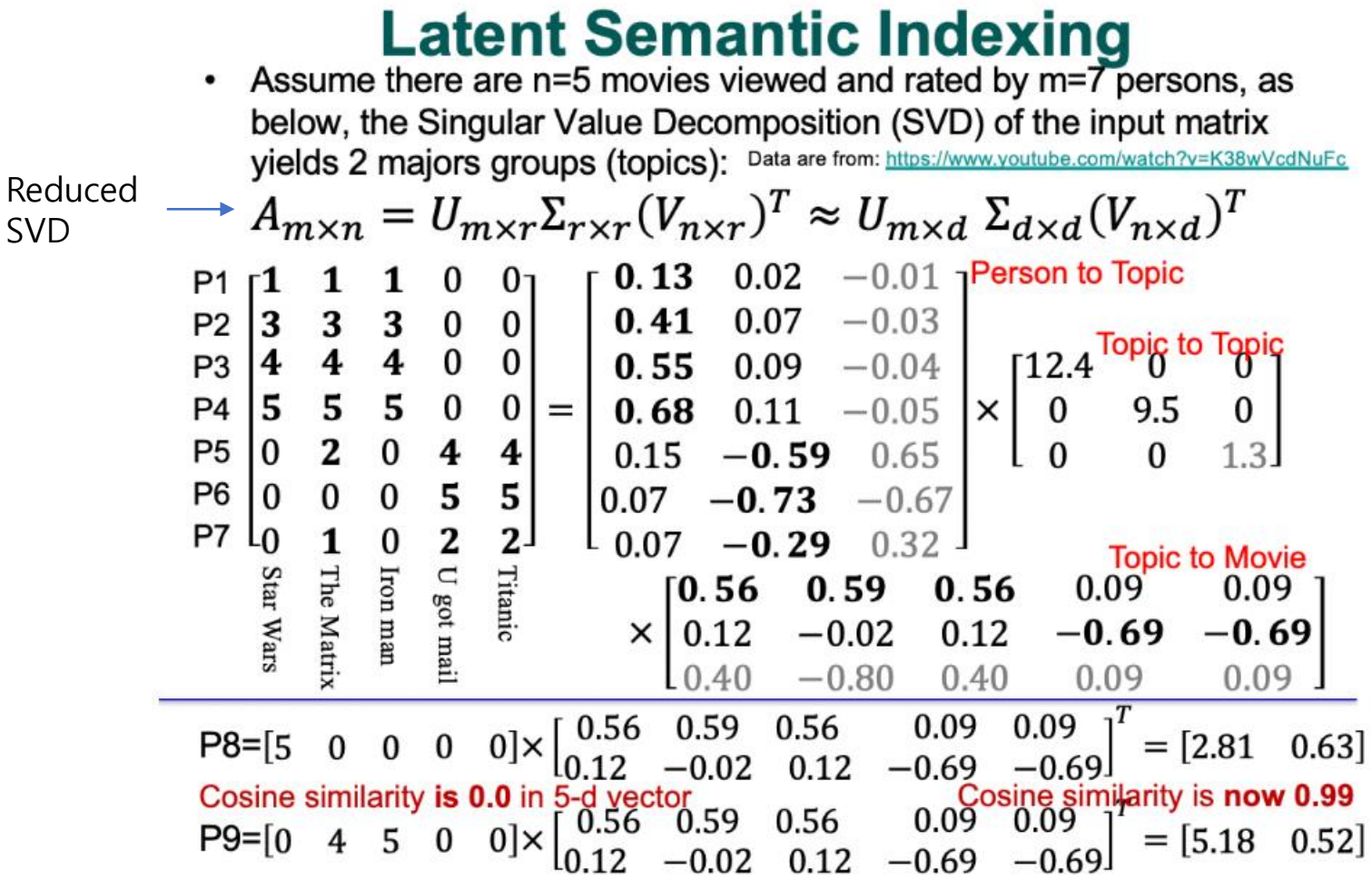
Attention by Query, Key, and Value

- Why **transformation** for Query, Key, Value?
  - we could still use the original encoder state vectors as the queries, keys, and values. So, **why we need the transformation**?
  - Simply a matrix multiplication: (I is the input (encoder) state vector)

    Query = I x W^Q , Key = I x W^K , Value = I x W^V
  - (1) If we do not transform input vectors, the dot product for computing the weight for each input's value will always yield a maximum weight score for the individual input token itself. (input *i* -번째 위치에서 *n* 개의 attention weight 를 계산하면 그 자신에서 가장 높은 값이 나옴)
  - For example, for the **pronoun** token, we need it to attend to its referent, not the pronoun token itself.
  - (2) The transformation **may yield better representations for Query, Key, and Value**. (see the next slide)
  - (3) conversion of the input vector into a space with a desired dimension (practically useful)

Attention by Query, Key, and Value

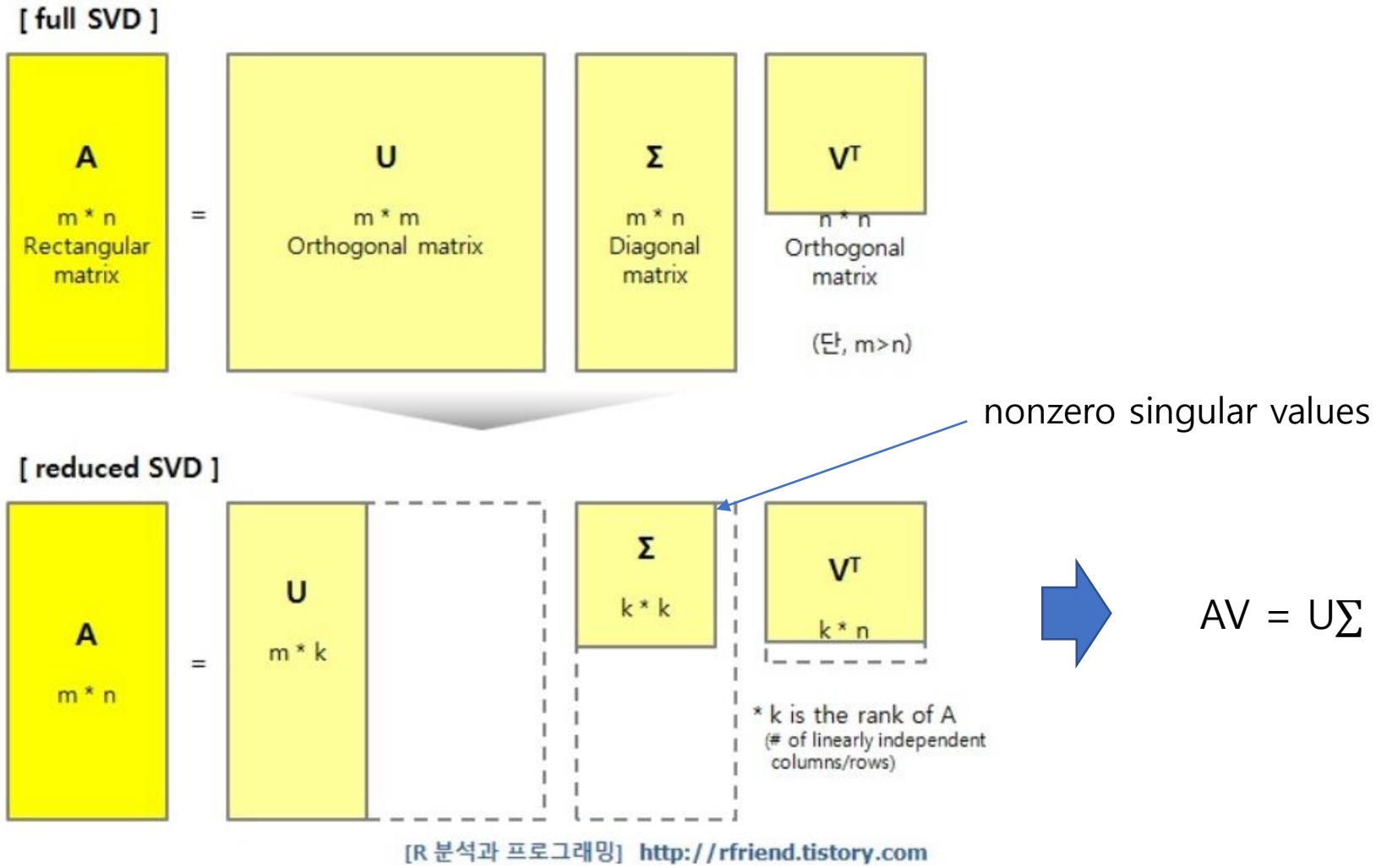
- transformation may yield better representation for computing similarity between two vectors. (https://youtu.be/K38wVcdNuFc?t=10)



- By multiplying an input vector with a matrix V (from the SVD), we get a better representation for computing the compatibility between two vectors, if these two vectors are similar in the topic space as shown in the example in the figure.
- And, these matrices for transformation can be learned in a neural network!
- (before transformation) similarity(P8,P9)=0
- (after transformation) similarity(P8’,P9’) = 0.99

