Generative AI

By **Prudvi**

Generative AI

- Generative AI refers to a subset of artificial intelligence that focuses on creating new content, such as text, images, audio, video, or even code.
- Unlike traditional AI systems that are designed for classification, prediction, or decision-making, generative AI produces data that mimics human-created content.
- It leverages machine learning models, particularly **deep learning techniques**, to understand patterns in existing data and generate new outputs based on those patterns.

Applications of Generative AI

Text Generation: Writing essays, code, or summarizing documents (e.g., ChatGPT, Bard).

Image Synthesis: Creating realistic images (e.g., DALL-E, MidJourney).

Video Generation: Creating deepfake videos or simulations.

Music and Audio Synthesis: Composing music or generating voiceovers.

Personalization: Generating custom content for e-commerce or marketing.

Scientific Research: Generating molecular structures or simulating environments.

Evolution of Generative AI

1. Early Beginnings (1950s–2000s)

1950s–1980s: Focused on symbolic AI and rule-based systems. Early neural networks (e.g., Perceptrons) laid the foundation but had limited capabilities.

1990s–2000s: Statistical methods like HMMs(hidden Markov Model) and GMMs(Gaussian Mixture Model) enabled simple generative tasks (e.g., speech synthesis). LDA was introduced for topic modeling.

2. Neural Network Revolution (2010s)

Key Models:

- CNNs revolutionized image processing.
- RNNs enabled sequence generation for language modeling.
- GANs (2014) by Ian Goodfellow introduced realistic image generation.

Advancements:

- Word embeddings (e.g., Word2Vec, GloVe) improved text generation.

Applications: Used in video games, media, and medical imaging.

3. Transformer Era and LLMs (2017– till today)

2017: Google's "Attention Is All You Need" introduced transformers, revolutionizing NLP and generative AI.

GPT Evolution:

GPT-1 (2018): First transformer-based generative model.

GPT-2 (2019): Generated high-quality, human-like text.

GPT-3 (2020): Set new benchmarks with 175 billion parameters.

Major Breakthroughs:

DALL-E (2021): Combined text and image generation.

Stable Diffusion (2022): Advanced image synthesis.

ChatGPT (2022): Made generative AI accessible to all.

4. Modern Advancements (2020s)

Multimodal AI: Models like GPT-4 combine text, images, and other data types.

Fine-Tuning: LoRA and adapters enable cost-efficient, domain-specific solutions.

Optimization: Techniques like quantization and distillation make models production-ready

Evolution of Generative AI

Early Probabilistic Models:

Restricted Boltzmann Machines (RBMs): Introduced probabilistic graphical models capable of learning distributions over input data, facilitating tasks like unsupervised learning and collaborative filtering.

Advances in Sequential Modelling:

Hidden Markov Models (HMMs) and Markov Chain Monte Carlo (MCMC) methods: Used for sequential data modelling and Bayesian inference, these techniques laid foundational principles for generative modelling.

Neural Network Innovations:

Autoencoders: Early neural networks designed for learning efficient representations of data through encoder-decoder architectures. Variants like **variational autoencoders (VAEs)** extended these capabilities to probabilistic generative tasks.

Game-Changing Generative Adversarial Networks (GANs):

