Attention Mechanism and Transformers

Team 6:

Dominik Soós

Sushmitha Halli Sudhakara

Table of Contents

- Transformers Introduction
- Section 1 – Attention Mechanism
  - – Scaled Dot-Product Attention
  - – Multi-Head Attention
  - – Limitations of Full Attention
- Section 2 - Transformer Model Architecture
  - – The Encoder
  - – The Decoder
  - – Transformer Training and Inference
  - – Advantages and Challenges
- Section 3 – Sparse Attention Mechanism
  - – Advantages
  - – Sparse attention in transformers

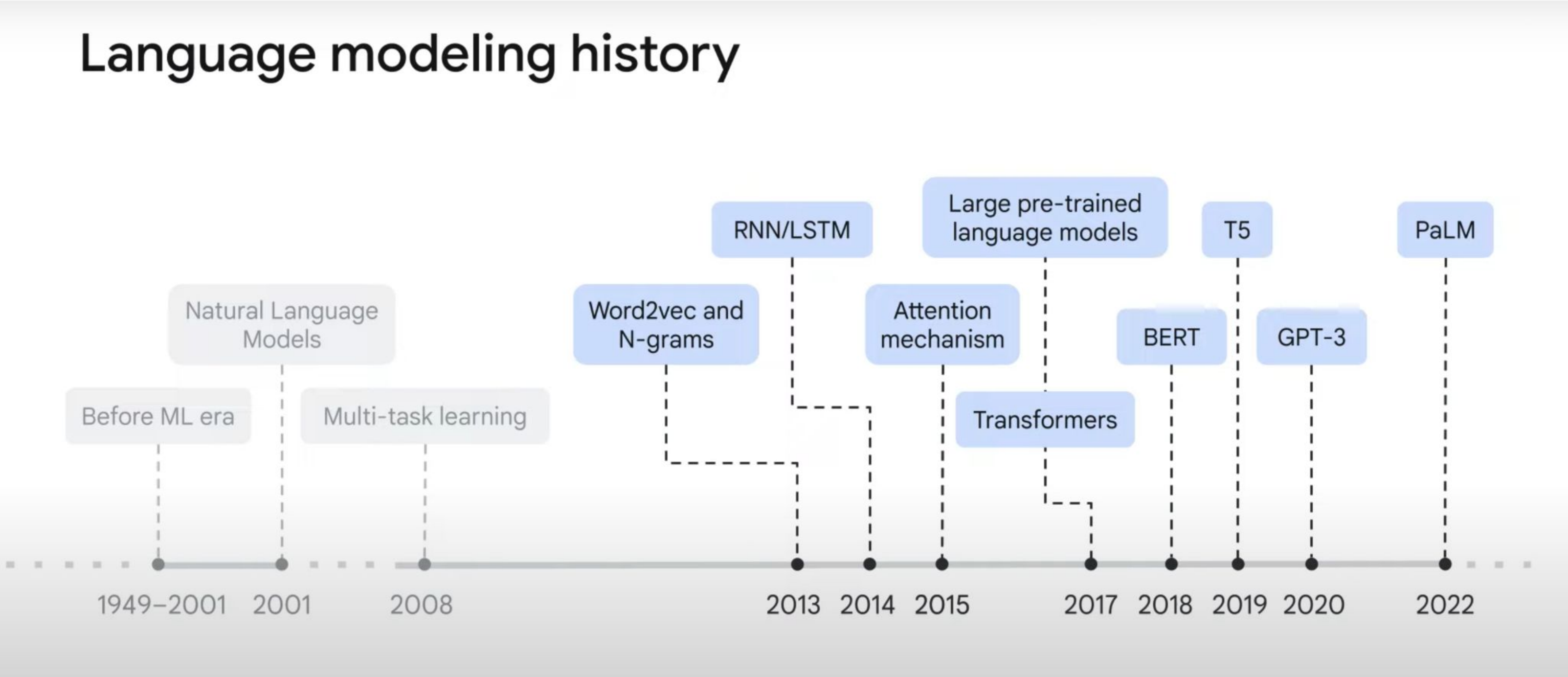
Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

History



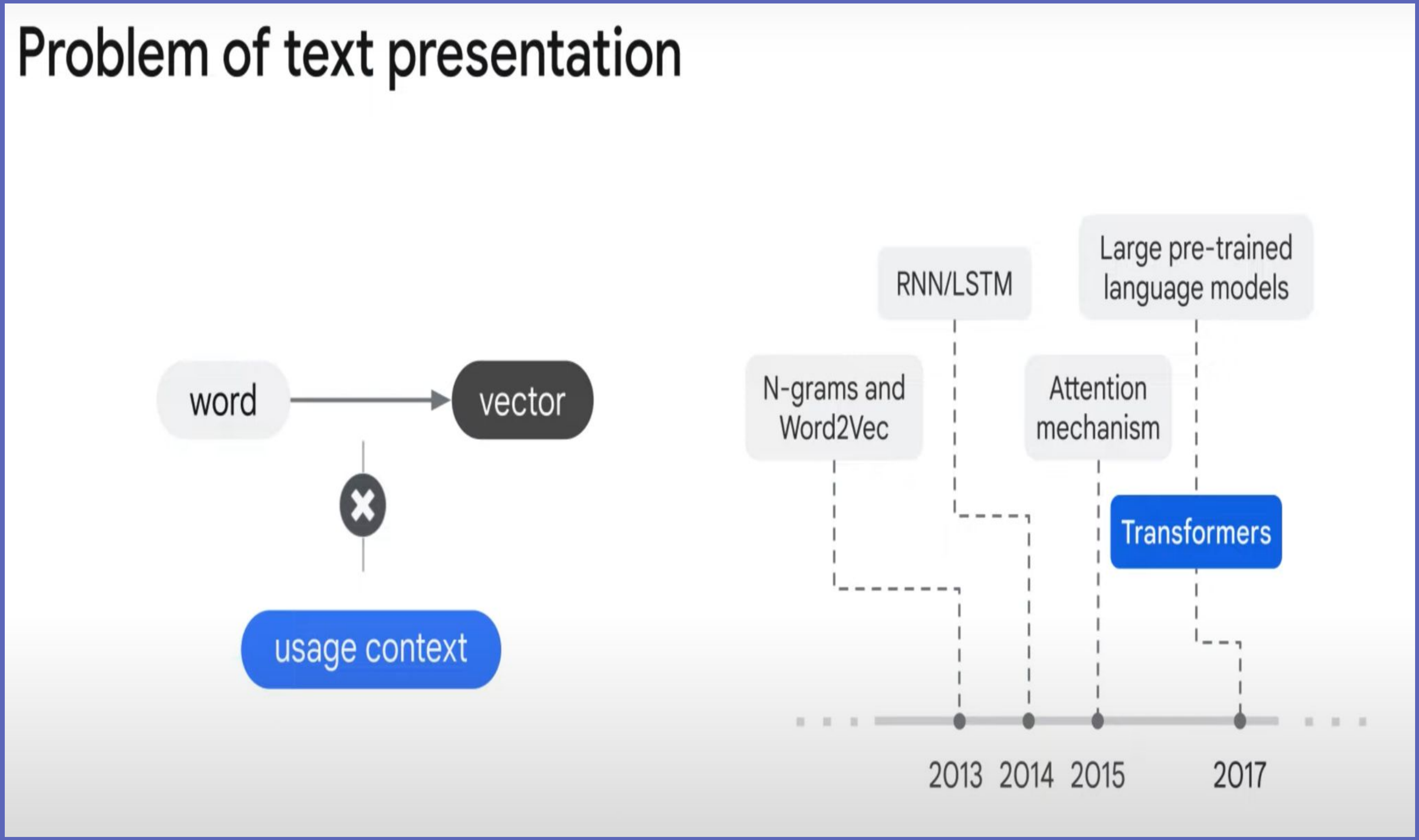
Pre-Transformer Era (RNNs, LSTM, GRUs)

What Worked:

- Encoding history

Challenges:

- Handling long sequences
- Capturing context effectively



Context in Language Models Before Transformers:

- Models represented words as vectors.
- Context was not included in these vectors.
- Example: The word "bank" in "river bank" and "bank robber" had the same vector.

The Impact of Attention Mechanisms:

- Attention mechanisms introduced context sensitivity.
- Now, "bank" in different sentences can have unique representations.

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

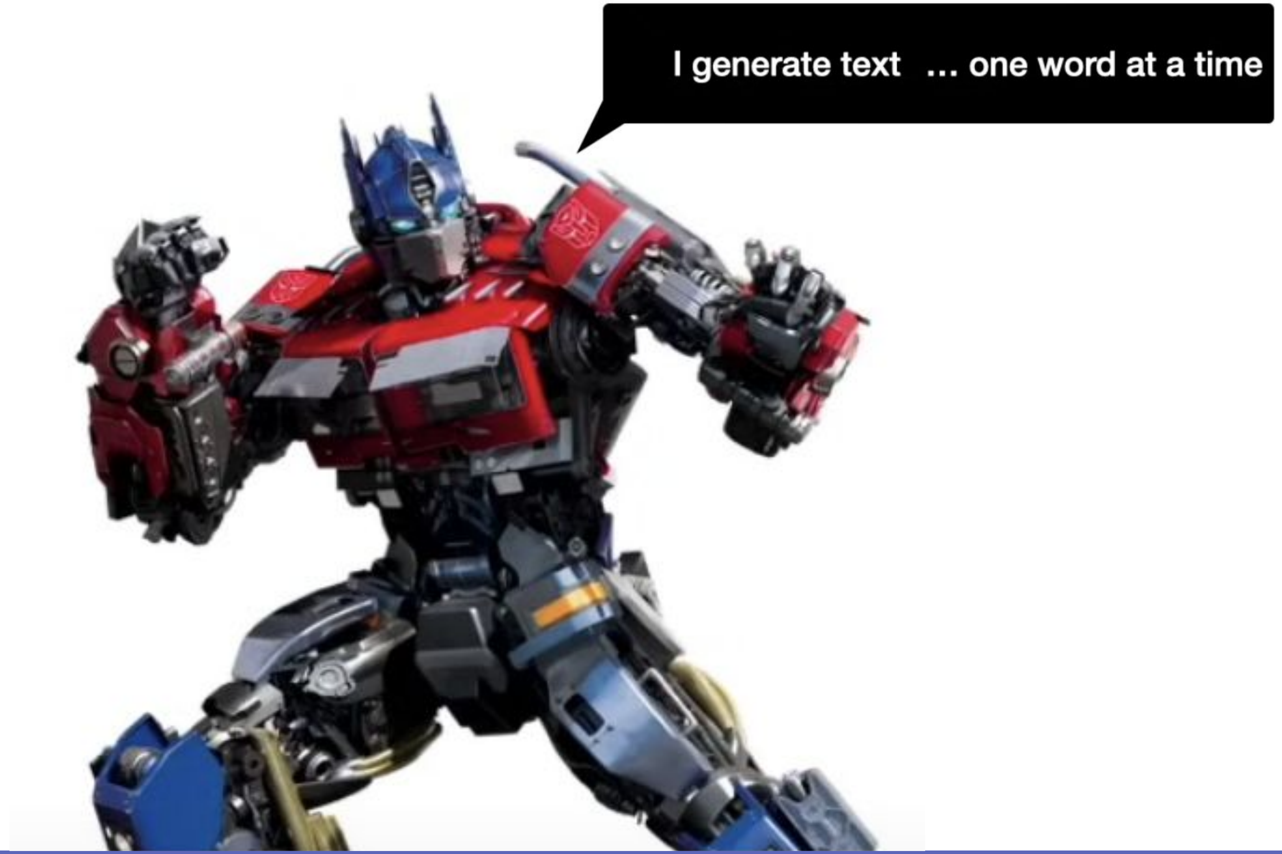
Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention



Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Transformer

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Transformer

Introduction
1 Attention Mechanism
2 Transformer Architecture
3 Sparse Attention

Self-Attention

– The context of the sentences help resolve ambiguities

– Query: What should I focus on?
– Key: How much should I focus?