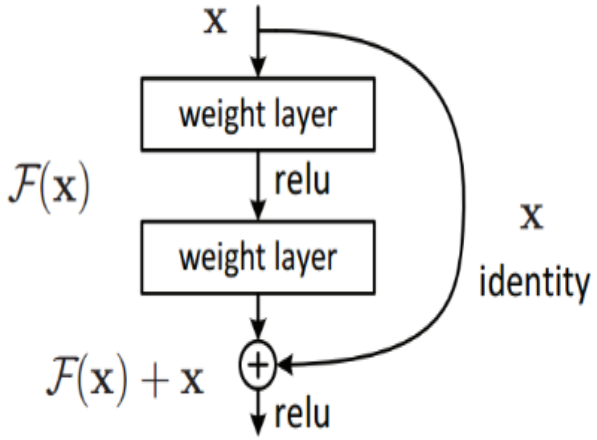
Reduced SVD

- Reduced SVD



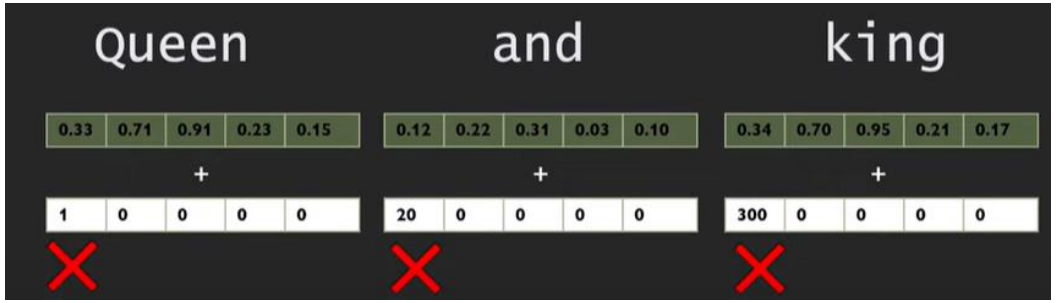
Residual Neural Networks (ResNets)

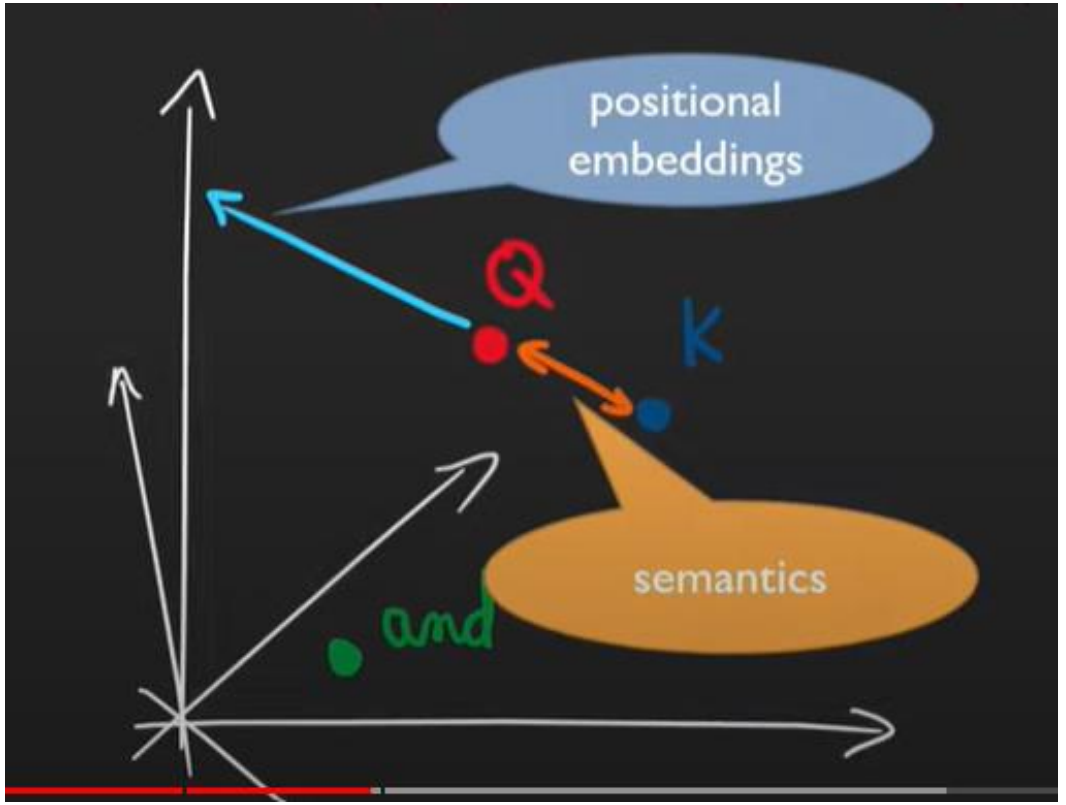
- Why ResNets?
  - Addressing Vanishing Gradients: Skip connections allow gradients to flow directly to earlier layers, and the gradients have a shorter path to propagate, making it easier to update the weights of earlier layers and improve the model's overall performance.
  - Facilitating Training of Deeper Networks: By allowing information to flow through shortcut connections, ResNets enable the training of much deeper neural networks.
  - Promoting Feature Reuse: The skip connections in ResNets facilitate the direct flow of information from earlier layers to later layers. This allows the later layers to access the activations and features from earlier layers, promoting feature reuse.
  - Easier Optimization: provide a form of identity mapping that makes it easier for the network to optimize its weights. This can speed up the convergence during training and enable more efficient optimization.



Positional Encoding

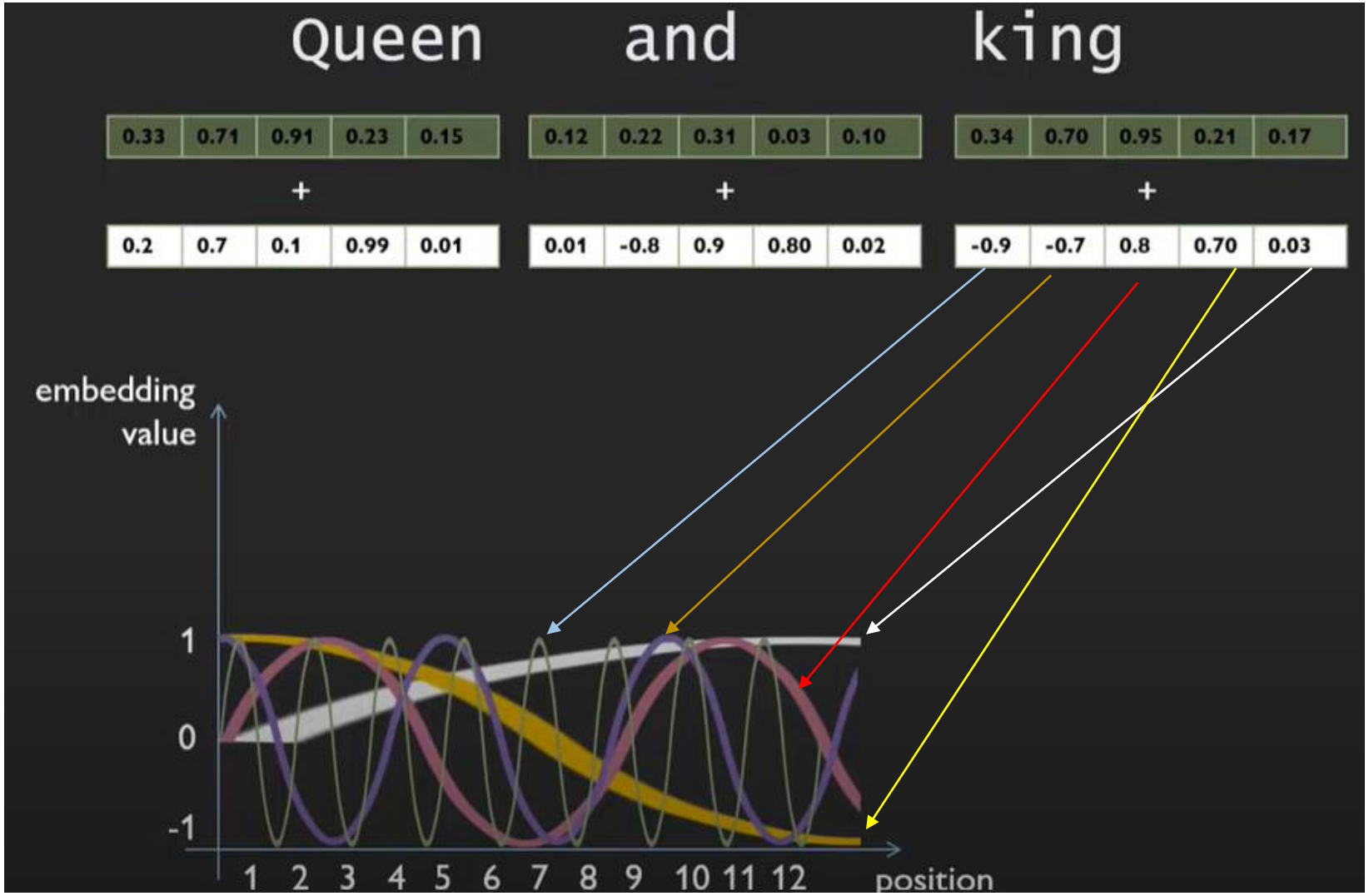
- Because the positional embeddings push the original vector by a bit, they should not be too large. Otherwise, they push the vectors into very distinct subspaces where the positional similarity or dissimilarity overtones the semantic similarity.