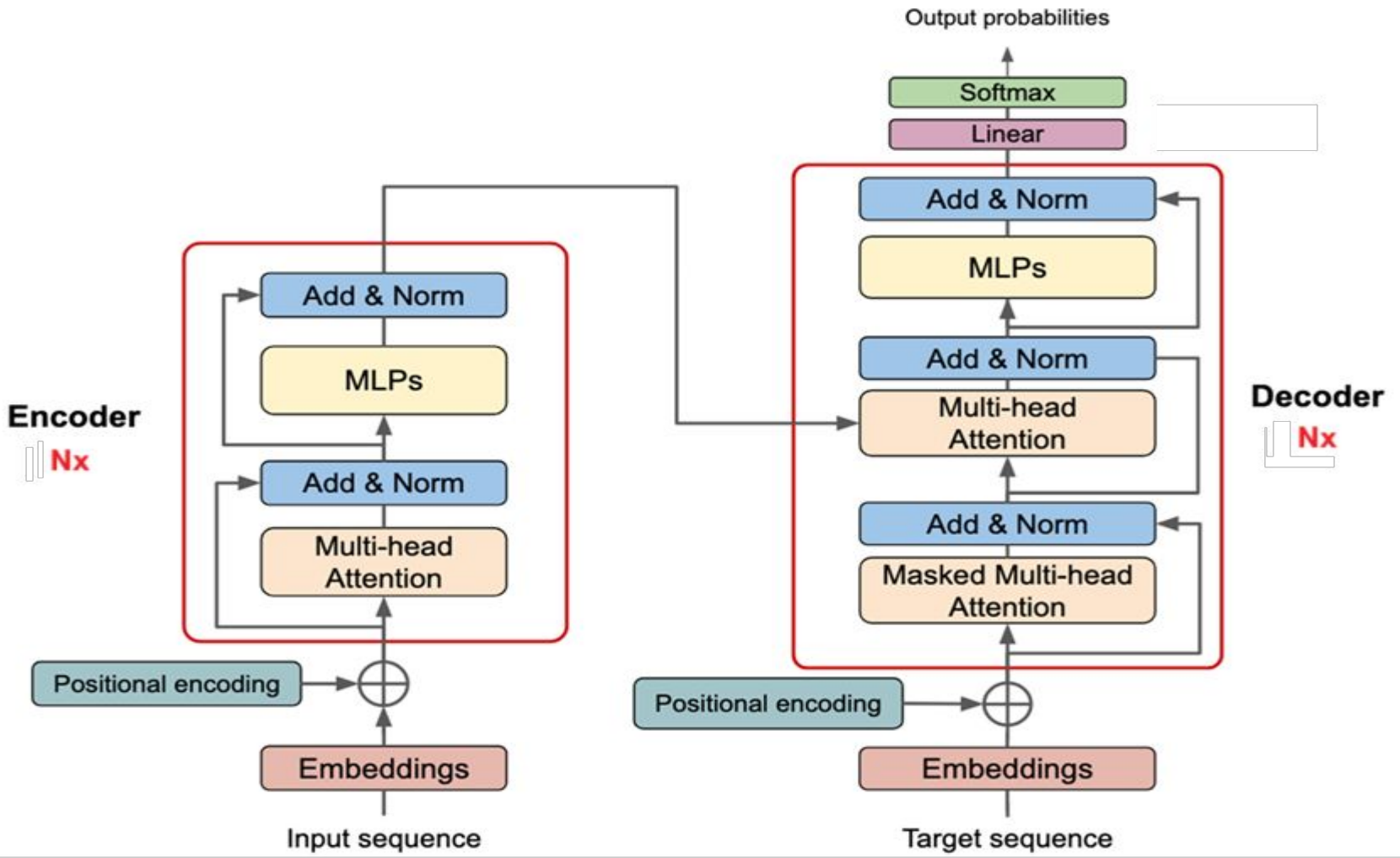
Introduced by Ian Goodfellow in 2014, **GANs** revolutionized generative modelling by employing adversarial training between a generator and a discriminator network. This approach enabled the generation of realistic images, audio, and text data, pushing the boundaries of AI creativity.

Transformers and LLMs have transformed AI:

Transformer Architecture: Created for natural language processing, transformers introduced self-attention mechanisms that greatly improved tasks like sequence-to-sequence operations and text generation. Recent innovations like **BERT**,**GPT**, **LLaMA**, **Mistral**, **Falcon and DALL·E etc**. have continued to revolutionize Generative AI.

Transformer Architecture

RNN : Maintains a hidden state to remember past information.
Transformer : Self Attention : Uses attention weights to "remember" relevant tokens.



Self Attention – Attention weights

Self-Attention (Within-Sequence):
Self-attention operates within a single sequence — each token attends to all tokens (including itself) in the same sequence.
Input Sequence (e.g., "The cat sat") -> Self-Attention -> Contextualized Representations of each token
i.e. The Vs The; The vs Cat ; The Vs sat ; cat Vs sat

Example: Money grows in bank
$money_{emb} = 0.9\ money_{emb} + 0.5\ bank_{emb} + 0.7\ grows_{emb}$
$bank_{emb} = 0.7\ money_{emb} + 0.95\ bank_{emb} + 0.1\ grows_{emb}$
$grows_{emb} = 0.8\ money_{emb} + 0.4\ bank_{emb} + 0.89\ grows_{emb}$

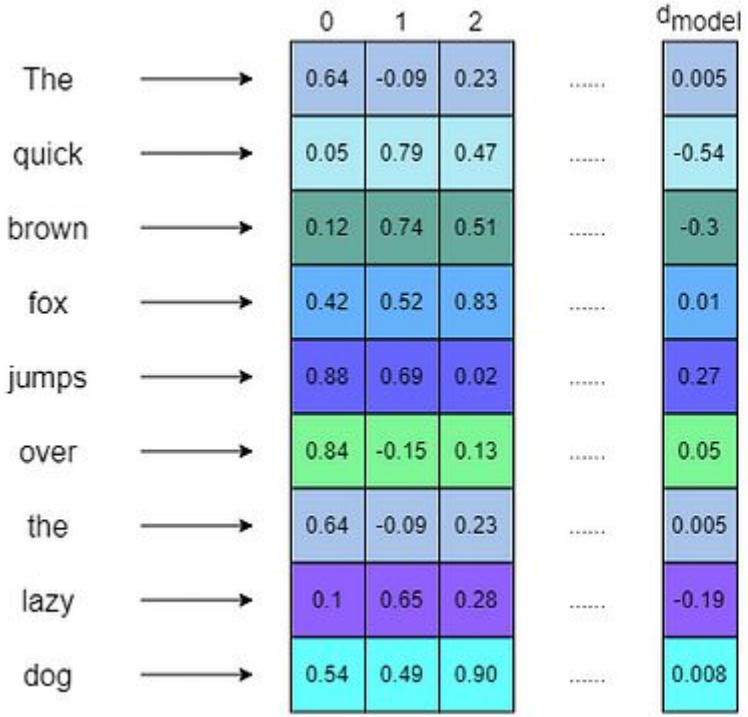
Example with Numbers
Let's assume the static embeddings are 3-dimensional for simplicity:

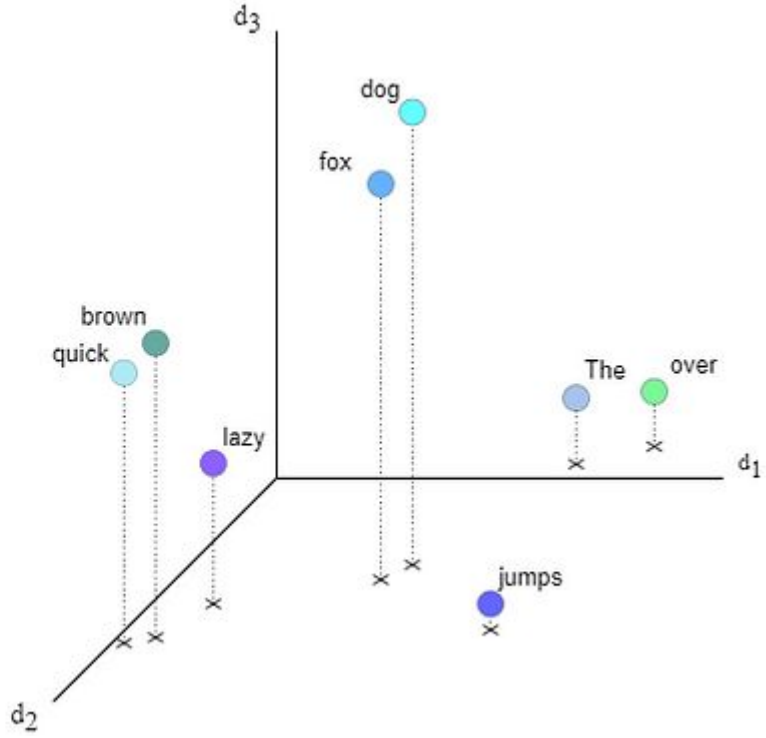
- money_emb = **[1.0, 0.8, 0.3]**
- bank_emb = [0.5, 1.2, 0.7]
- grows_emb = [0.2, 0.1, 0.9]

Suppose below are **attention weights** for "money": i.e. **dot product** of money Vs money and money Vs bank and money Vs grows
- money_emb: 0.9
- bank_emb: 0.5
- grows_emb: 0.1

Compute the contextual embedding:
money_context_emb = $(0.9 \cdot [1.0,0.8,0.3])+(0.5 \cdot [0.5,1.2,0.7])+(0.1 \cdot [0.2,0.1,0.9])$
= [0.9,0.72,0.27]+[0.25,0.6,0.35]+[0.02,0.01,0.09]
= [1.17,1.33,0.71]
Thus, the **contextual embedding for "money"** is **[1.17, 1.33, 0.71]**