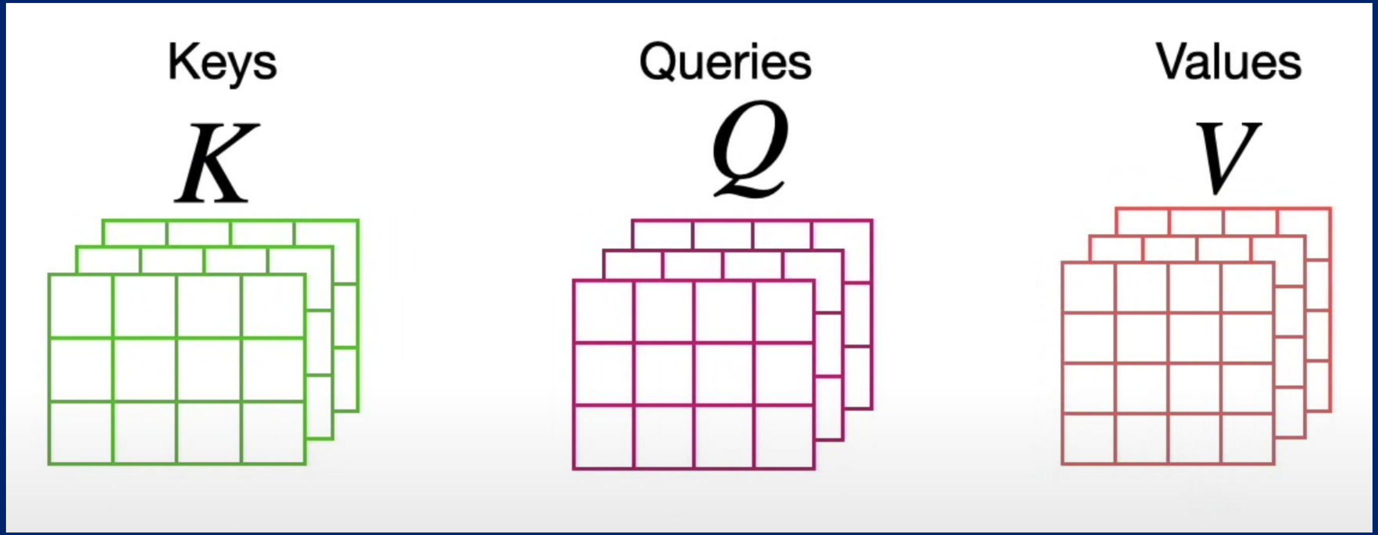


Figure 3. Illustration of Key, Query and Value matrices [5]

Similarity Measures

1.
2.
3.
4.
5.

- Dot Product
- Cosine similarity
- Scaled Dot-Product

orange, cherry, phone

Cosine similarity = dot product, up to a scalar

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

where θ is the angle between $\mathbf{a}$ and $\mathbf{b}$.

Figure 4. Definition of dot product [1]

$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}},$$

where A_i and B_i are the ith components of vectors $\mathbf{A}$ and $\mathbf{B}$, respectively.

Figure 5. Definition of cosine similarity [2]

Scaled Dot-Product Attention

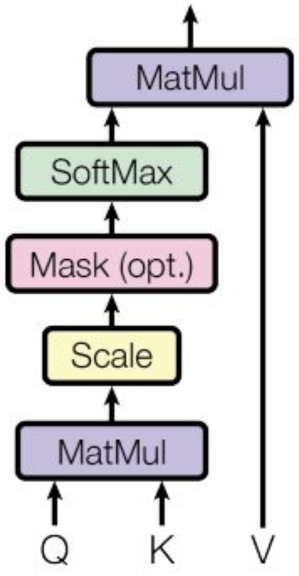


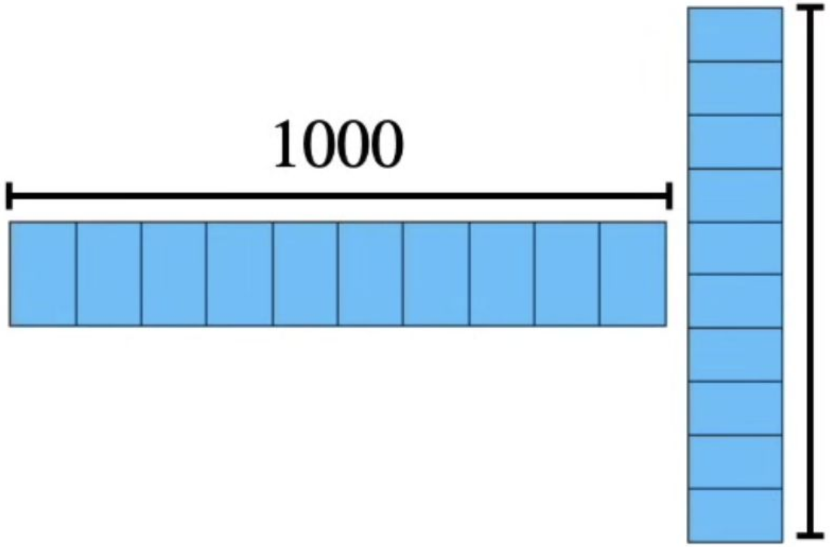
Figure 6. Scaled Dot Product Attention [3]

– Dot product divided by the square root of the length of the vector

– an apple and an orange

– an apple phone

– move ambiguous apple



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

Scaled Dot-Product Attention

1.

2.

3.

4.

5.

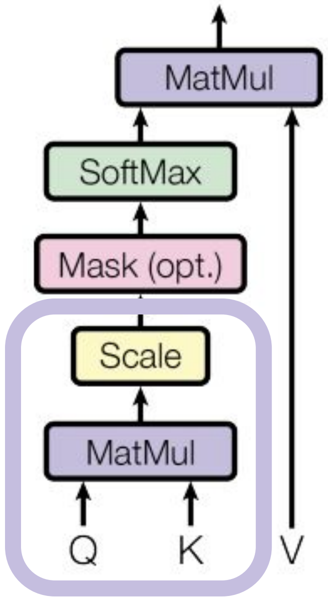
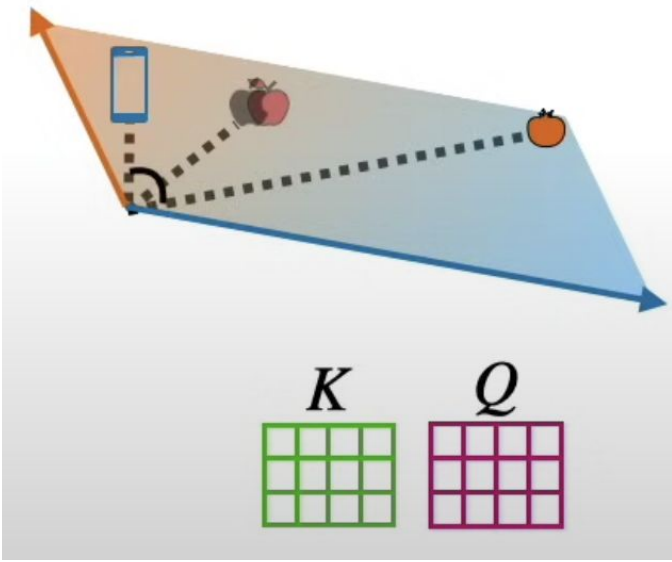


Figure 6. Scaled Dot Product Attention [3]

– Dot product divided by the square root of the length of the vector

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\boxed{\frac{QK^T}{\sqrt{d_k}}}\right)V$$



Scaled Dot-Product Attention

1.
2.
3.
4.
5.

– Dot product divided by the square root of the length of the vector

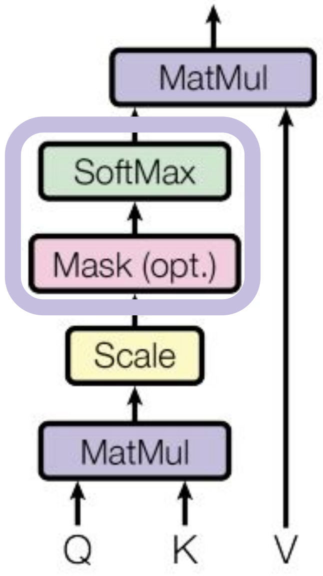
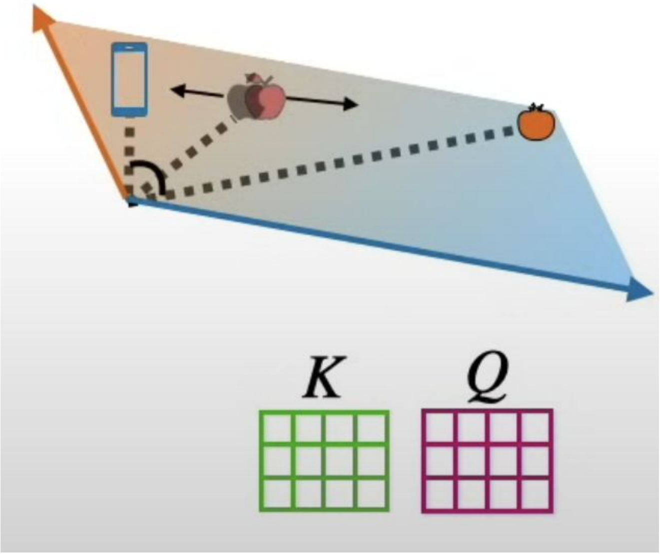


Figure 6. Scaled Dot Product Attention [3]

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



Scaled Dot-Product Attention

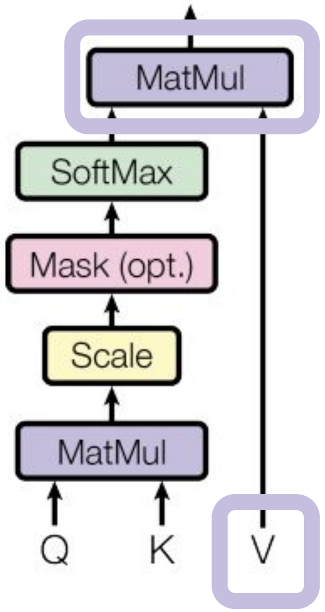
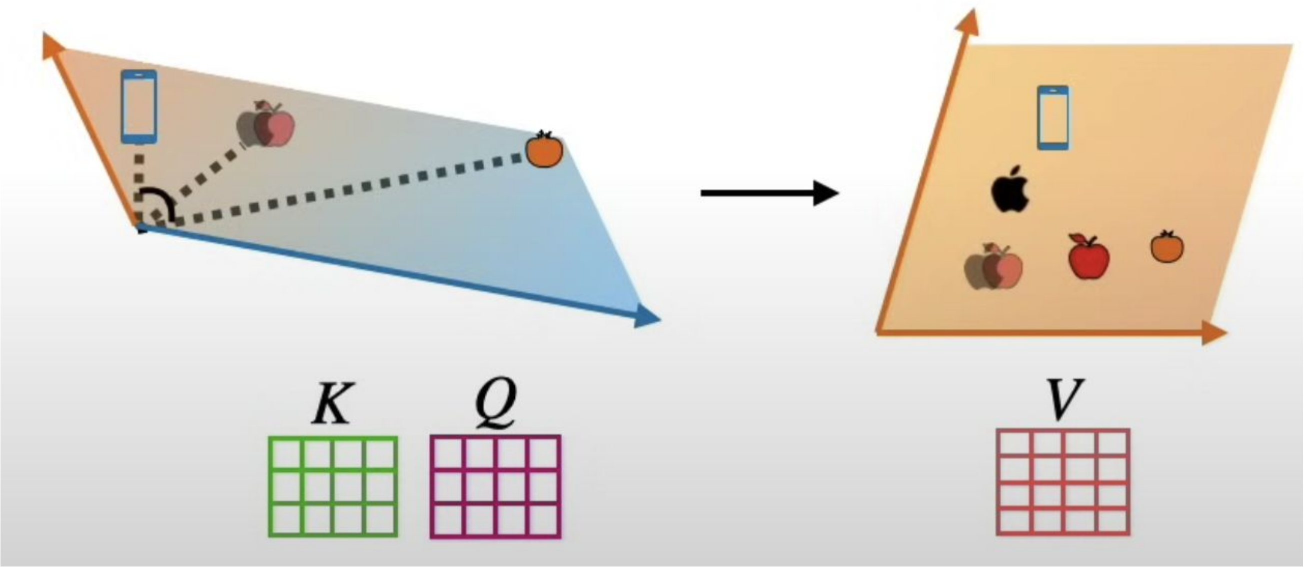


Figure 6. Scaled Dot Product Attention [3]

– Dot product divided by the square root of the length of the vector

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\boxed{V}$$



Multi-Head Attention

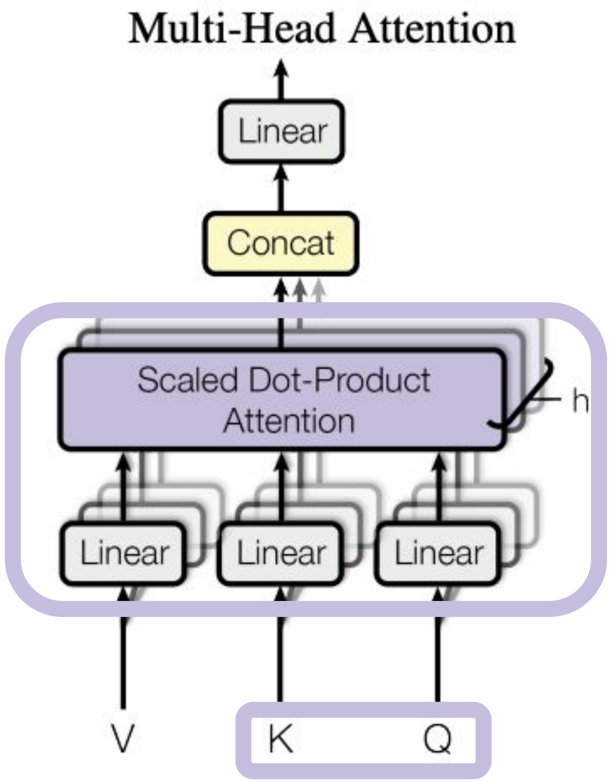


Figure 7. Multi-Head Attention [3]

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Which one do you think is the best linear transformation?