(51) Positional embeddings in transformers EXPLAINED | Demystifying positional encodings. - YouTube

Positional Encoding



벡터의 각 element 마다 (다른 컬러로 표기) 주기를 달리 함으로써 단어 위치로 인한 positional vector 값들의 차이가 나타나게 함

(51) Positional embeddings in transformers EXPLAINED | Demystifying positional encodings. - YouTube

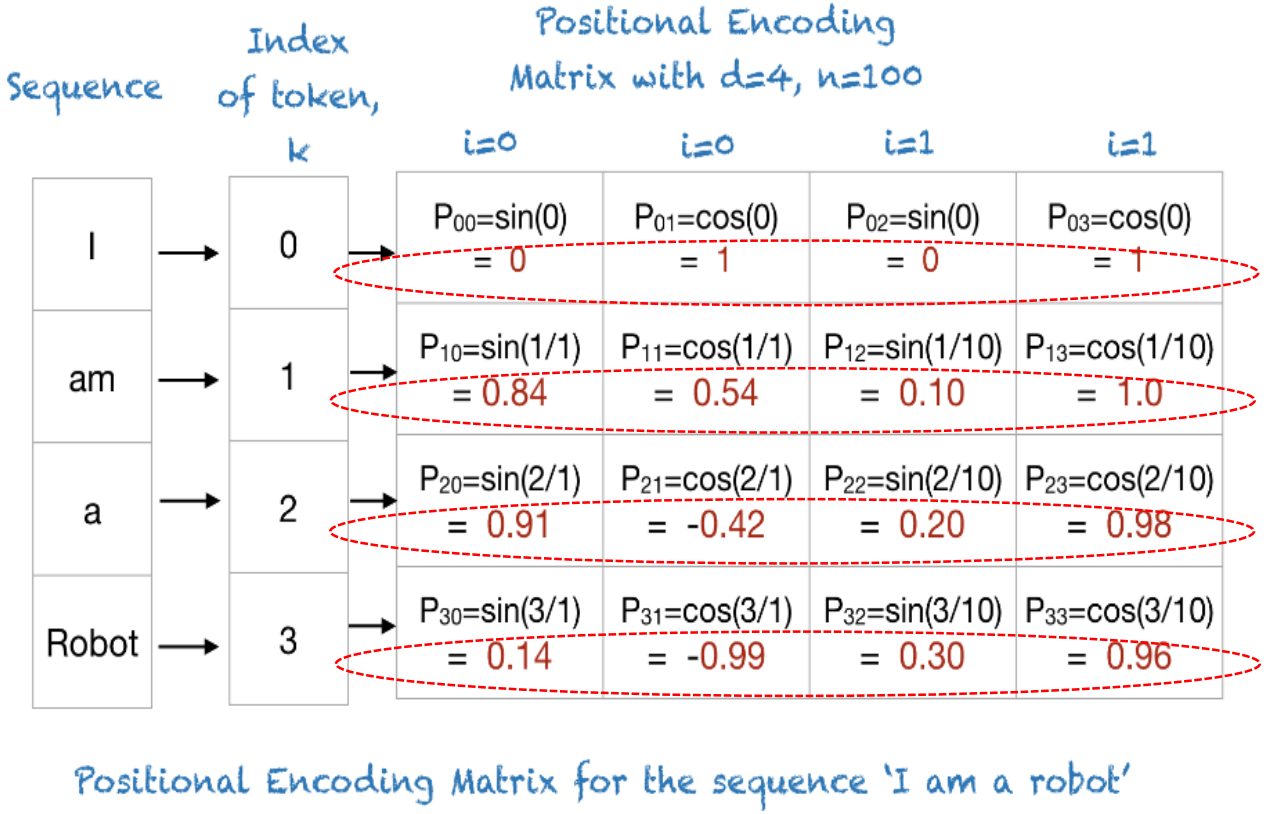
Positional Encoding

- Positional Encoding
  - Usually designed to have a small magnitude compared to the token embedding values, which allows the model to retain the meaningful information from the original token.
- Sinusoidal Positional Encoding

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

- k: position of an object in the input sequence ($0 \le k < L$)
- d: dimension of the output embedding
- P(k,j): position matrix
- n: user-defined scalar (e.g. set to 10,000)
- i: used for mapping to column indices $0 \le i < d/2$, with a single value of i maps to both sine and cosine functions

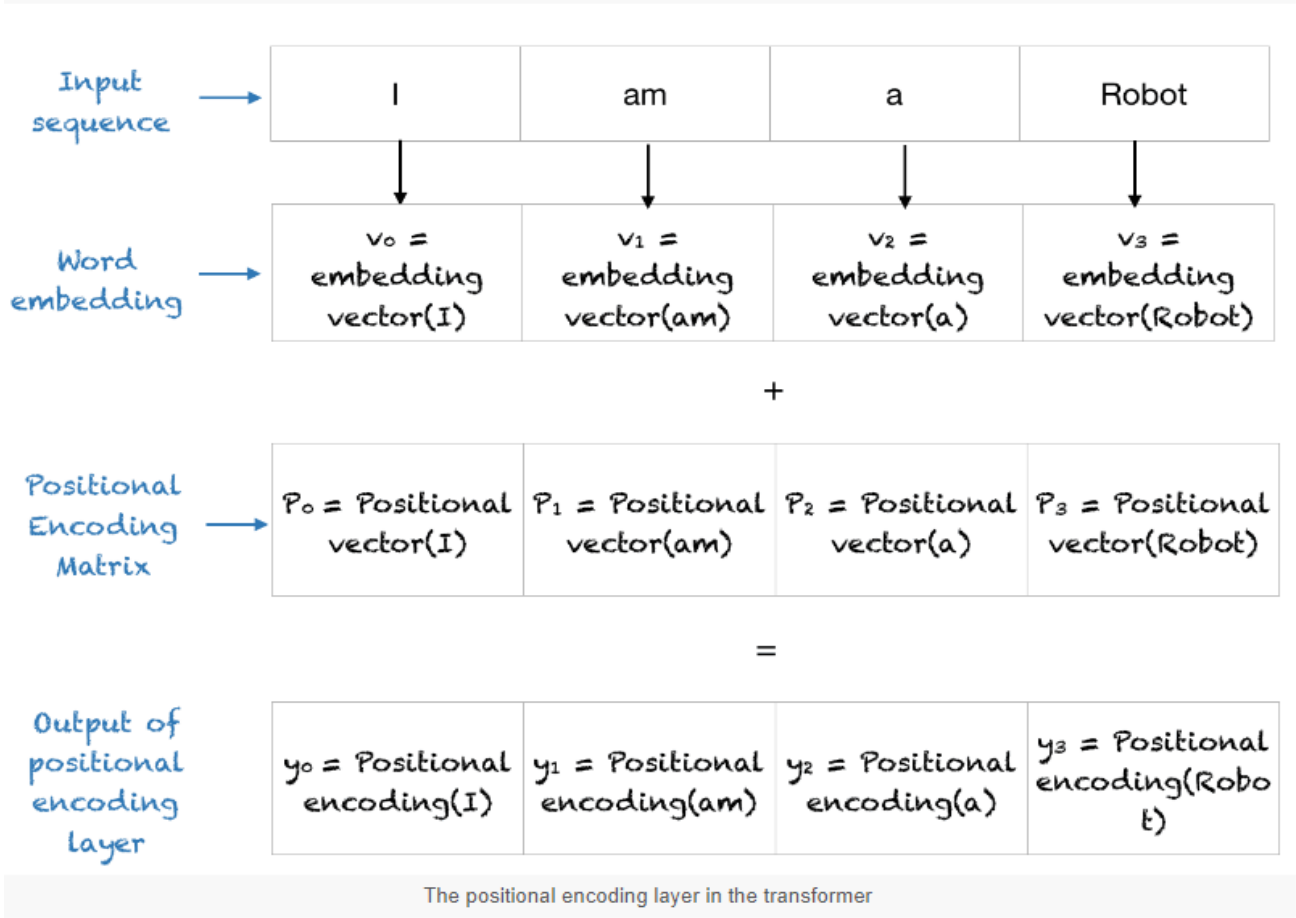


Positional Encoding Matrix for the sequence 'I am a robot'

https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Positional Encoding

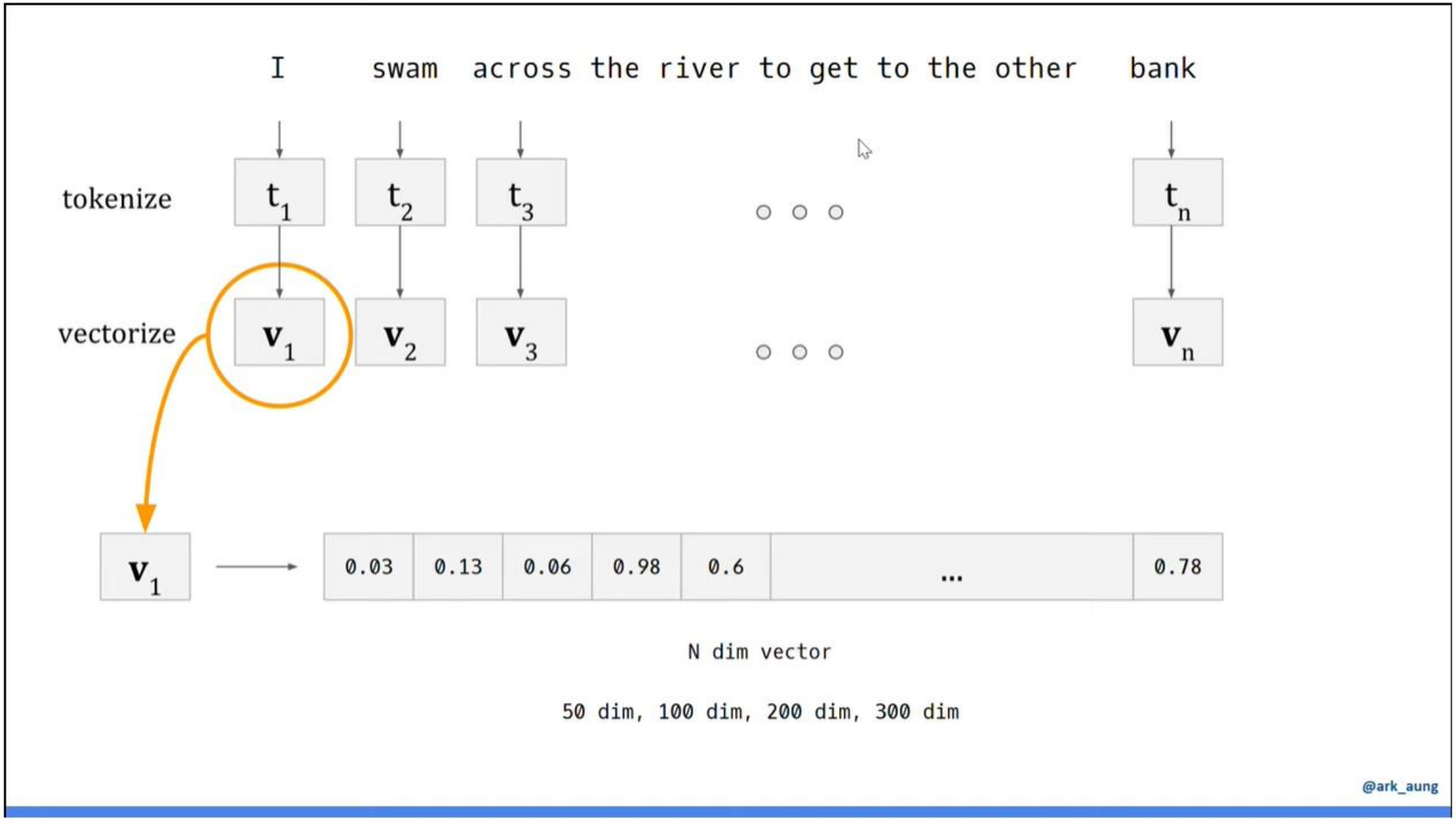
- Final output of the Positional Encoding Layer