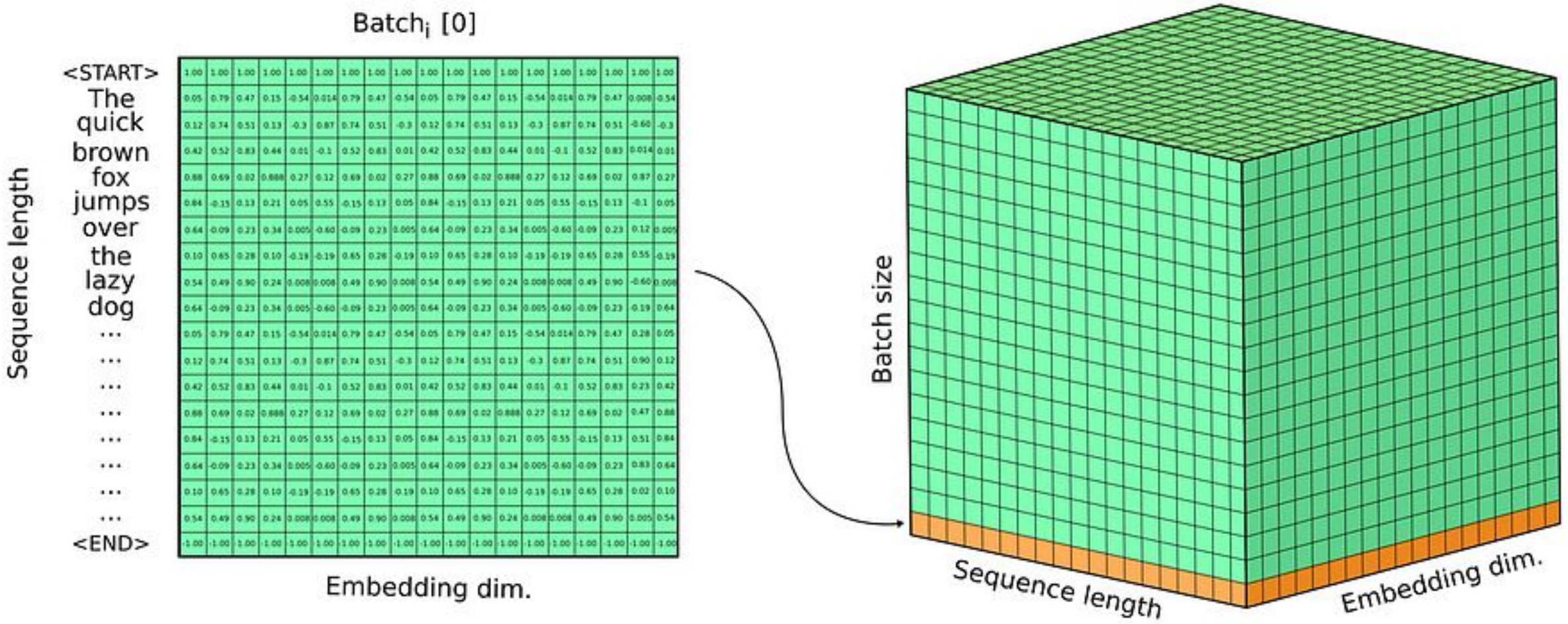
Encoder - Input embedding



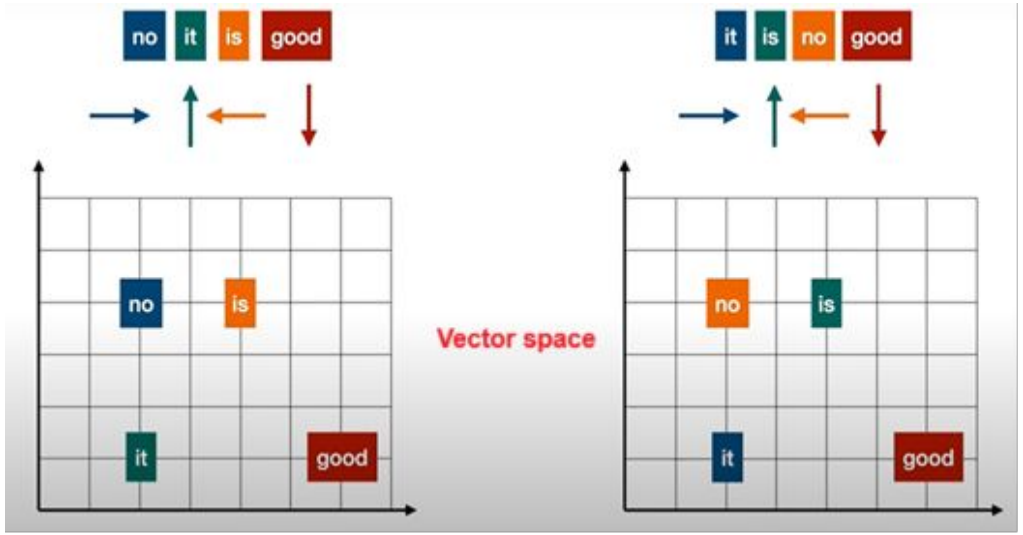


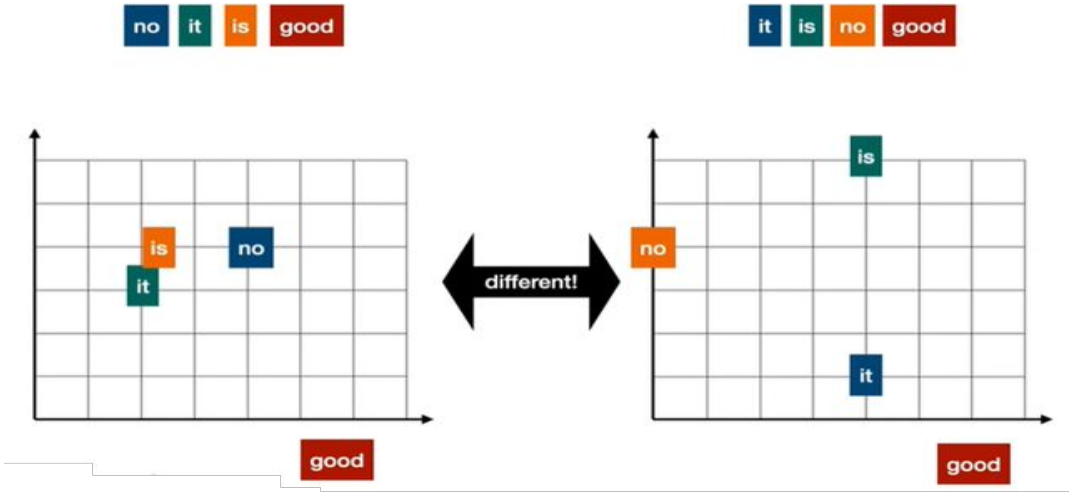
Transformer Architecture

Input is a tensor of shape *[N, L, E]* where *N* is the batch size, *L* the sequence length, and *E* the embedding dimension.