A B C

Multi-Head Attention

1.
2.
3.
4.
5.

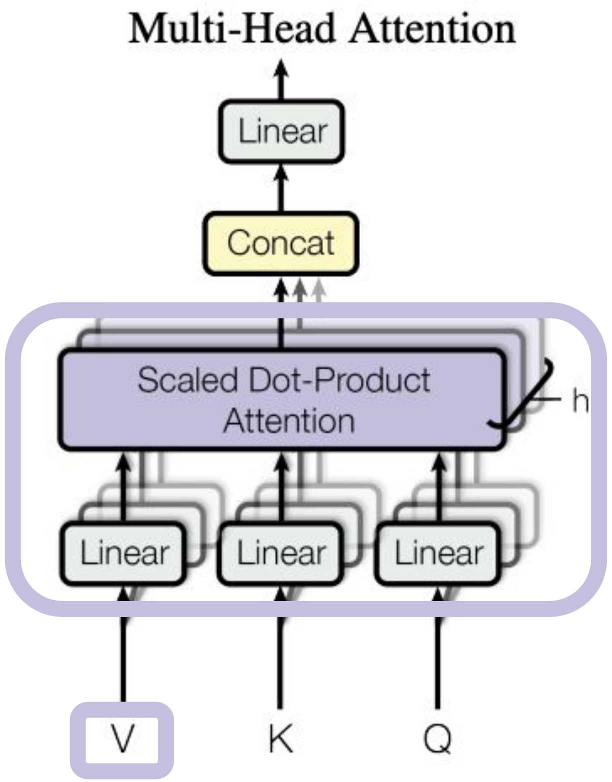
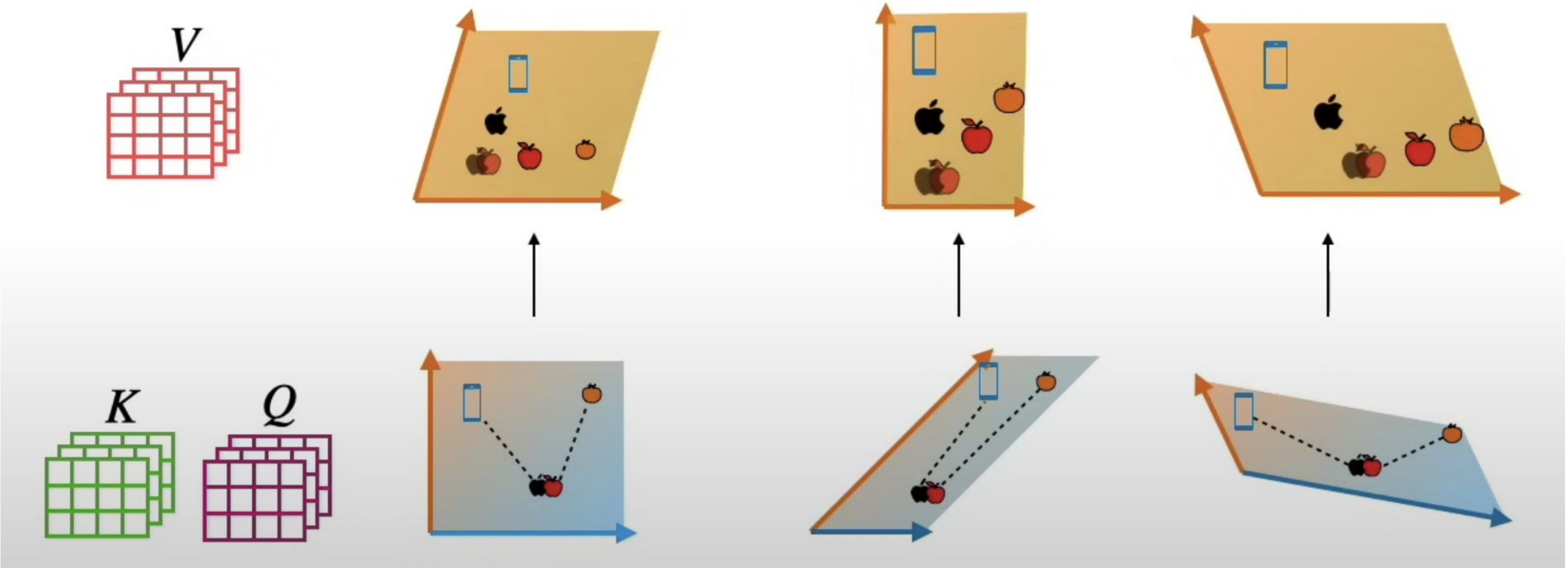


Figure 7. Multi-Head Attention [3]

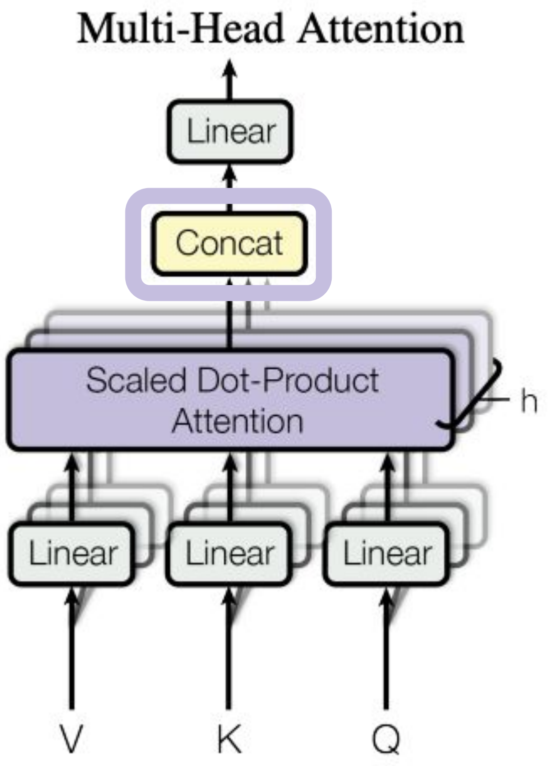
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

How do we know which one is the best embedding?



Multi-Head Attention

1.
2.
3.
4.
5.



$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

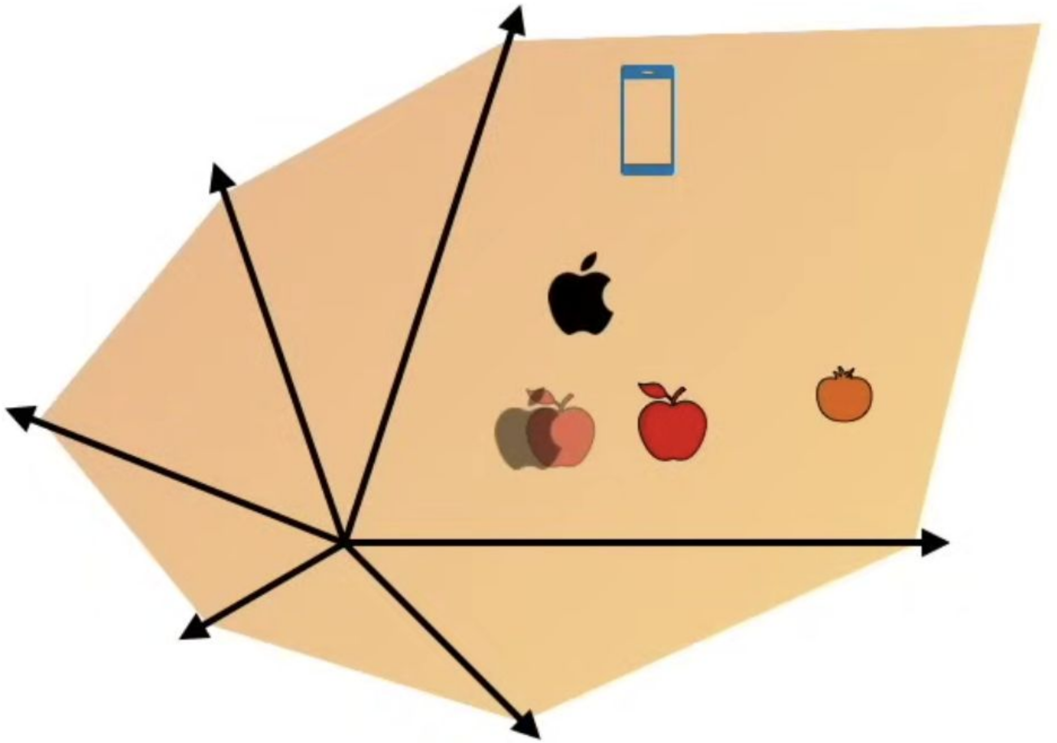


Figure 7. Multi-Head Attention [3]

Multi-Head Attention

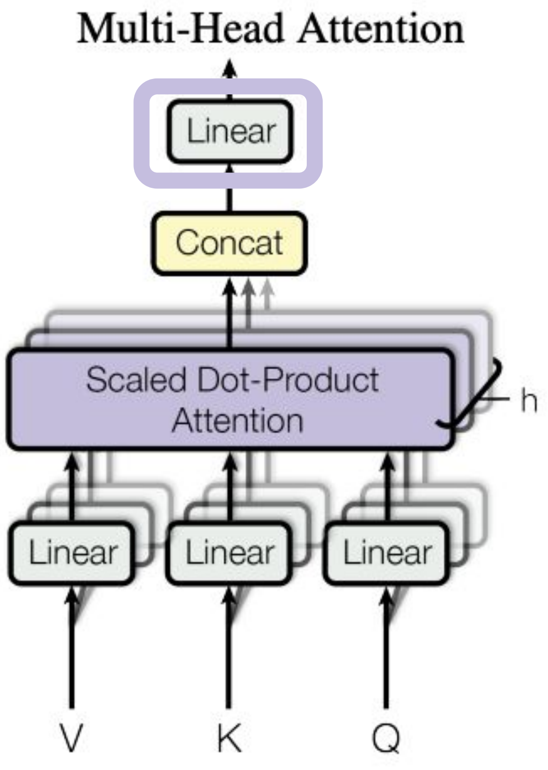
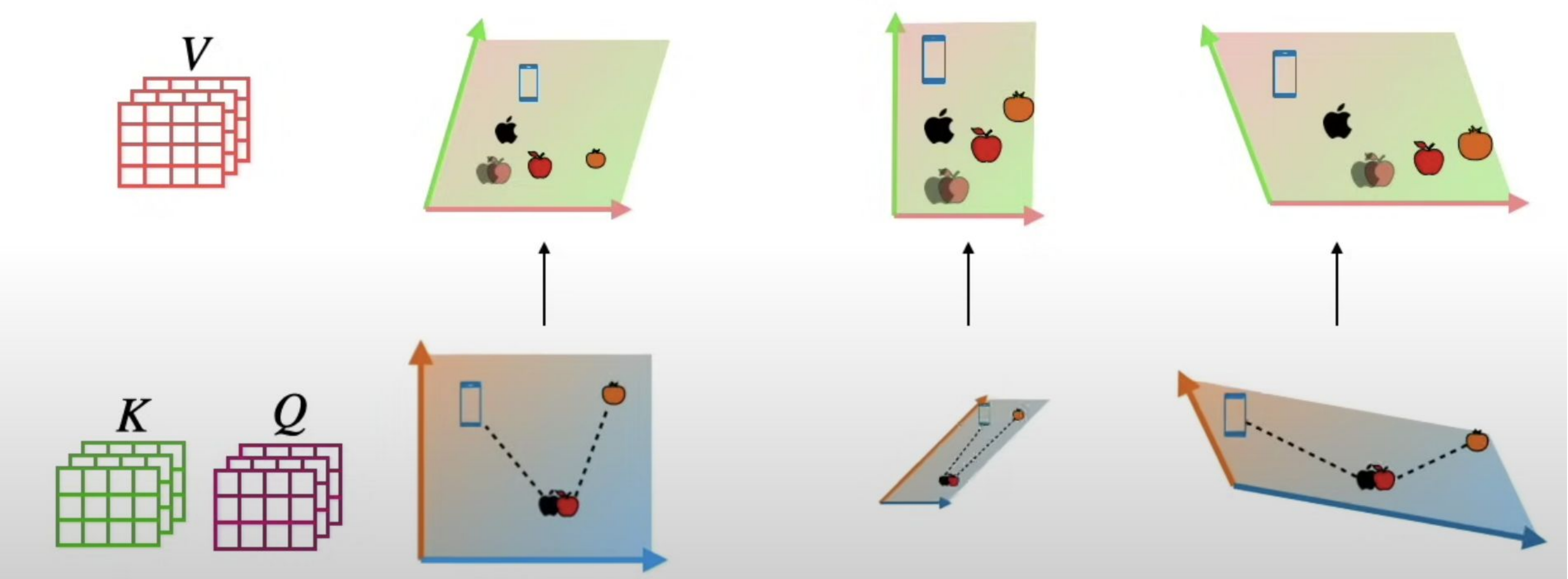