The positional encoding layer in the transformer

https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

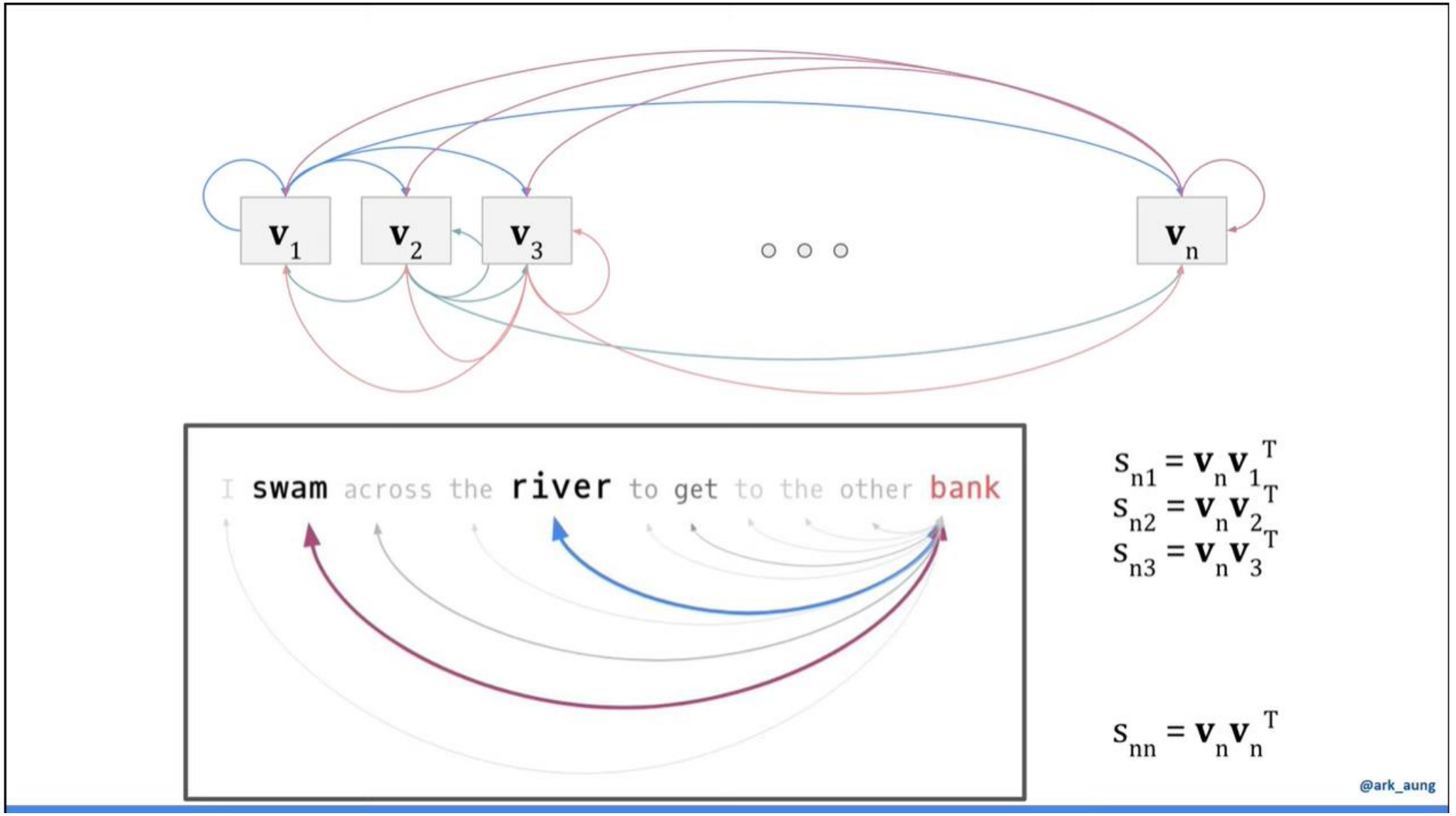
Example: Self-Attention



Intuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

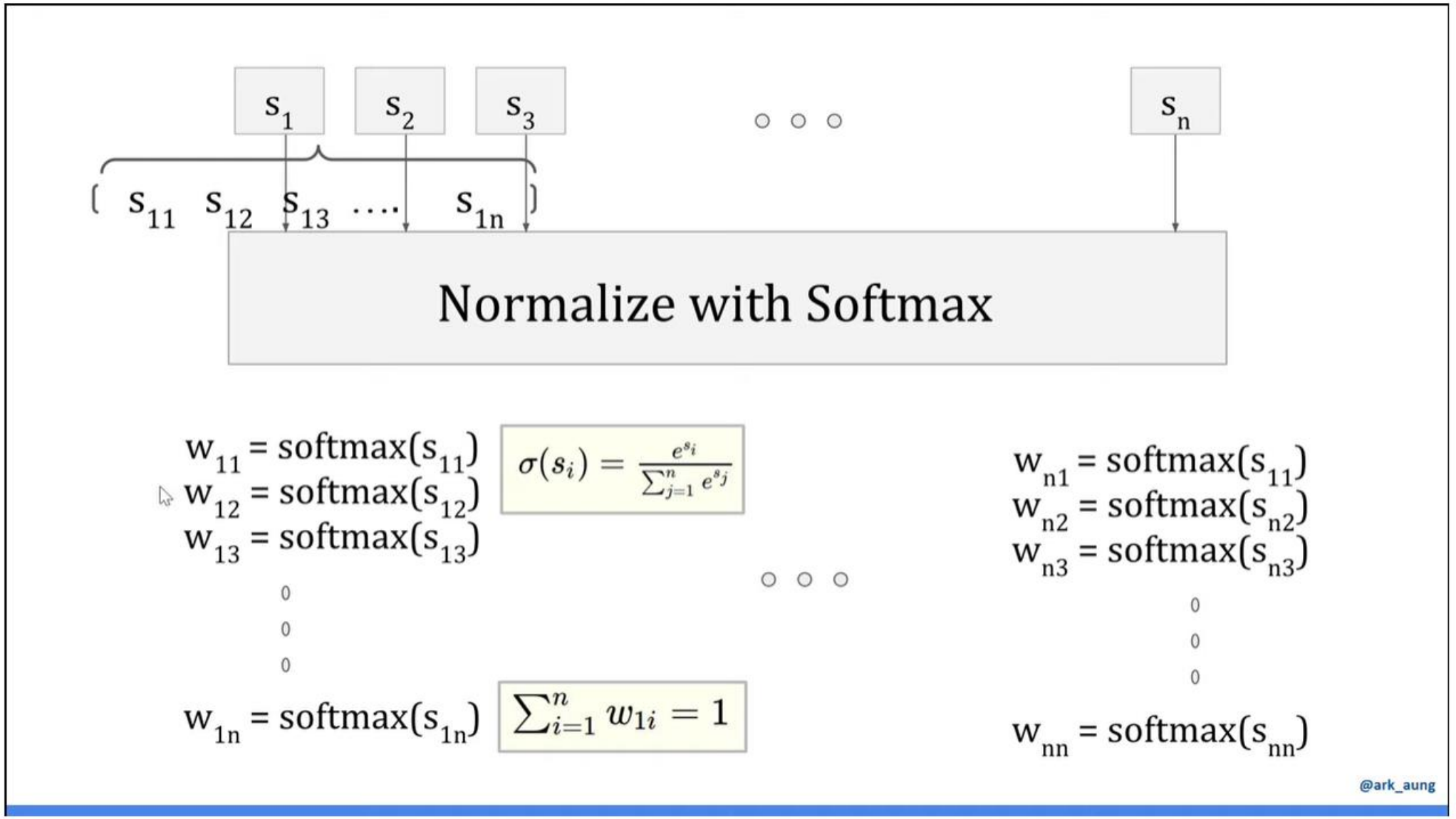
Example: Self-Attention