Positional Embedding





As the words/tokens are represented in vector space, they will be same unless we add the position of the token in the sequence.

So when you add the position they will become different, which will be captured by the NN while learning.

Here is *why absolute position alone is NOT enough*, and why Transformers prefer **relative** position

Sentence1: The cat chased the mouse
Sentence2: Yesterday, the cat chased the mouse.

Even though the **meaning and structure are the same**, the absolute positions are completely different.
But the **relative relationship between "cat" and "chased"** stays the same

Sentence1: I saw her duck.
Sentence2: I saw her, duck!
Relative: *her* is now far from *duck*, separated by a comma

Positional Embedding

Solution1: take position and add to embedding

0.60	0.71	0.43	0.32	1

Token at pos 1

0.90	0.11	0.01	0.22	2

Token at pos 2

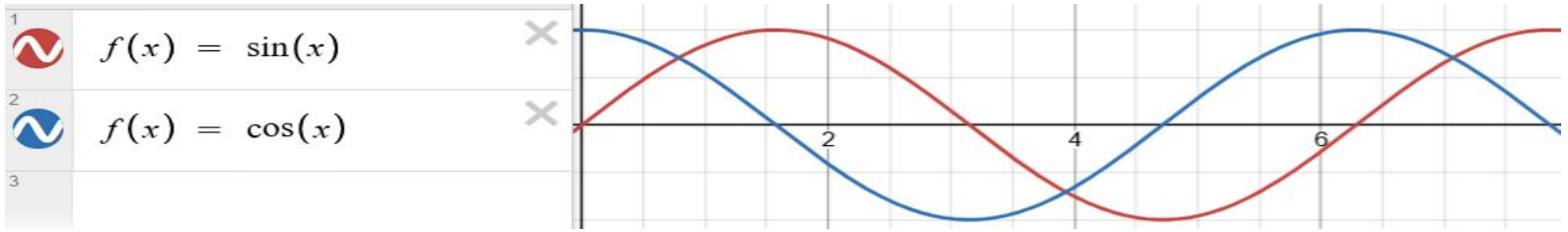
No bounding it can go max to any number with number of tokens, it is always better to keep in bound as this embedding's will be given to Neural network for learning.

Solution2: Normalize so the values will be bounded, but if different length of sentences will have different ranges
Ex: if 4 tokens, so 1/4,2/4,3/4,4/4 and if 5 tokens then 1/5,2/5.3/5,4/5,5/5

Key issue: Relative position than absolute position, the solution is using geometric functions such as sine and cosine.
With basic solutions, there is **no relative position** assigned means the distance between the tokens can be rightly measured as with absolute positions there could be continuous values can appear. Ex: between 1 and 2 there are so many other values as 1.01,1.001.1.3,1.45 and so on…

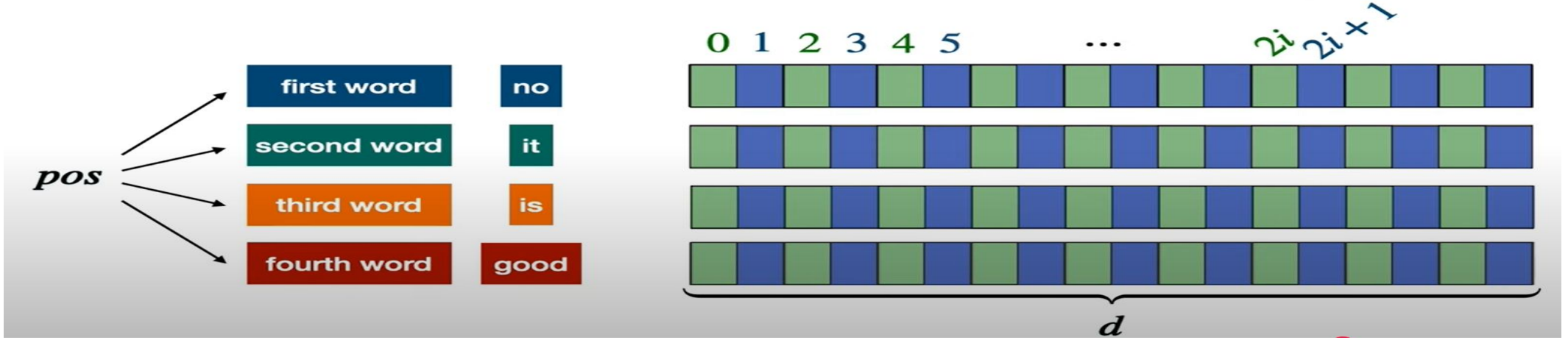
Example of Relative Position
Consider the sentence: **"I deposited money in the bank yesterday."**
Here, the word **"money"** is in position 3, and **"bank"** is in position 6.
The **absolute positions** of the words are 3 and 6.
The **relative position** of **"money"** to **"bank"** is: Relative Position=Position of Bank−Position of Money=6−3=+3
This relative position (+3) indicates that **"bank"** is three tokens after **"money."**

Positional Embedding



sin has boundary of -1 to 1 and periodically repeat the sequence, but with only sin wave the position info might get repeated so used cosine also at each position. Ex: f(x) = sin(1) and f(x) = cos(1) and concatenated to embedding, still as the position 1 and may be 110th position have a chance to repeat same numbers for longer sequences then researchers used below.