Figure 7. Multi-Head Attention [3]

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Matrix that learns which linear transformations are better → scale



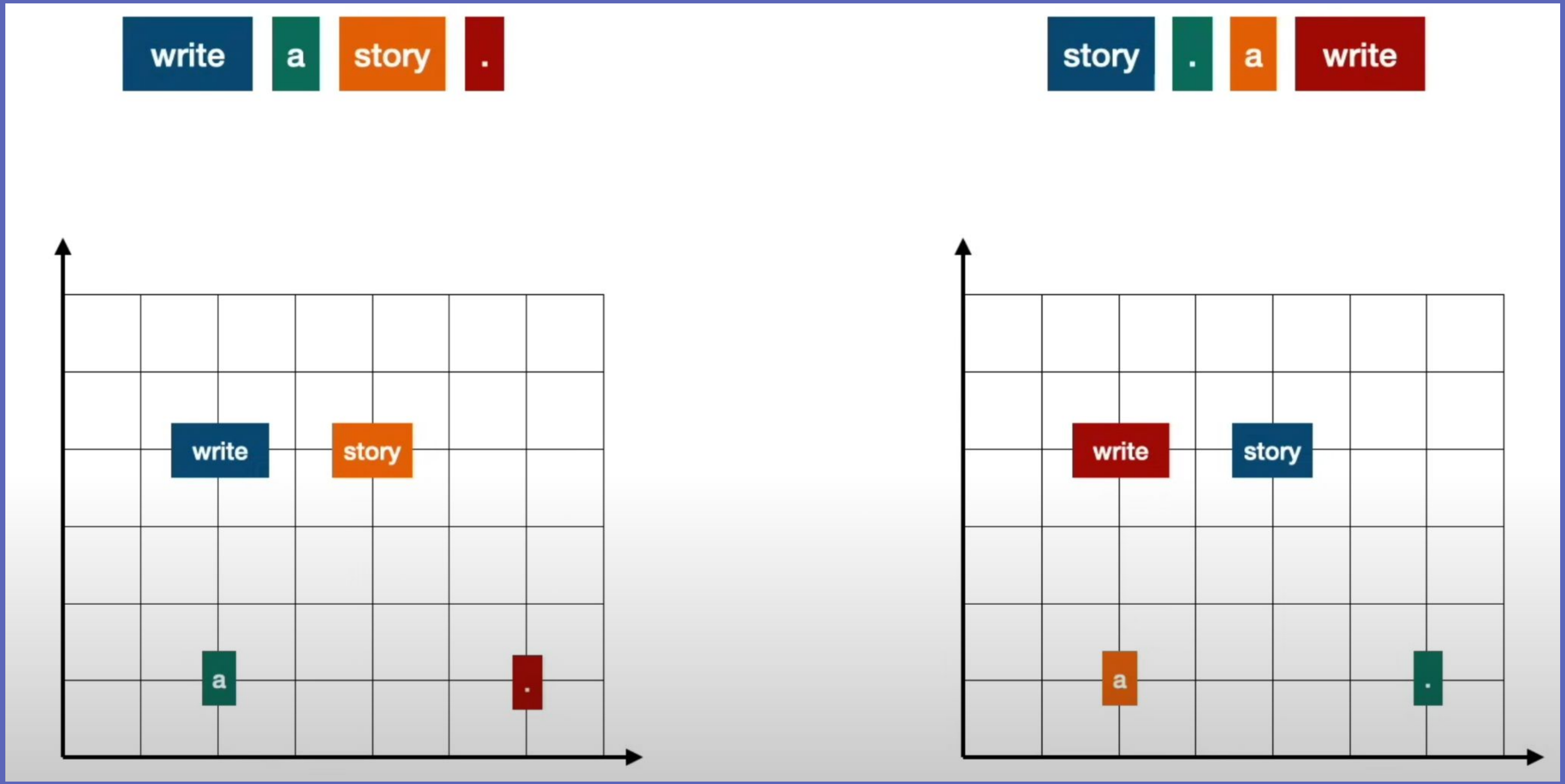
Positional Encodings

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

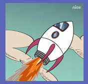


Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention



Introducing Transformers

What They Are: A cutting-edge model transforming how machines understand language.

The Magic Inside: Uses a special "**attention mechanism**" to know which words to focus on.

Structure: Built with two powerful parts - **the encoder** reads the text, **the decoder** predicts the future.

Power of Parallel Processing: Unlike their predecessors (RNNs & LSTMs), **transformers process words all at once**, not one by one.

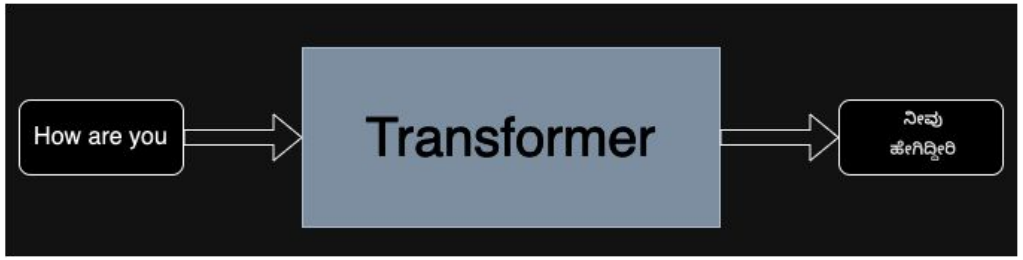
Overcoming Challenges: Say goodbye to slow processing! They leap over the sequential bottleneck that held back previous models.

Drawbacks: Expensive computation of the nxn (embedding dimension) attention matrix.

Model Architecture - A High level overview of transformer model

The Transformer architecture **excels at handling text data** which is inherently sequential.

They take a text sequence as input and produce another text sequence as output. eg. to translate an input English sentence to Kannada.



Core Components:

- **Encoder Layers:** Analyze the input text, understanding its context and meaning.
- **Decoder Layers:** Generate the transformed output text based on the encoder's analysis.
- **Embedding Layers:** Convert words into numerical data for both Encoder and Decoder, facilitating deeper understanding.
- **Output Layer:** Produces the final text sequence, completing the transformation.