- Similarity
  - Dot product
  - Cosine similarity



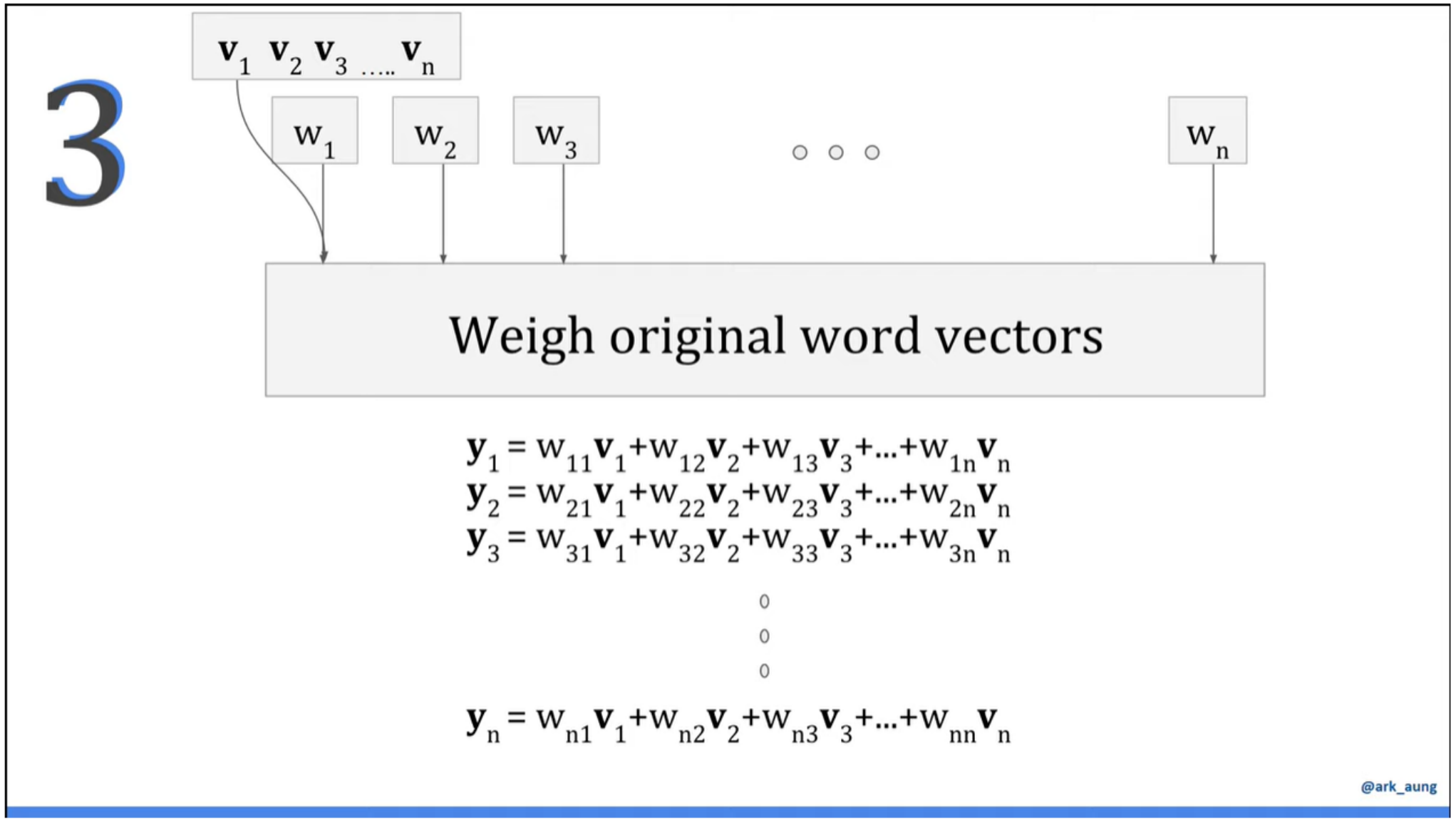
intuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

Example: Self-Attention



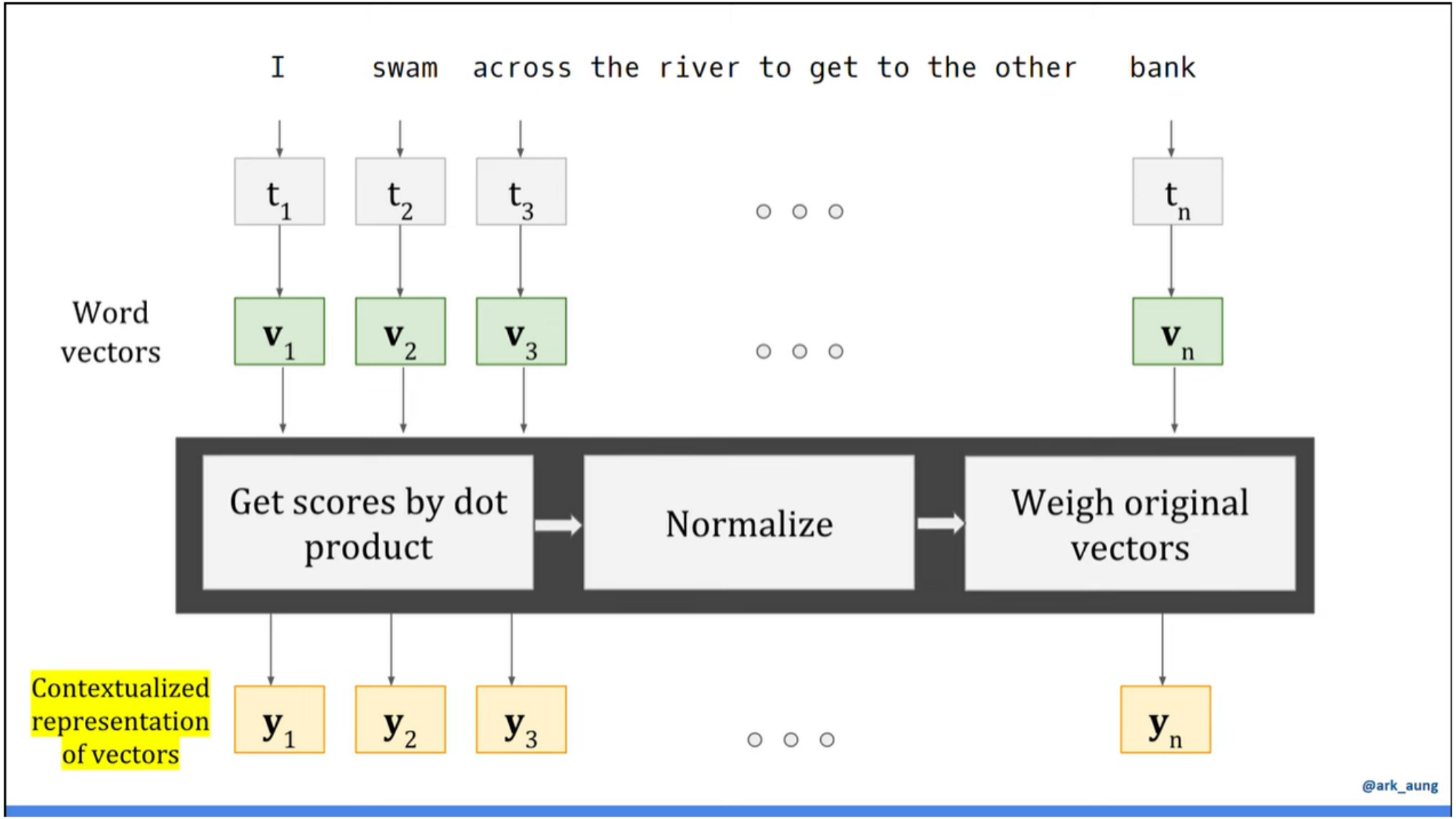
intuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