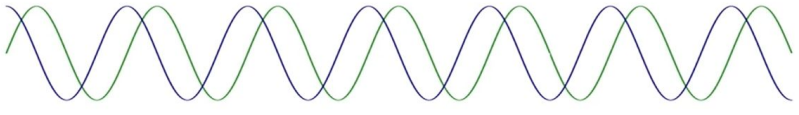
$$PE(pos,2i) = \cos\left(\frac{pos}{10000^{2i/d}}\right) \qquad PE(pos,2i+1) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$



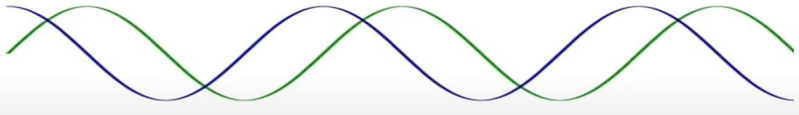
Positional Embedding

$$PE(pos,2i) = \cos\left(\frac{pos}{10000^{2i/d}}\right) \qquad PE(pos,2i+1) = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

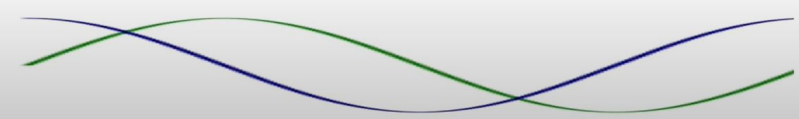
$i = 0$ $\quad \cos(pos)$, $\sin(pos)$



$i = 1$ $\quad \cos\left(\frac{pos}{10000^{2/d}}\right)$, $\sin\left(\frac{pos}{10000^{2/d}}\right)$

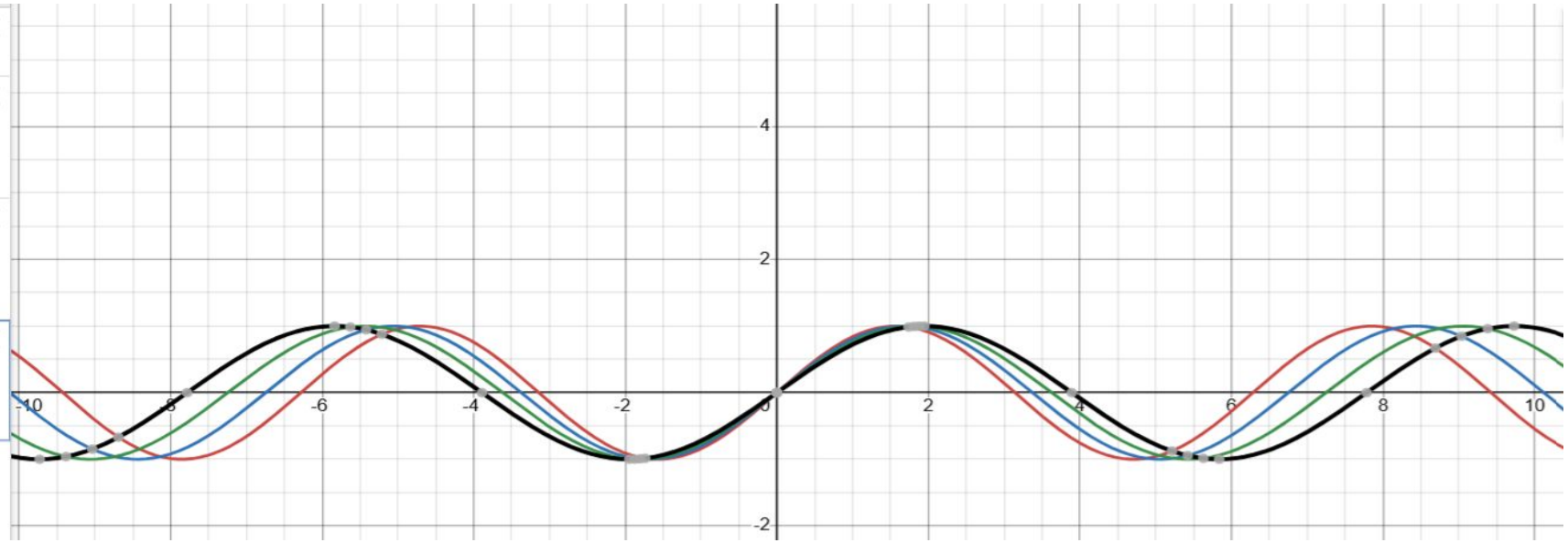


$i = 2$ $\quad \cos\left(\frac{pos}{10000^{4/d}}\right)$, $\sin\left(\frac{pos}{10000^{4/d}}\right)$