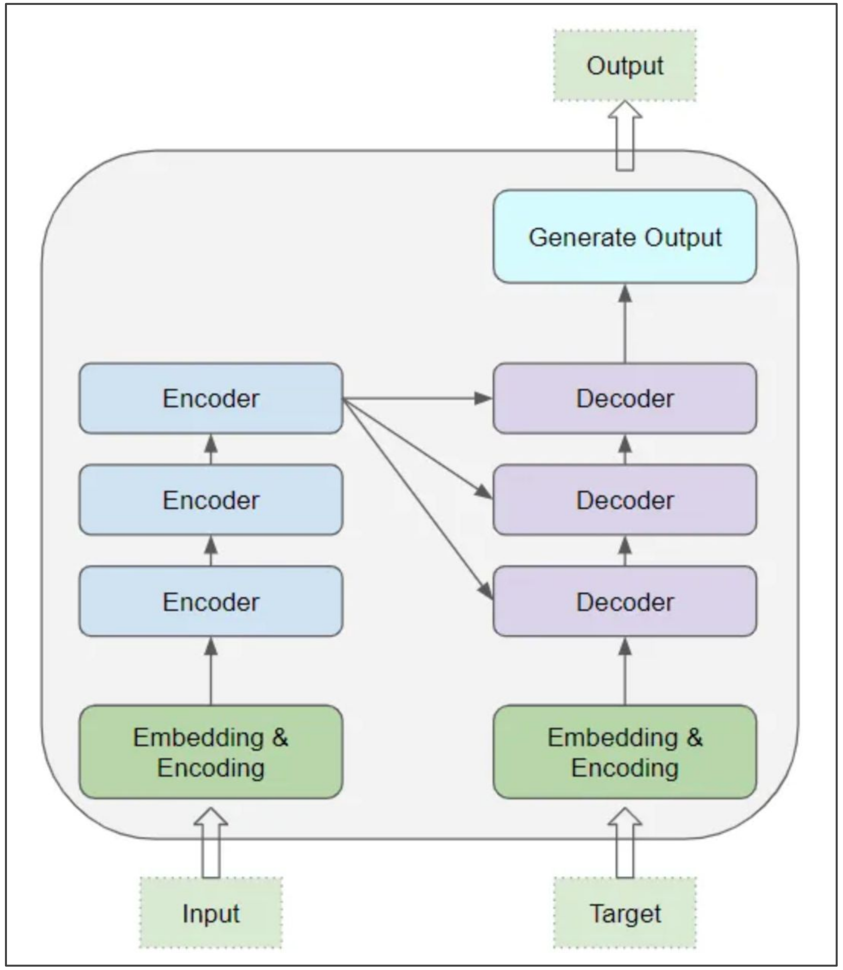


Figure . High level overview of transformer architecture

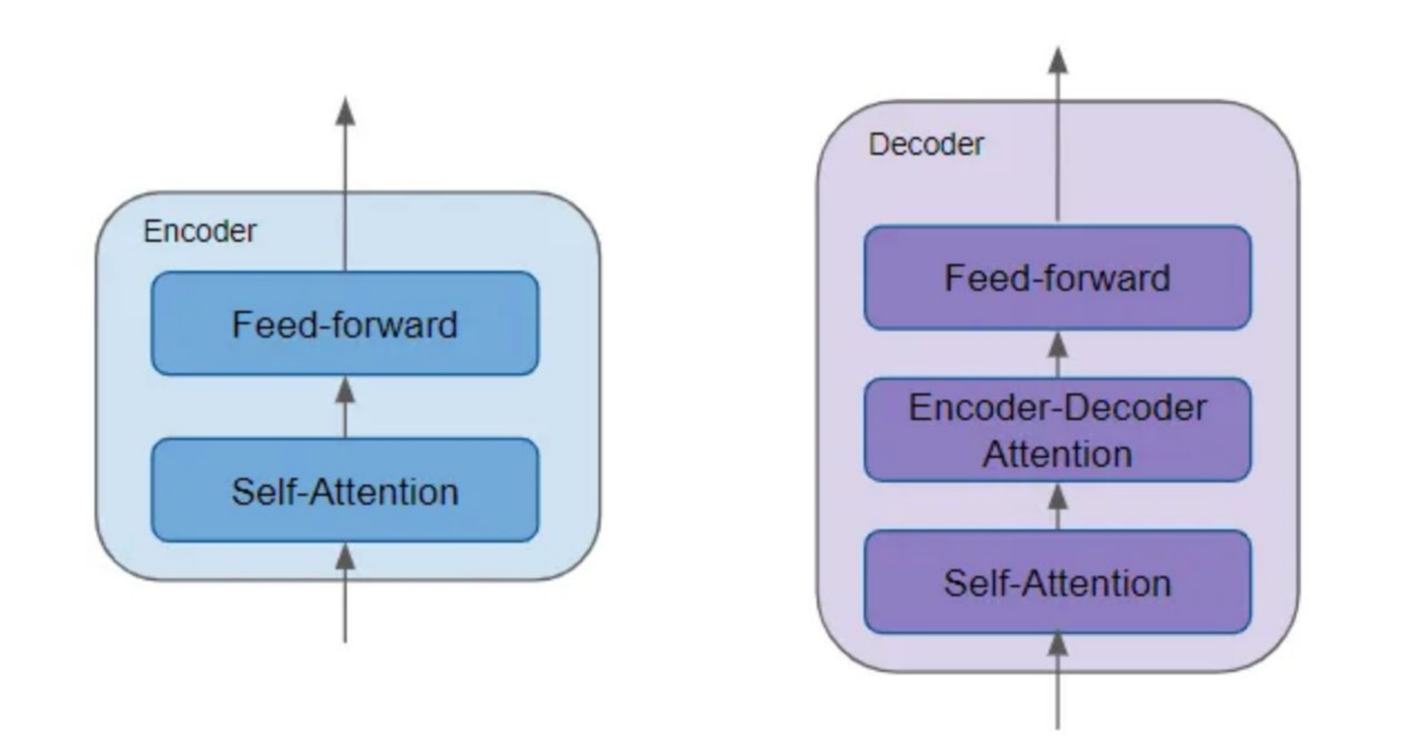
.. continued

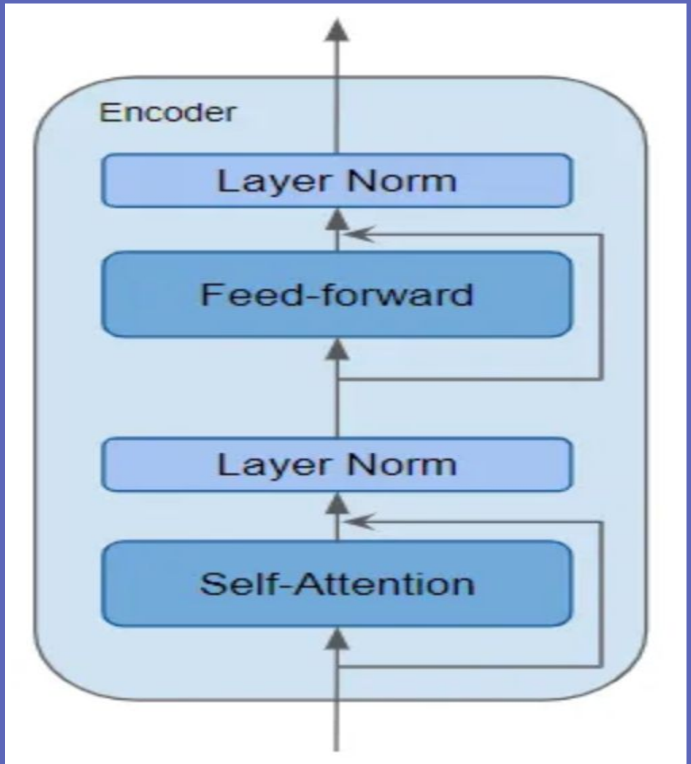
2.
3.
4.
5.

All the Encoders are identical to one another. Similarly, all the Decoders are identical.

- The Encoder contains the all-important **Self-attention layer** that computes the relationship between different words in the sequence, as well as a **Feed-forward layer**.
- The Decoder contains the **Self-attention layer** and the **Feedforward layer**, as well as a second **Encoder-Decoder attention layer**.
- Each Encoder and Decoder has its own set of weights.

- The Encoder is a reusable module that is the defining component of all Transformer architectures.
- In addition to the Self-attention and Feedforward layers , it also has **Residual skip connections** around both layers along with two **LayerNorm layers**.





1.

2.

3.

4.

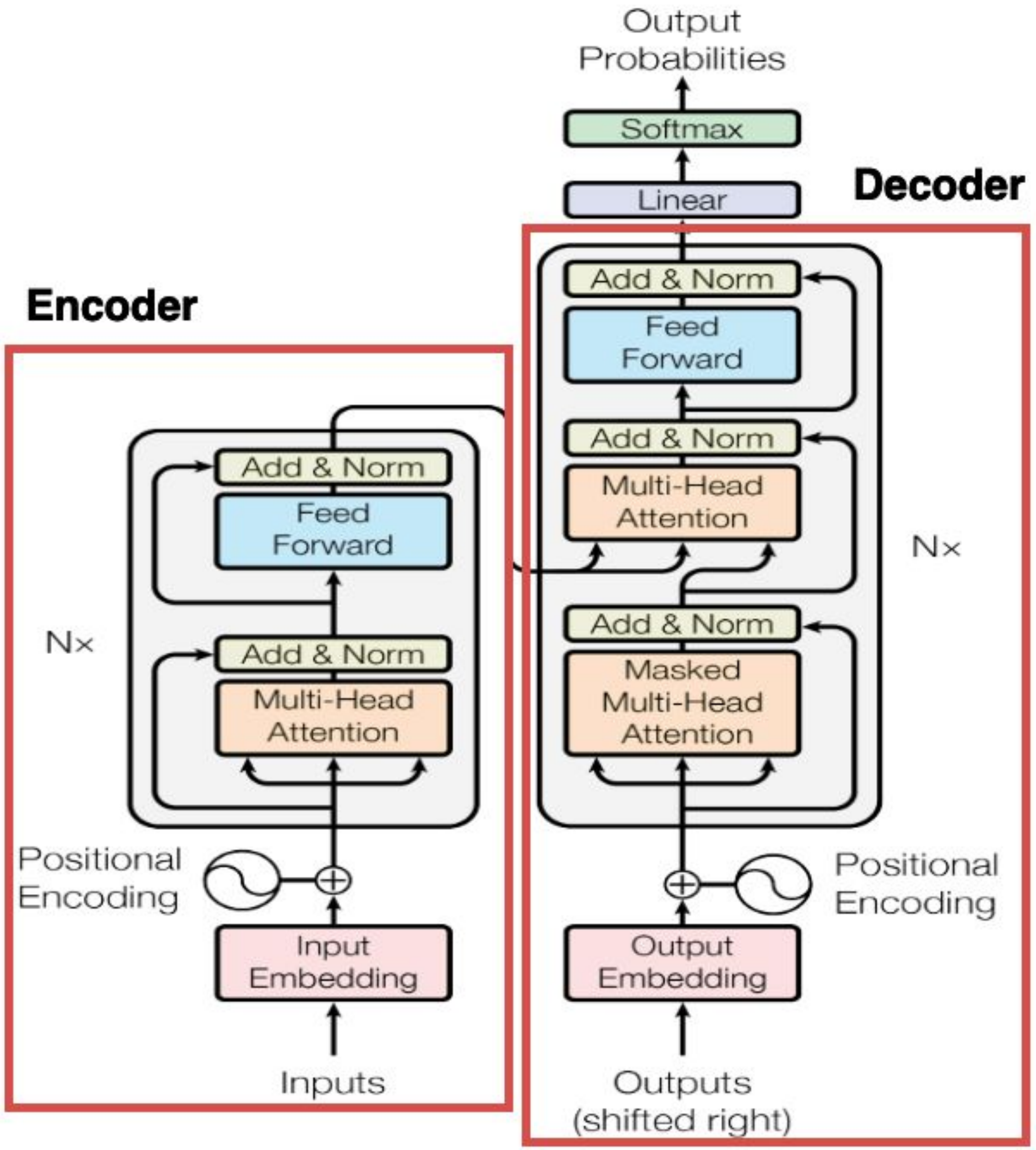
5.

In depth representation of the encode and decoder can be found here:

https://raw.githubusercontent.com/ajhalthor/Transformer-Neural-Network/main/Transformer_Architecture_complete.png

Example: English to Kannada translation

Hyper parameter:
- Batch size: 30 (passing in 30 sentences at once through the network)
- Max. number of words in a sentence: 50

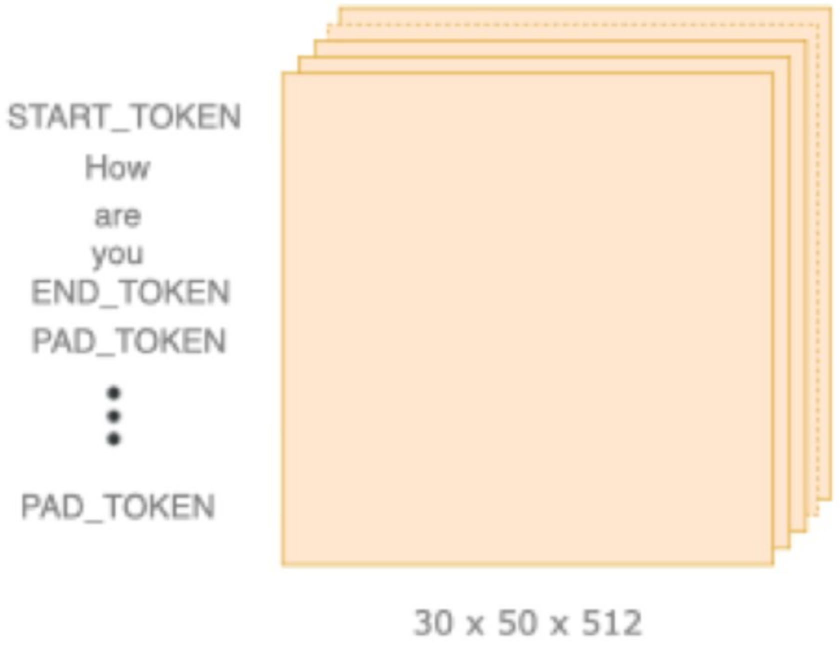


1.
2.
3.
4.
5.

Encoder - in depth

Input Embeddings - The Foundation of Translation

- **Purpose:** Converts words into numbers for neural network processing.
- **How It Works:** Each word becomes a 512-dimensional vector, capturing its meaning.
- **Batch Processing:** Processes 30 sentences simultaneously, with up to 50 words each.
- **Key Role:** Forms the base for contextual understanding in translation tasks.



1.
2.
3.
4.
5.

Encoder - in depth

Positional Encoding - Understanding Word Order

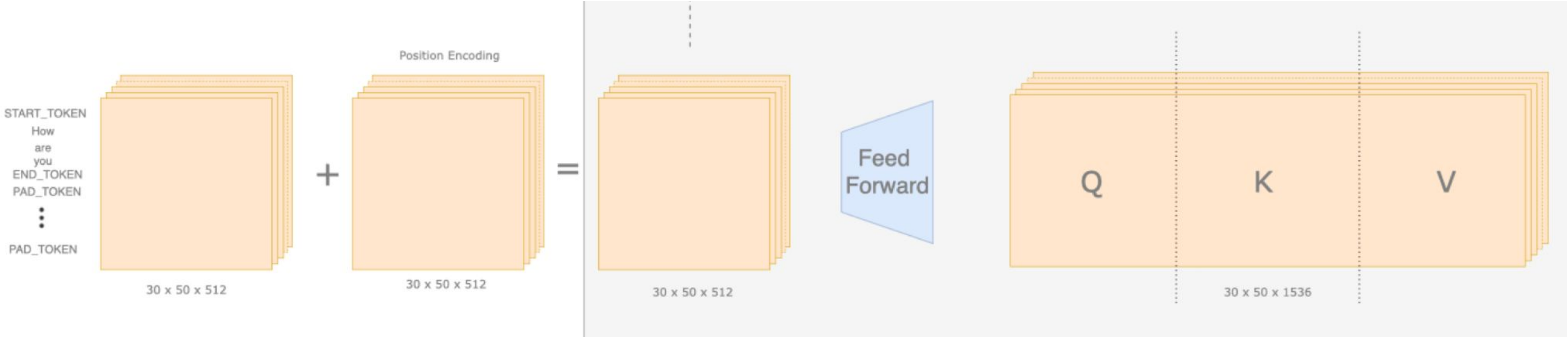
- **Why Needed:** Transformers process words in parallel, not in sequence.
- **Method:** Adds unique sine and cosine function values to word embeddings to indicate word positions.
- **Compatibility:** Matches embedding size ([30, 50, 512]), seamlessly integrates with input embeddings.
- **Impact:** Enables the model to grasp sentence structure and word context, critical for accurate translation.