Example: Self-Attention



Intuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

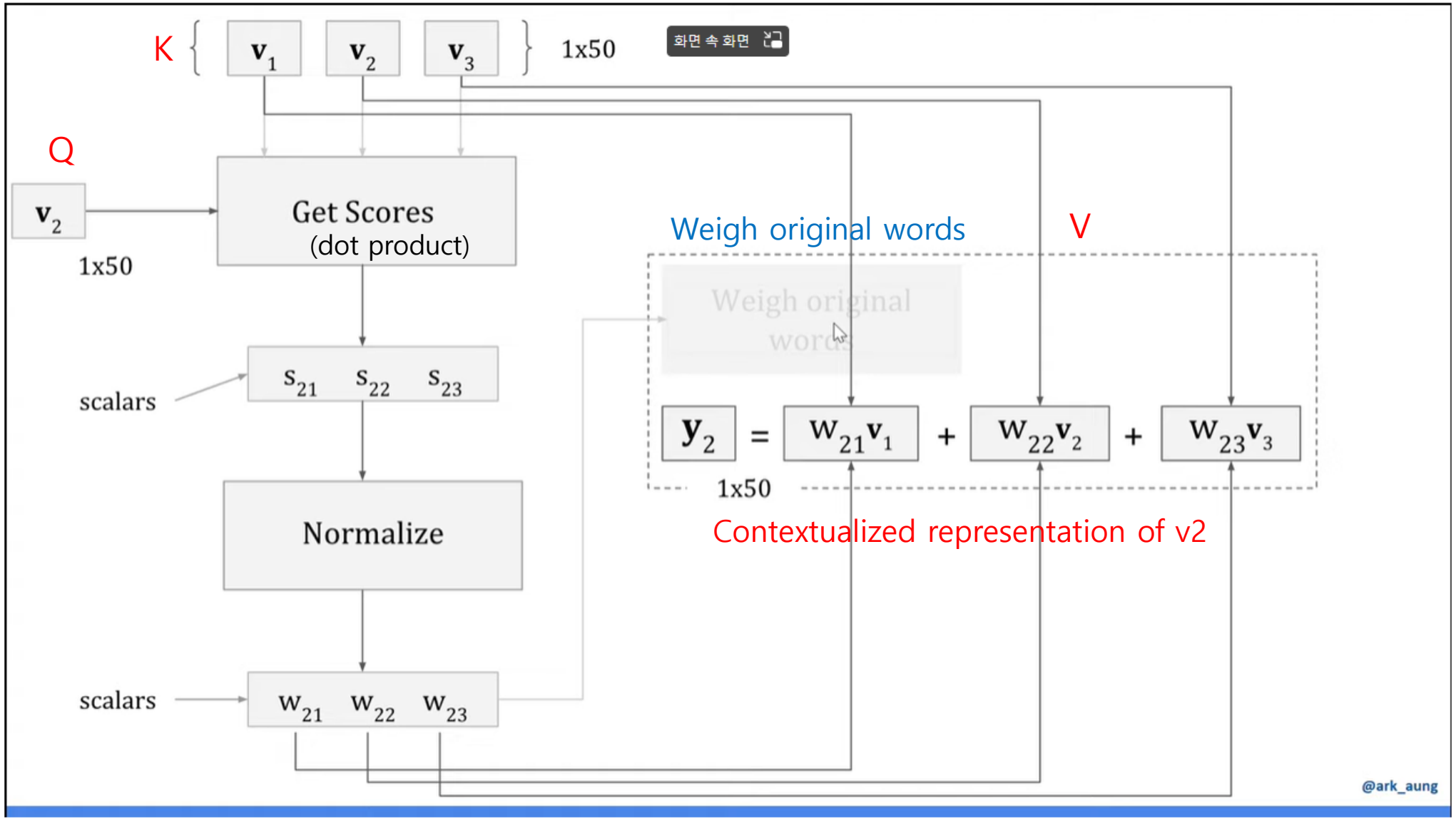
Example: Self-Attention



Intuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

Example: Self-Attention mechanism

- Example
  - 3 words
  - 50-d vectors

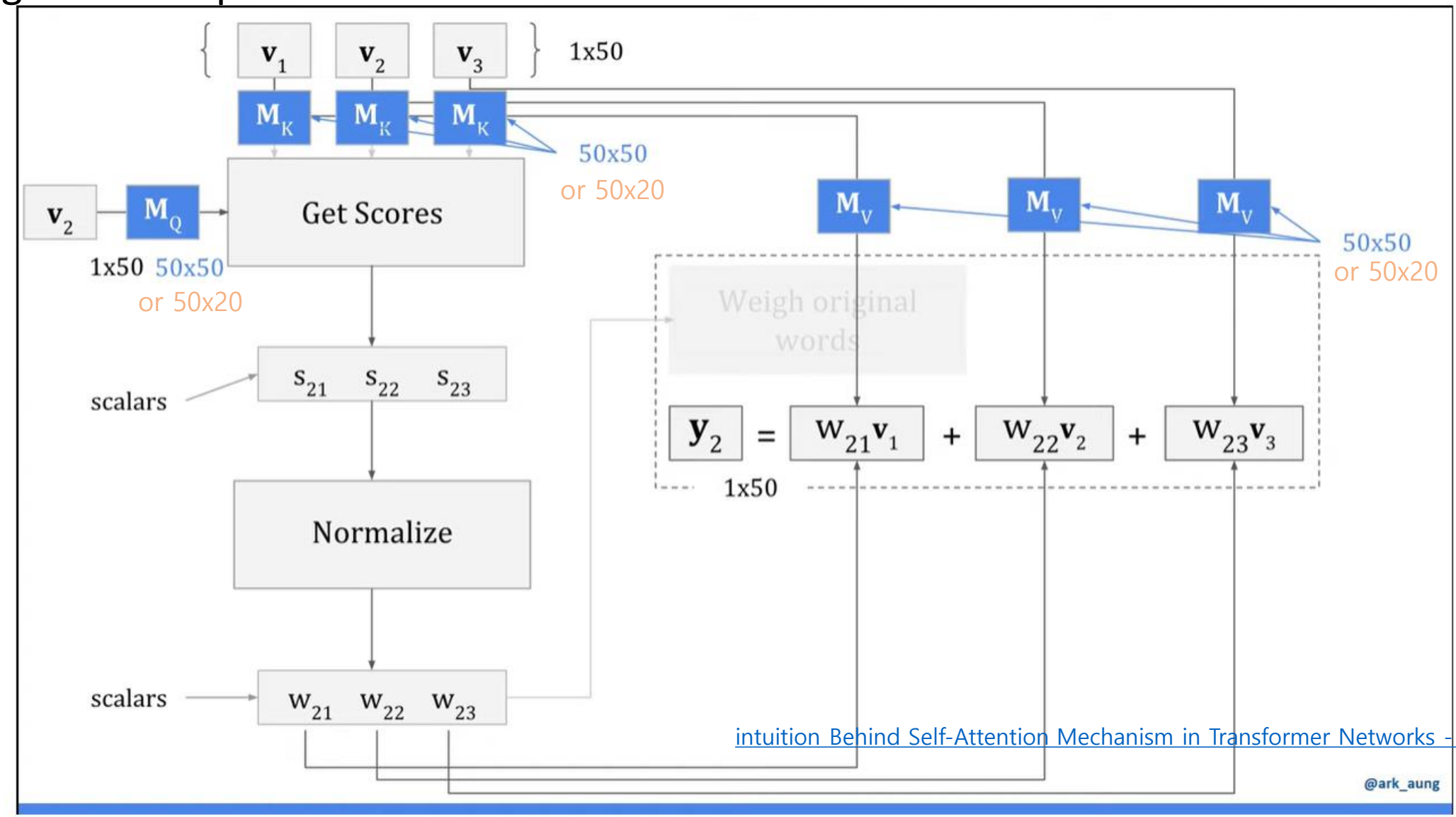


intuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

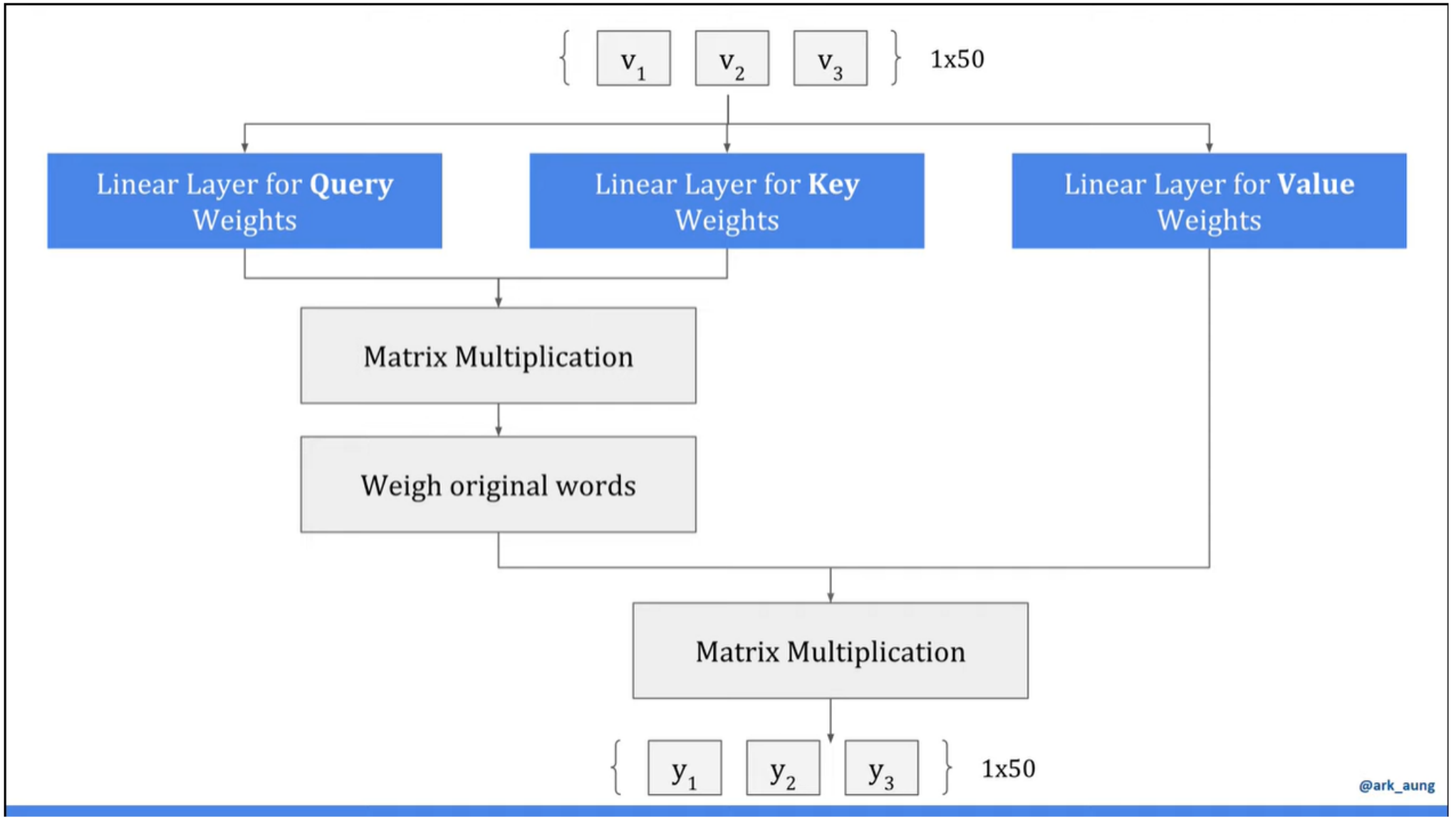
Example: Self-Attention

(*) may have different size

- Injecting learnable parameters

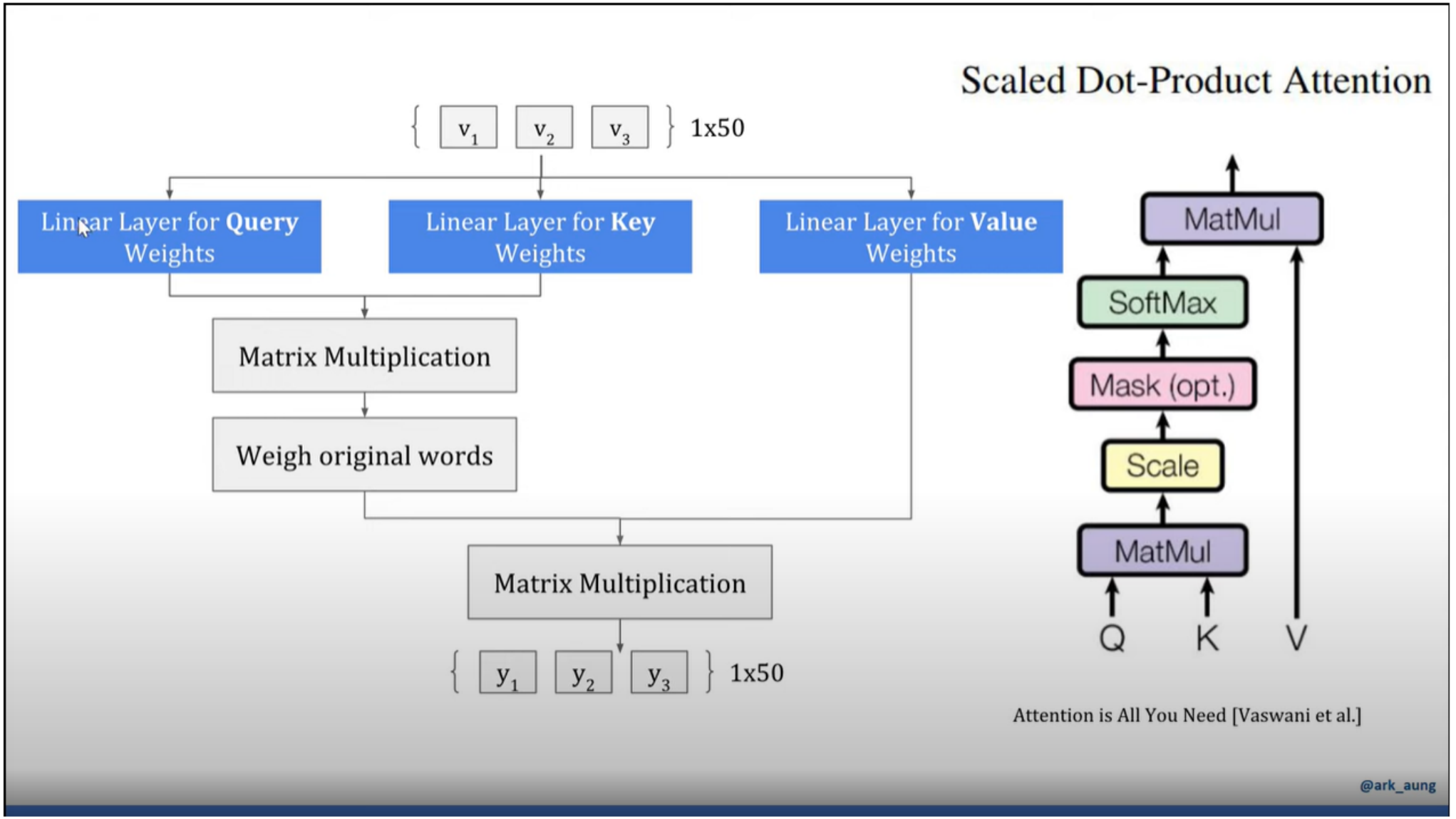


Example: Self-Attention



intuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