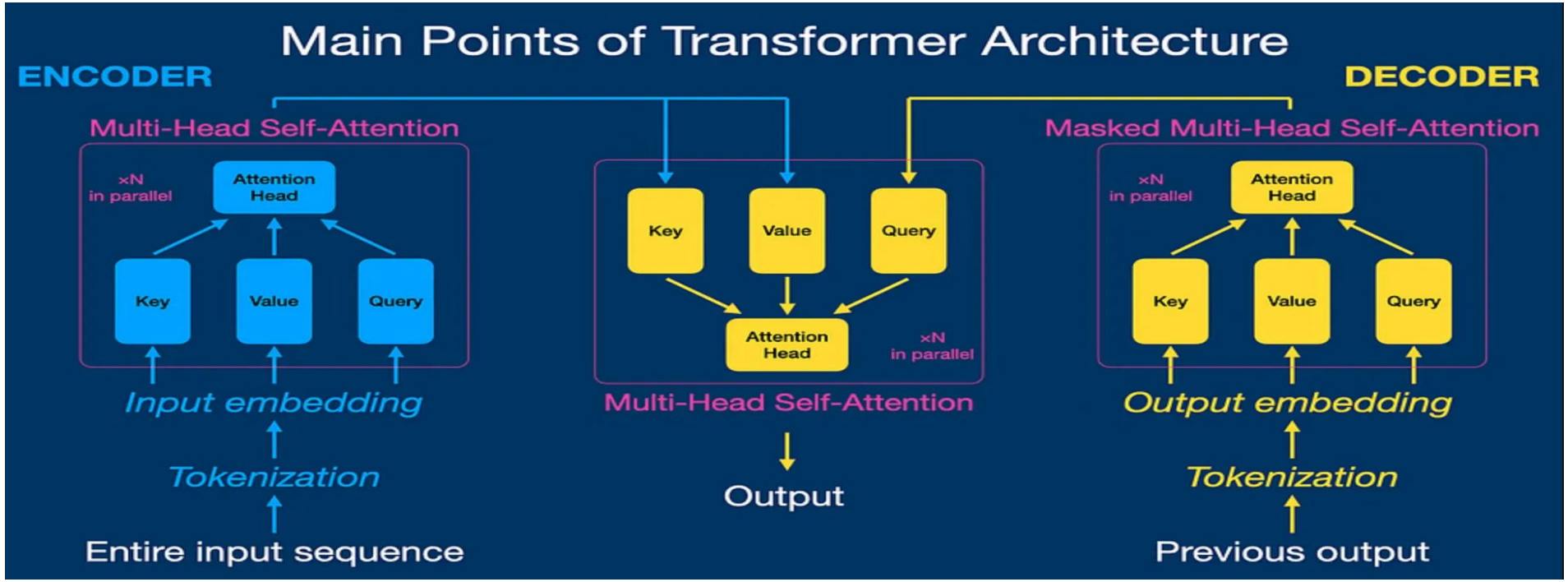


$\vdots$

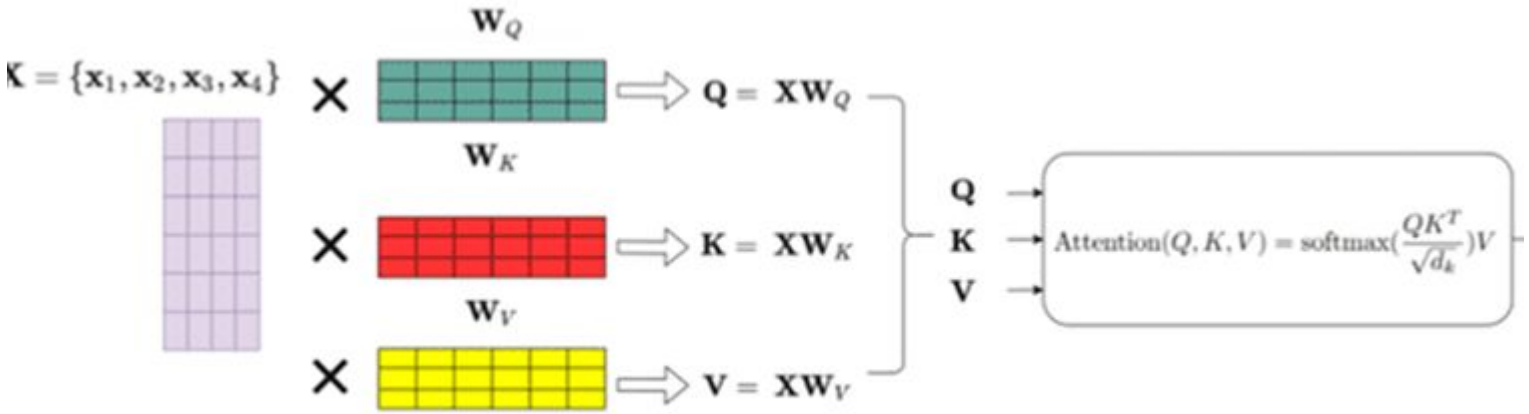
1. $f(x) = \sin(x)$
2. $f(x) = \sin\left(\frac{x}{(10000)^{\left(\frac{4}{(516)}\right)}}\right)$
3. $f(x) = \sin\left(\frac{x}{(10000)^{\left(\frac{8}{(516)}\right)}}\right)$
4. $f(x) = \sin\left(\frac{x}{(10000)^{\left(\frac{12}{(516)}\right)}}\right)$
5.