Encoder - in depth

1.
2.
3.
4.
5.

After positional encoding we get a final tensor of shape 30x50x512

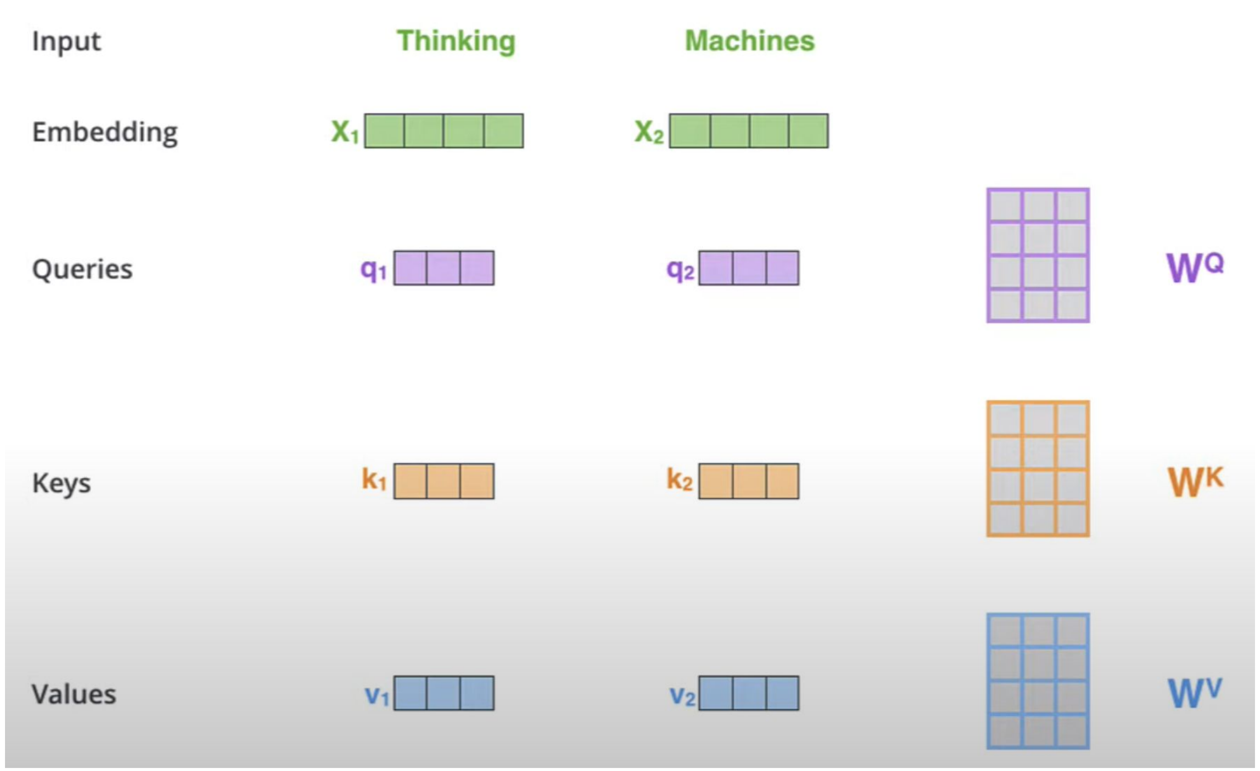
This final tensor is passed through a feedforward network in order to get Query (Q), Key (K) and Value (V) vectors.



1.
2.
3.
4.
5.

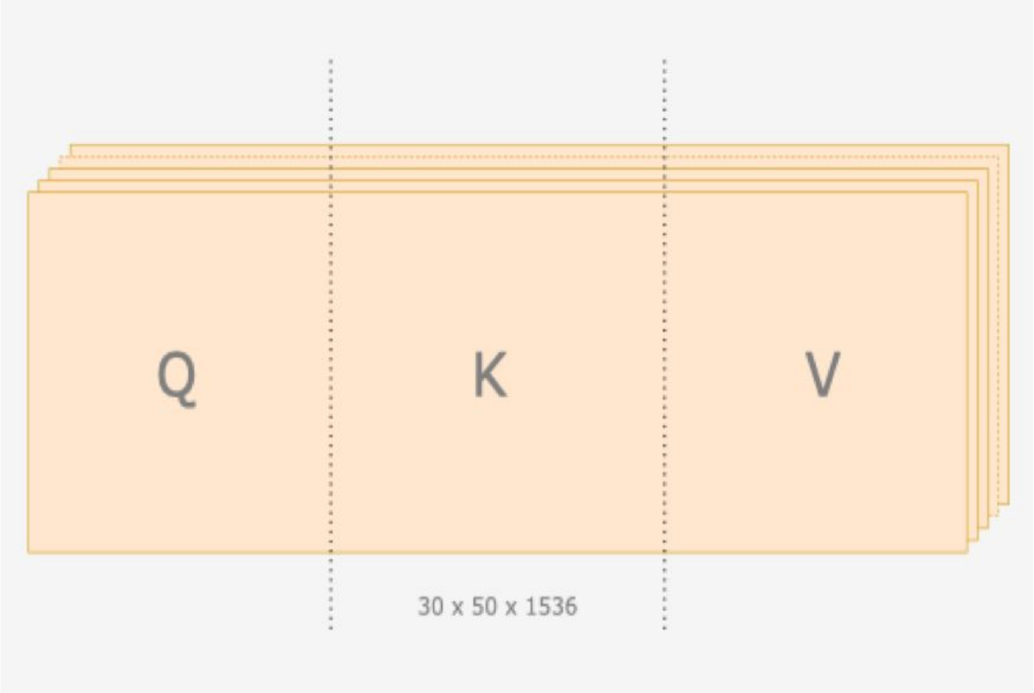
Encoder - in depth

Every word is split into 3 vectors - Query, Key and Value vectors. These vectors are used for different operations in later stages.



Therefore every 512 dimensional vector converted into a 512 time 3 that is 1536 dimensional vector.

Every 1536 dimensional vector now constitutes a word.



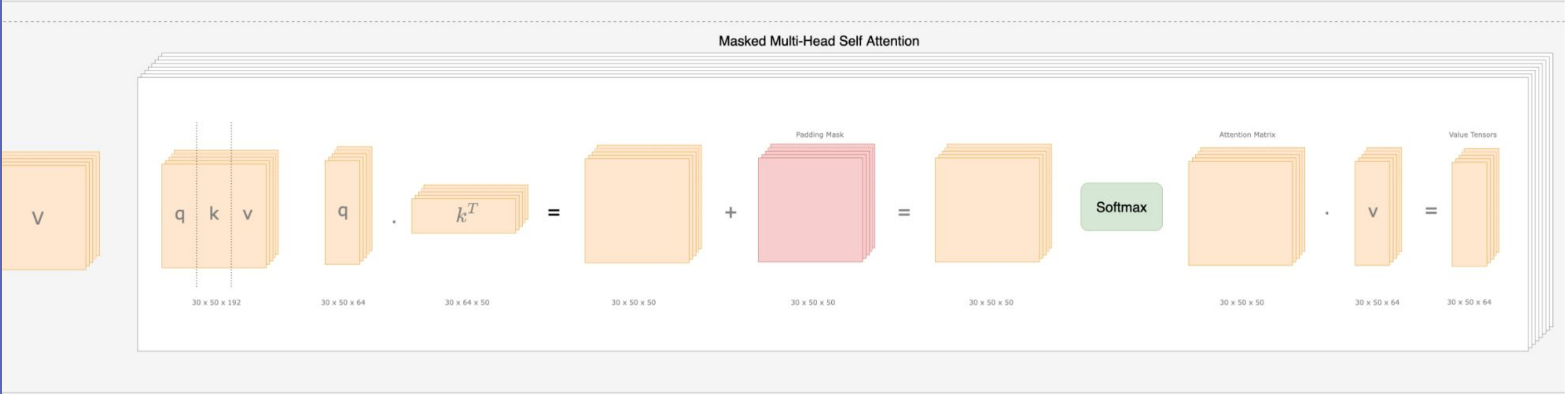
Encoder - in depth

Reason we split a word into 3 vectors is to perform **Multi-head attention**.

- **Total number of heads:** 8
- **Self attention:** Every word in the sentence is compared to every word in the same sentence - to analyze and build context.
- **30 x 50 x 192** - represents every word for 1 head

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

Scaling Value (8 in our case)



Encoder - in depth

Padding Mask

- Many sentences might have less that 50 words
- Padding tokens are added to standardize sentence lengths

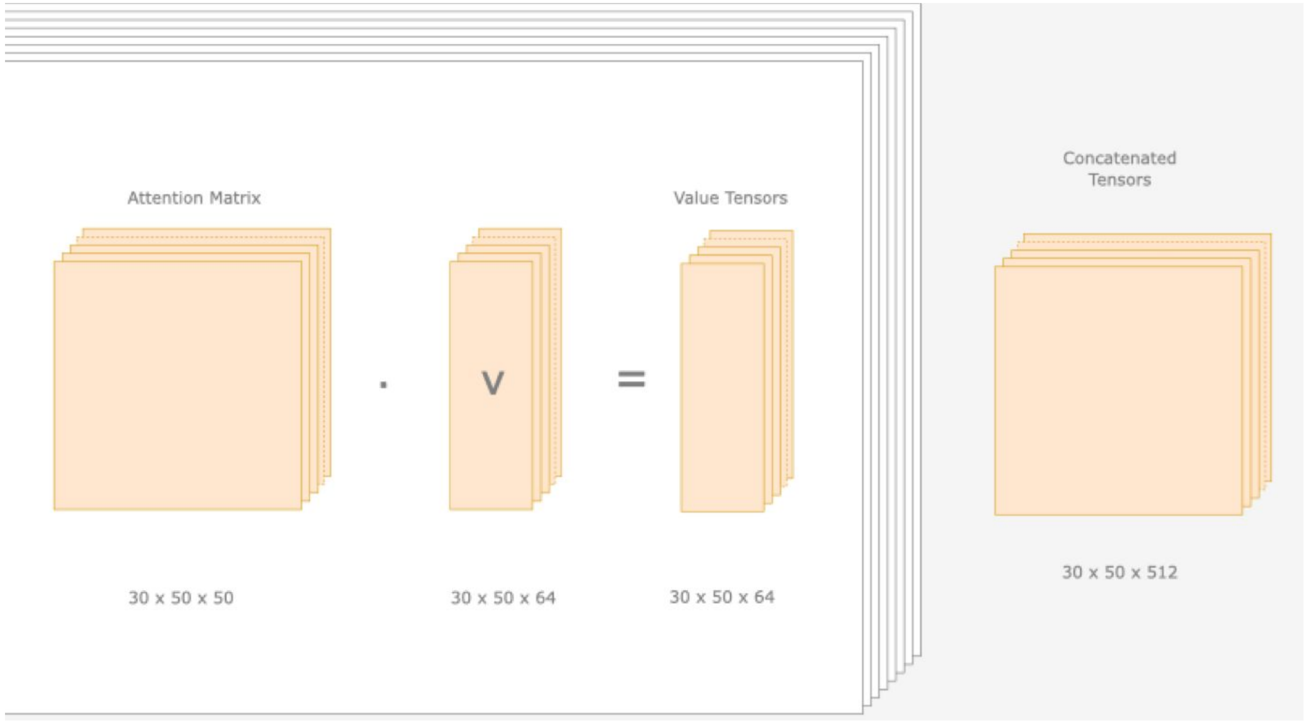
Attention Matrix

- 30 x 50 (number of words) x 50 (number of words) - probability distribution for every single row
- Each value in the matrix quantifies how much attention each word should pay to every other word

Encoder - in depth

Value tensors

- Product of **Attention Matrix (30x50x50)** and **Value vector(30x50x64) = Value tensor (30x50x64)**
- Contextually aware tensors
- **Value tensor (30x50x64) - output of 1 attention head**
- **Concatenate for 8 attention heads -** We get **30x50x512** dimensional vector



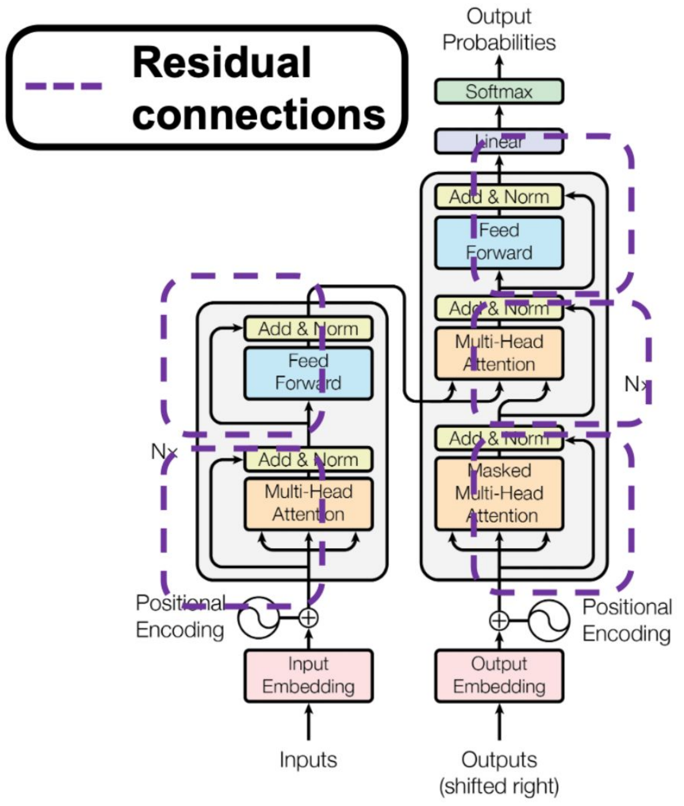
1.
2.
3.
4.
5.