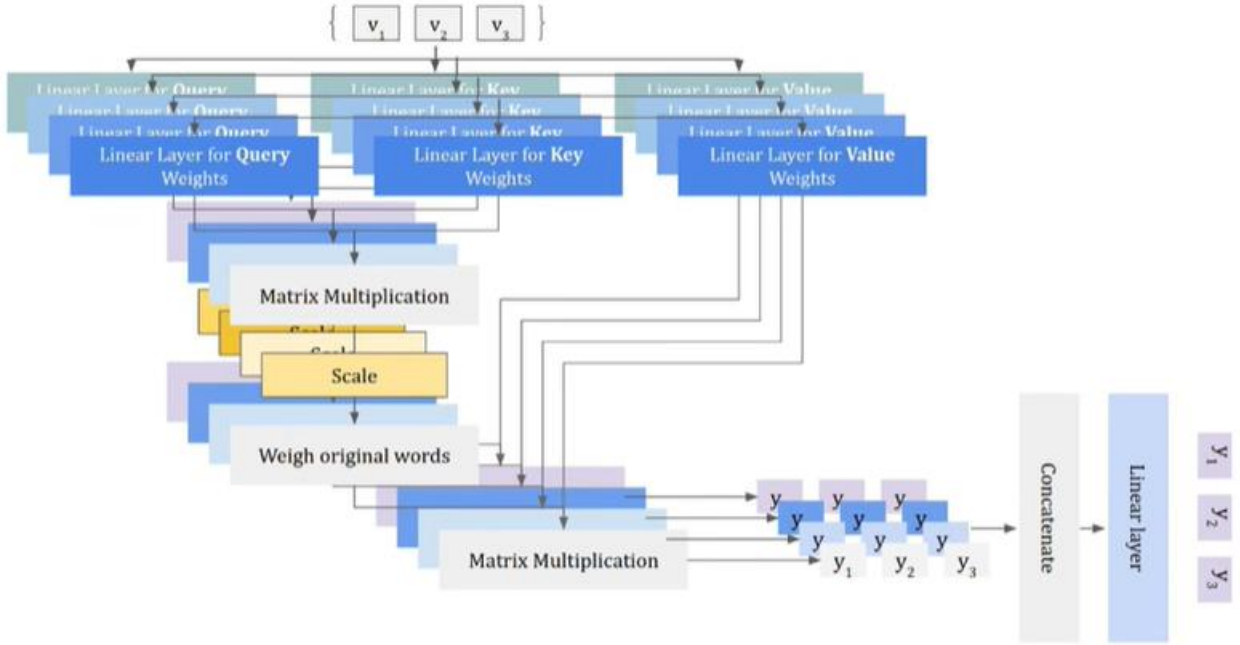
Example: Self-Attention

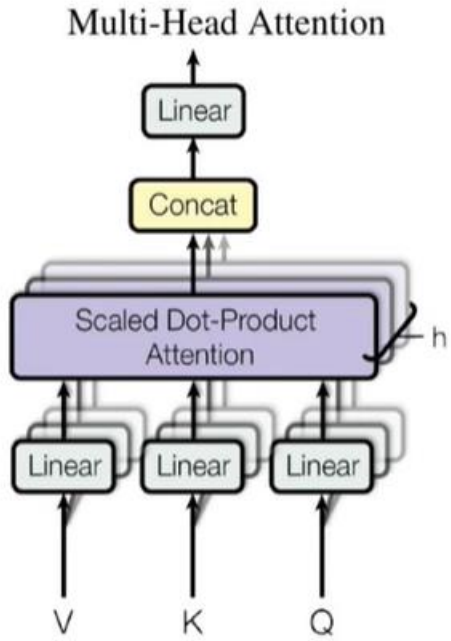


ntuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

Example: Self-Attention

- Multi-head attention
  - Diversity



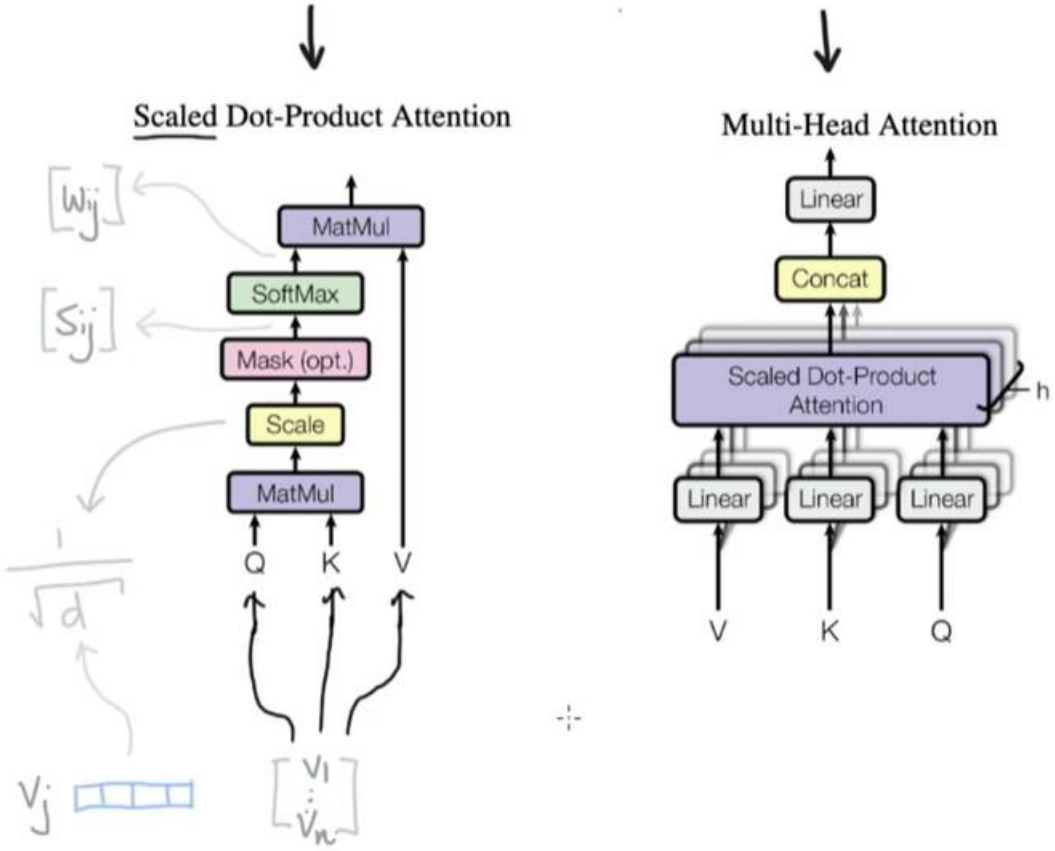


Attention is All You Need [Vaswani et al.]

@ark_aung

ntuition Behind Self-Attention Mechanism in Transformer Networks - YouTube

Transformer



the operation "Attention(Q, K, V)" in the context of the Transformer model means the process of assigning weights or importance scores to different elements in a sequence (or "values") based on their relevance to a specific element (or "query").