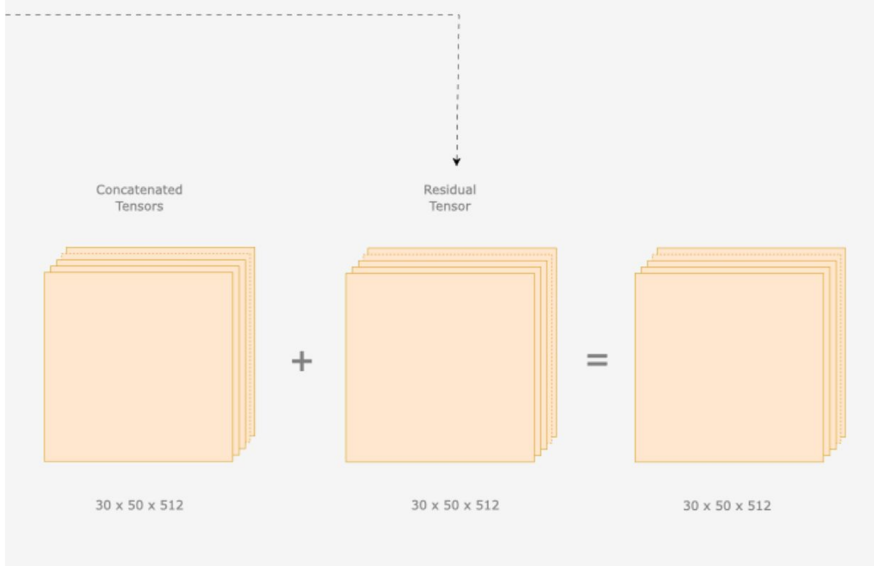
Encoder - in depth

Residual connection

- Loss values propagate backward to adjust weights.
- Loss diminishes as it propagates through layers.
- Deep networks risk loss not reaching all areas; parameters remain unchanged.
- Results in vanishing gradients; halts learning.

Skip Connection or Residual connection help by enabling better propagation of the **inputs in the forward direction** and the **loss in the backward direction**, facilitating learning.





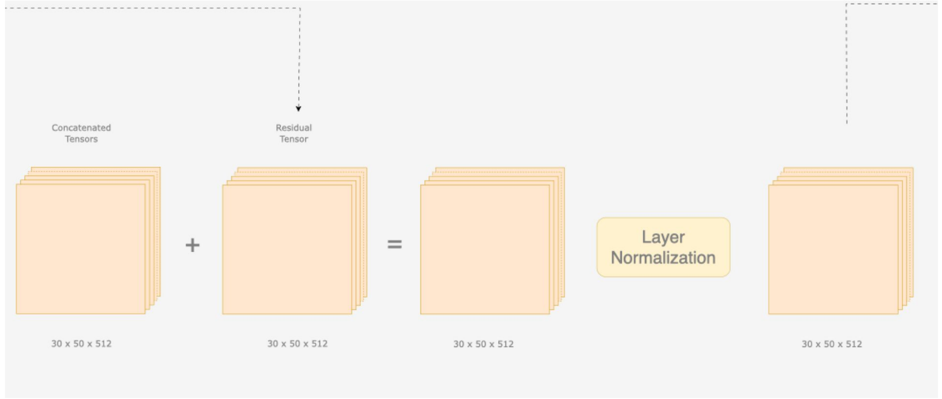
Encoder - in depth

1.
2.
3.
4.
5.

Layer Normalization:

- **Goal:** Stabilizes training by controlling activation magnitude during the forward phase and gradient updates during backpropagation.
- **Mathematical Process:** Normalizes by subtracting the mean and dividing by the standard deviation across features for each sample independently.
- **Application in Layer Norm:** Normalizes across the feature layer (e.g., 512 dimensions), adjusting each tensor value by the layer mean and standard deviation.
- Includes a small ε (epsilon) term with the standard deviation to avoid division by zero errors.

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

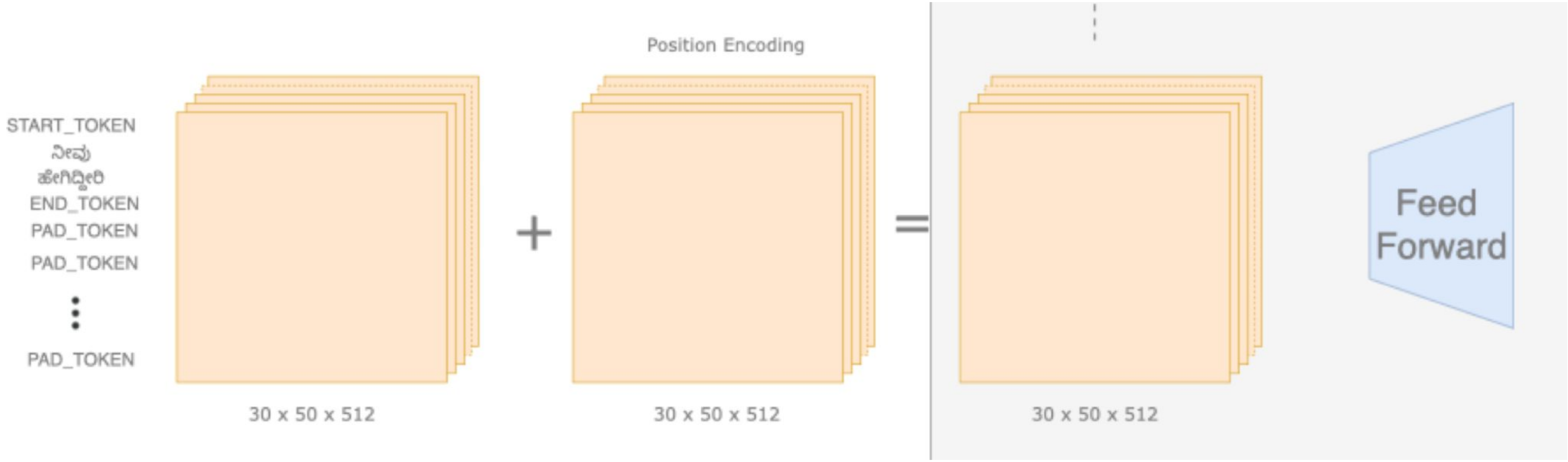


Decoder - in depth

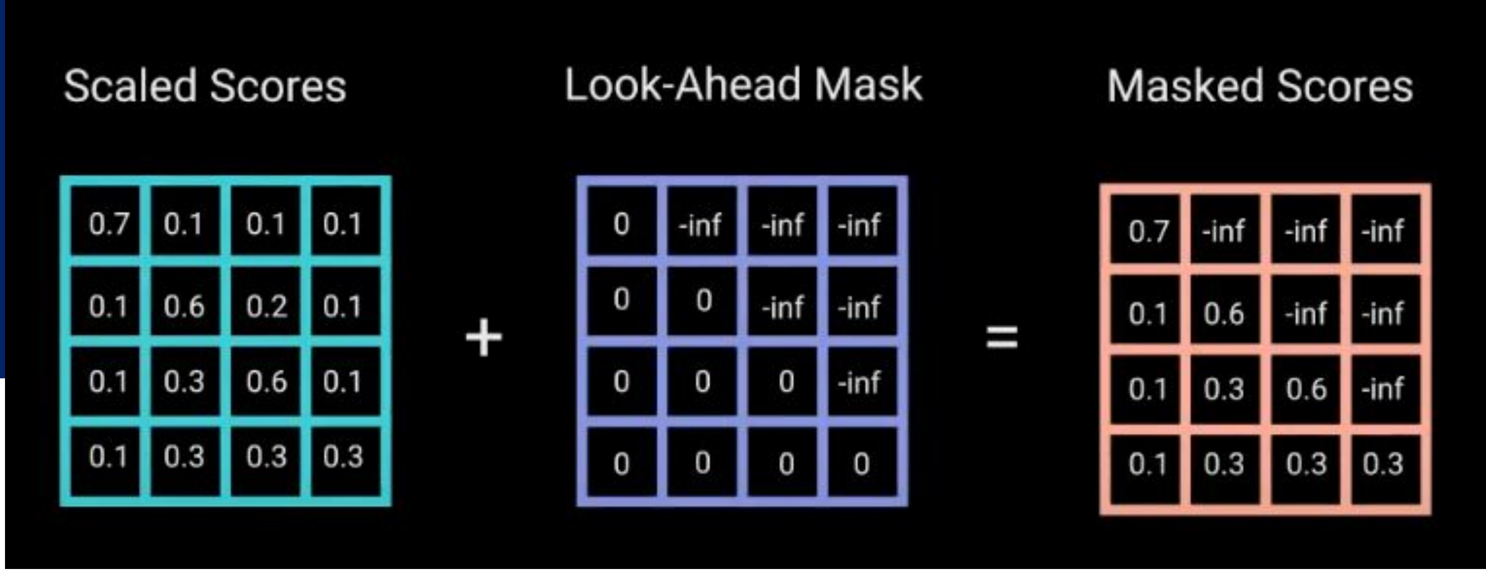
1.
2.
3.
4.
5.

Output Embeddings, Positional encoding, Feedforward -> Q, K, V -> Dot Product(Q, K^T)

- Initiate with tokens: Start_token, input sentence (target language), end_token, and padding up to 50 words.
- Apply positional encoding for word order.
- Generate Q, K, V vectors via a feedforward network.
- Perform Mass Multi-head Self-attention with Q, K, V.



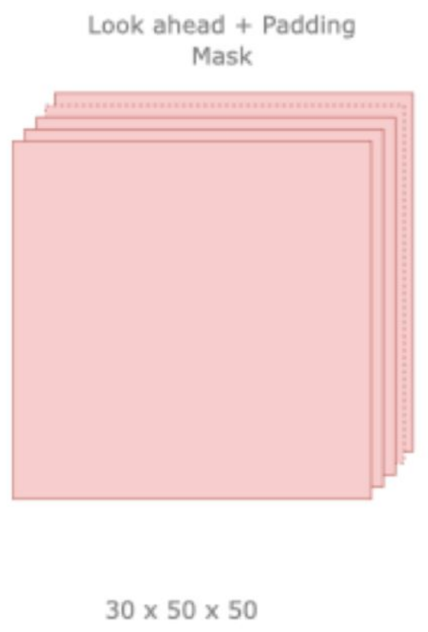
Decoder - in depth



1.
2.
3.
4.
5.

Padding + Look ahead Mask

- **Purpose of Look-Ahead Mask:** Prevents the decoder from "cheating" by accessing future target words during training.
- **Generation/Inference Phase:** During sentence translation, future words in the target language (Kannada) are not available.
- **Training Implementation:** A mask is applied to ensure predictions for a word only depend on previous words, not future ones.
- **Preventing Contextual Information Leakage:** Ensures a word cannot attend to future words for context during training.
- **Combination with Padding Mask:** Look-ahead mask is used alongside the padding mask to manage sequence lengths and prevent future look-ahead.



Decoder - in depth

1.
2.
3.
4.
5.

Multi-Head Cross Attention Overview
Utilizes a set of query vectors as input.
Focuses on cross-linguistic attention between target and source sentences.

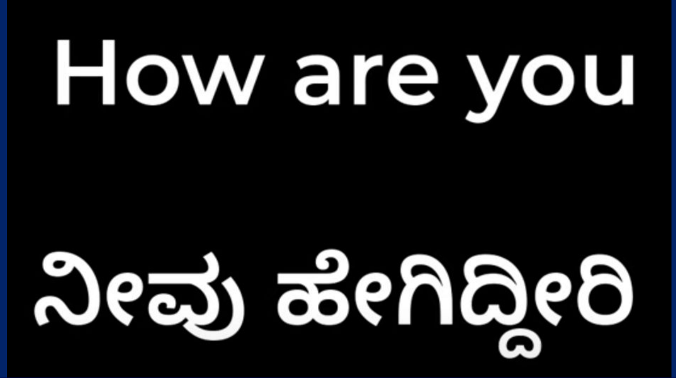
Key Components:
- Query: Represents target language words, guiding the focus.
- Key and Value: Vectors obtained from the Encoder, embedding source language (English) information into the target.

Functionality:
- Instead of self-attention within a single sentence, it cross-references every word in the target language (e.g., Kannada) with every source word in the English sentence.

Objective:
- To enhance target language vectors with encoded English information, facilitating more accurate translation or understanding.

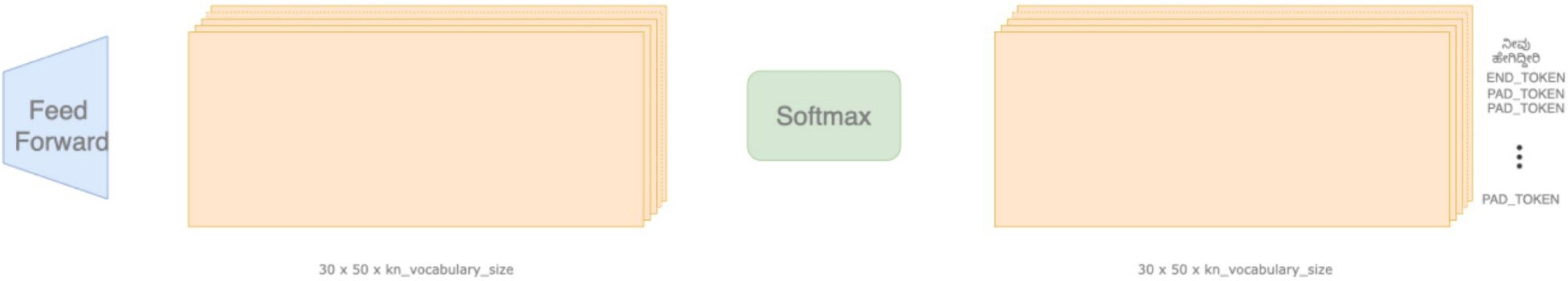
https://raw.githubusercontent.com/ajhalthor/Transformer-Neural-Network/main/Transformer_Architecture_complete.png

Decoder - in depth



1.
2.
3.
4.
5.

- **Feedforward Network Role:** Expands final tensor to match the size of the target language vocabulary.
- **Vocabulary Size Importance:** Determines the number of possible word predictions by the model.
- **Sentence Size vs. Vocabulary:** Sentences may be short, but vocabulary can be extensive.
- **Example Translation Process:** English (3 words) → Kannada (2 words).
- **Softmax Function Application:** Yields a probability distribution across all Kannada words; select highest probability word.
- **Prediction vs. Labels:** Output words "Neevu" and "Hegidiri" in Kannada compared against target labels.



1.
2.
3.
4.
5.

Transformer Training and Inference

- The training involves **passing input sequence (e.g., English sentence) and target sequence (e.g., Kannada translation) through the encoder and decoder, respectively**, and using the output of the decoder to calculate loss against the actual target sequence. This loss is used to update the model's parameters.
- Loss Calculation:

$$\text{Cross Entropy} = -\sum_i y_i \log(\hat{y}_i)$$

where y_i is the actual distribution (one-hot encoded vector), and $\hat{y}_i$ is the predicted probability distribution for all classes (words in the vocabulary).

- For translating a new sentence, the encoder processes the input sentence, and the decoder generates the output translation one word at a time, starting with a start token and ending when an end token is produced.

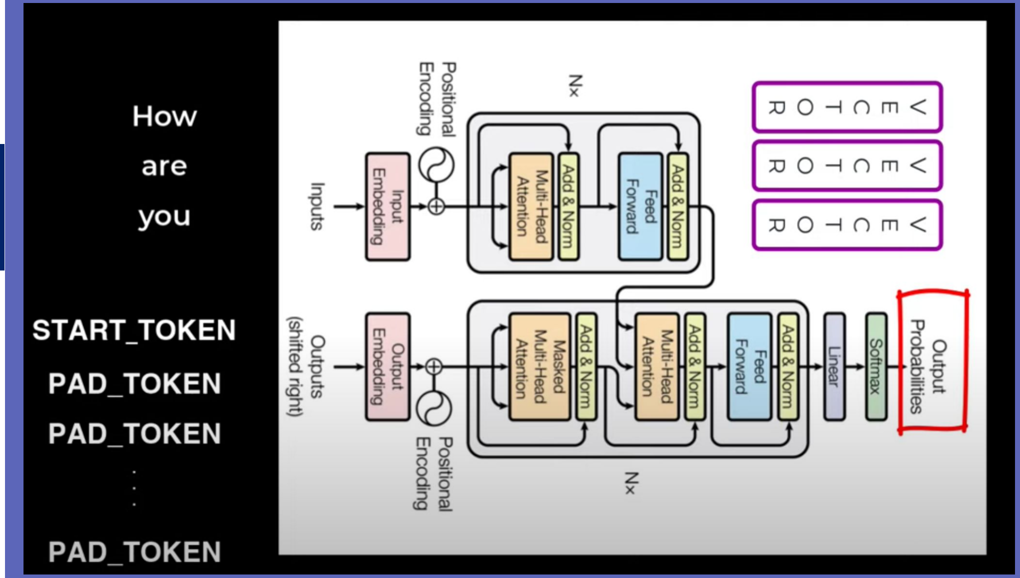
Transformer Training and Inference

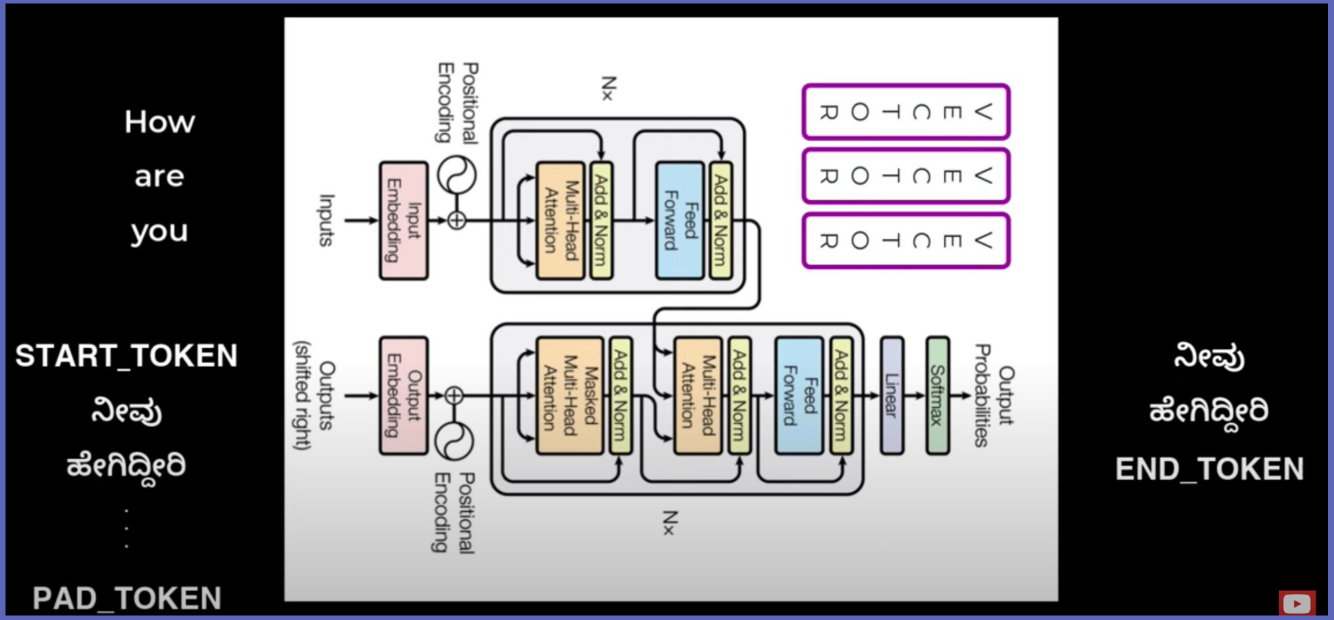
For translating a new sentence, the encoder processes the input sentence, and the decoder generates the output translation one word at a time, starting with a start token and ending when an end token is produced.

1.

2.

3.





Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Attention

Advantages

Highly parallel
Long Range dependencies

Challenges

Too much computation of the N x N attention matrix

Sparse Attention to reduce dense N x N attention matrix

Introduction

1 Attention Mechanism

2 Transformer Architecture

3 Sparse Transformer

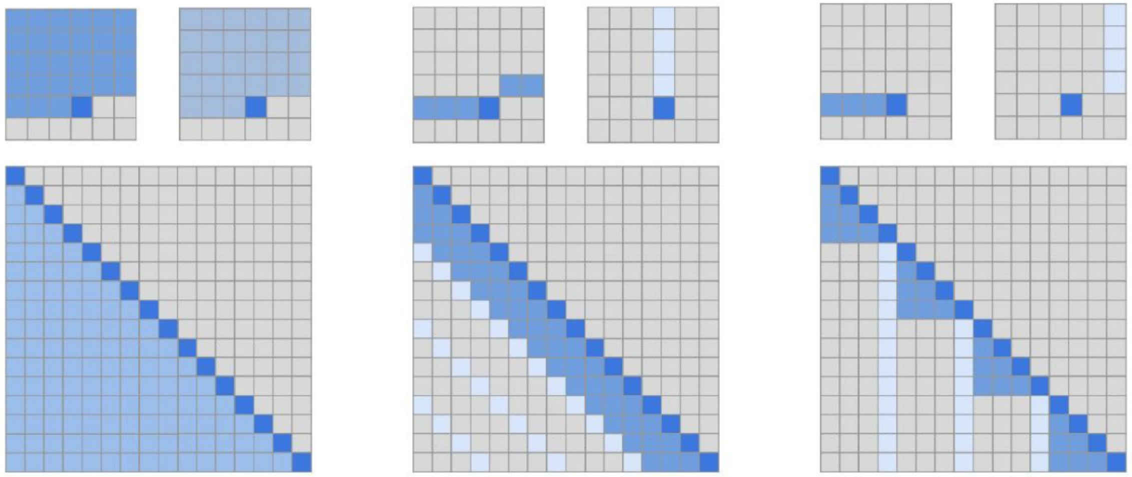
Sparse Transformer

Model architecture:

Sparse Transformers: A variation of the Transformer architecture that employs sparse factorizations of the attention matrix to reduce computational complexity from $O(n^2)$ to $O(n\sqrt{n})$.

Architecture Variants:

Two-dimensional factorized attention: Strided attention and fixed attention patterns to efficiently handle different data structures (e.g., images, text).



(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

Thank you!

Do you have any questions?

References and Helpful resources

[1]: https://en.wikipedia.org/wiki/Dot_product

[2]: https://en.wikipedia.org/wiki/Cosine_similarity

[3]: Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[4]: https://www.youtube.com/watch?v=Nw_PJdmydZY

[5]: https://www.youtube.com/watch?v=OxCpWwDCDFQ

[6]: https://www.youtube.com/watch?v=Nw_PJdmydZY

[7]: https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-science-pdf-f22dc900d2d7