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

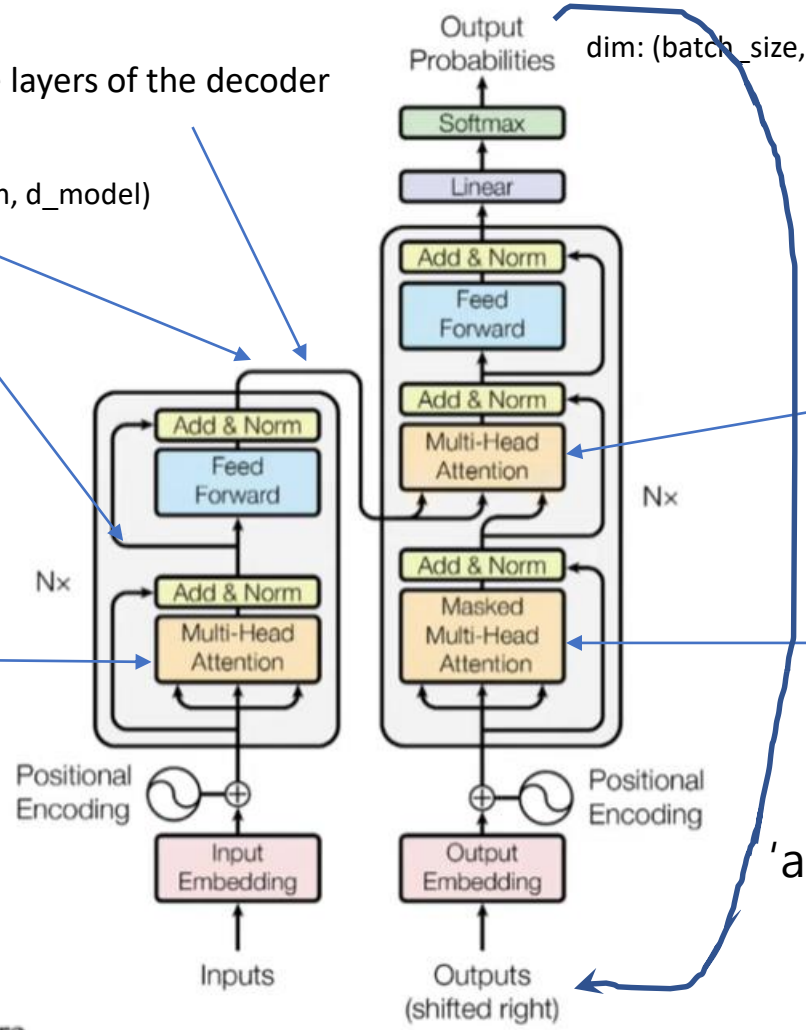
$$W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}, W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}, W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$$

$$W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$$

Transformer

is used as the input for all the layers of the decoder

dim: (batch_size, seq_len, d_model)



dim: (batch_size, seq_len, d_model)

every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to-sequence models (Query: represents the current position to be generated)

Each position in the encoder can attend to all positions in the previous layer of the encoder

each position in the decoder to attend to all positions in the decoder up to and including that position.

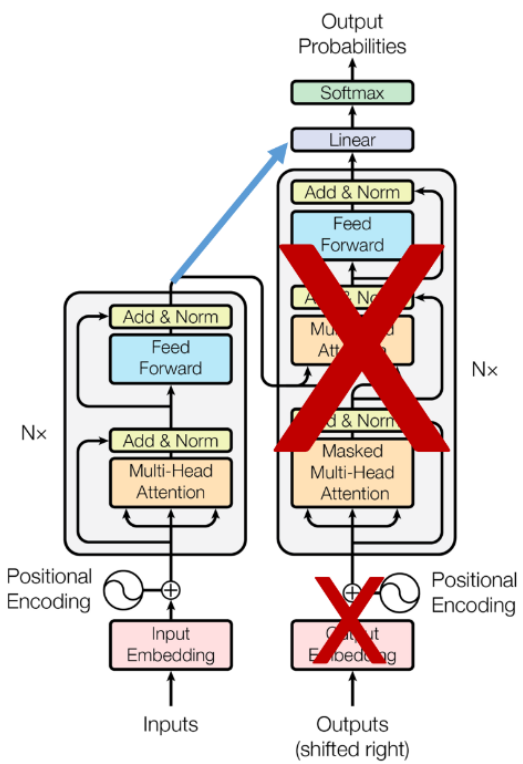
'autoregressive'

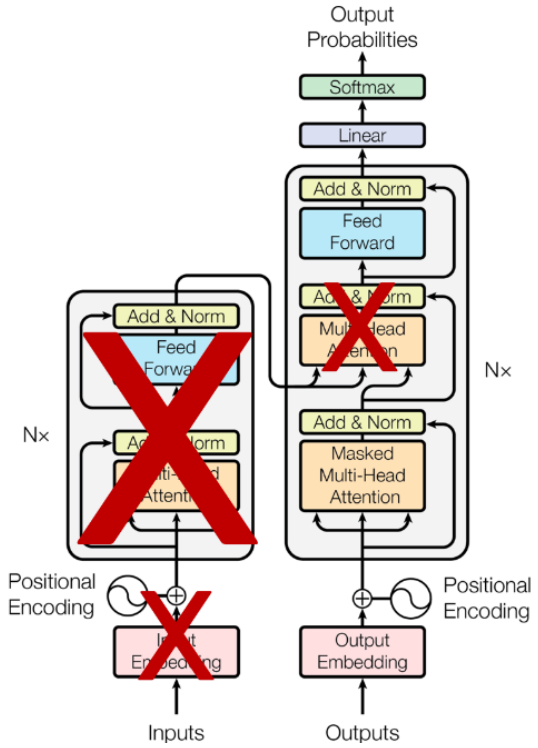
Figure 1: The Transformer - model architecture.

dim: (batch_size, seq_len, d_model) # typically, d_model=d_emb

GPT and BERT

- GPT:
  - Language model (predicts the next word)
  - Unidirectional
- BERT(Bidirectional Encoder Representations from Transformers)
  - Masked Language Model
  - Predicts the blanks in the middle of sentences (bidirectional)



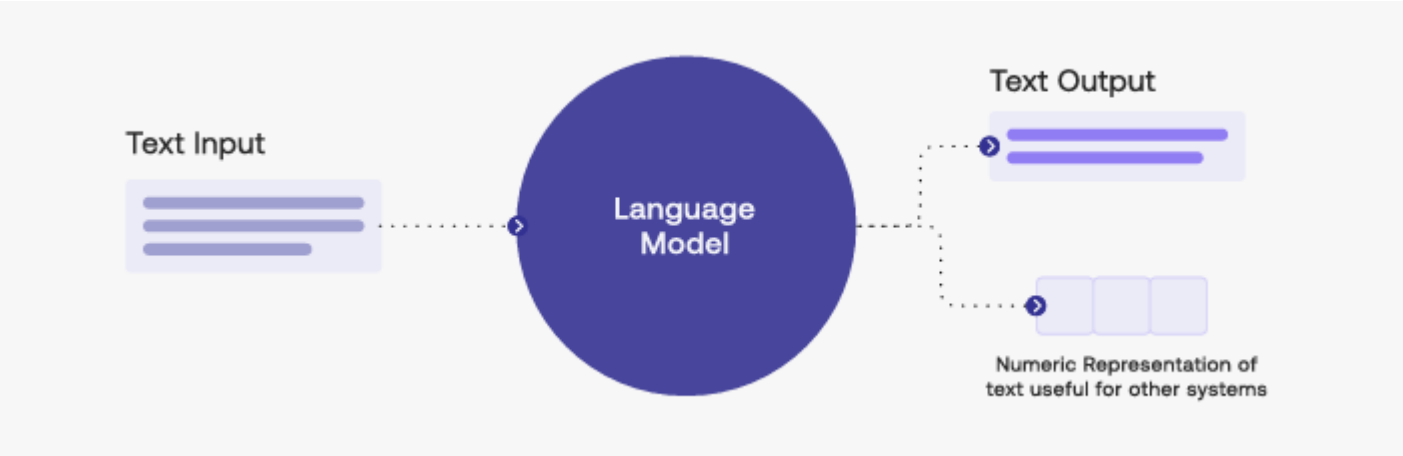


GPT
어제 카페 갔었어 거기 사람 많더라

BERT
어제 카페 갔었어 ☐ 사람 많더라

What is GPT?

- Large Language Model (LLM)
  - Predicts text which should follow from a given prompt using probabilistic methods
  - Machine learning model trained on massive amounts of text



- GPT-3 Language Model
  - 499 Billion tokens (words) drawn from the Web, books, and Wikipedia
  - Trained to predict the next word in a sequence

GPT-3

- 175 Billion parameters
- Trained with 45 TB of text data which includes sources from Wikipedia and books
- It has 96 decoder layers and is built on a system with 285k CPU cores, 10K GPUs, and 400 Gbps network connectivity for each GPU server
- Input sequence
  - Up to 2048 tokens (words encoded as numeric values)
  - Encoding allows word meanings to be compared (e.g., similar words have similar numeric values)
- Output sequence
  - Up to 2048 “guesses” at the most likely next word in the sequence
- Key advantages
  - **Attention** (a technique for determining how important each token is to the output)
- Risk of LLMs (by Dr. Josh Locklair)
  - LLMs absorb the worldview and biases of their human creators
  - LLMs “blindly” propagate information without consideration of human impact
  - LLMs cannot consider the truth of the text they generate

How Transformers are used in ChatGPT?

- **Tokenization**
  - The input text is divided into individual tokens and each token is assigned a unique numerical representation.
- **Embedding**
  - Each token is then transformed into a high-dimensional numerical vector, called an embedding
- **Positional Encoding**
- **Transformer Layers** (Attention Mechanism + Feedforward Network)
- **Decoder**
  - It takes the transformed input sequence and generates the output sequence token by token.
  - At each step, the model uses a combination of the previously generated tokens and the contextual information from the input sequence to predict the next token.
- **Generation**
  - The output tokens are decoded back into readable text, and the process can be repeated to generate a response or continuation.