Transformer Architecture – Q, K & V

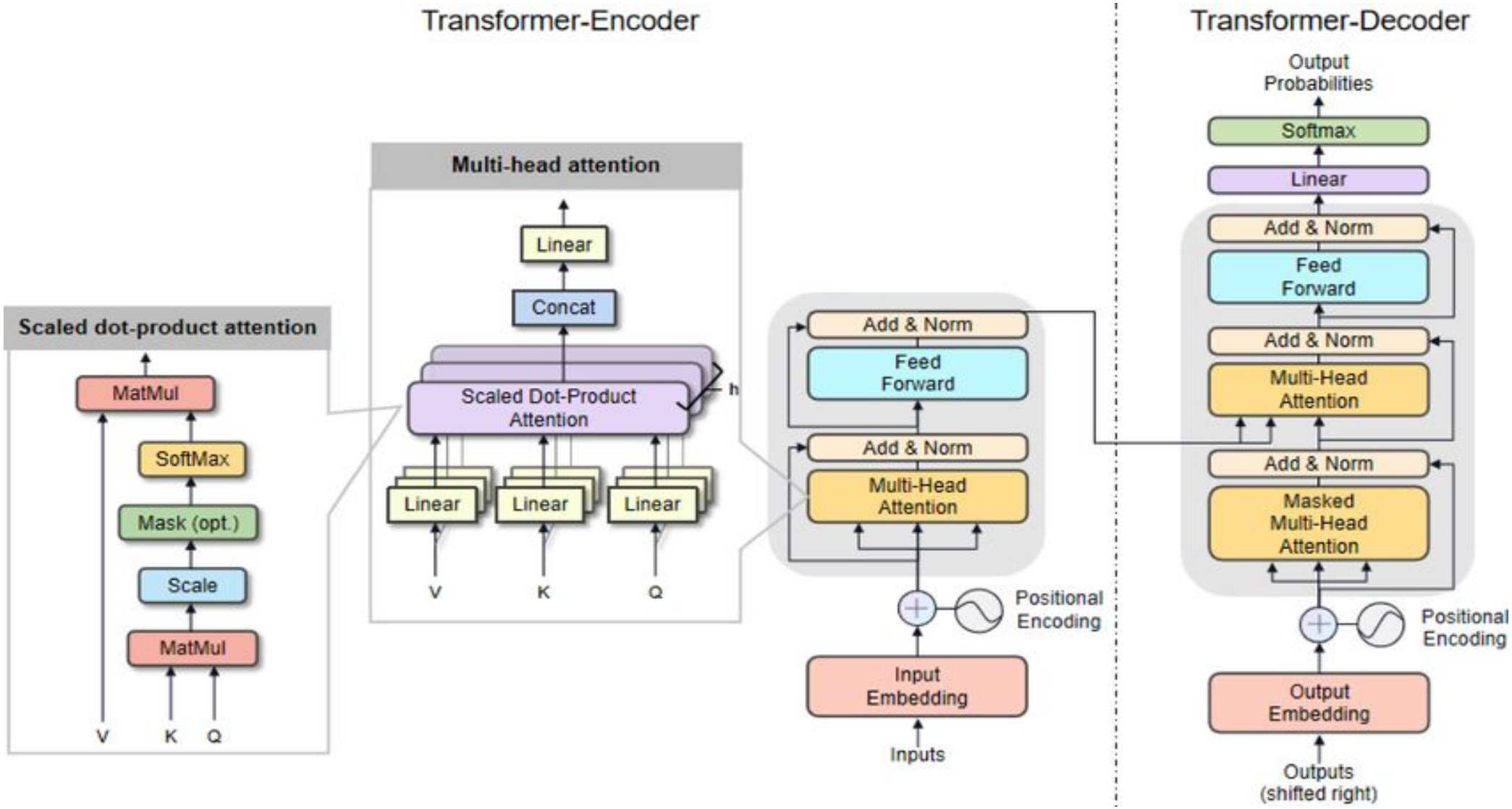


Q - Query
K - Key
V - Value



Each token embedding(x) will be considered for Q, K ,& V after a linear transformation which are learnable parameters

Transformer Architecture – Multi head attention



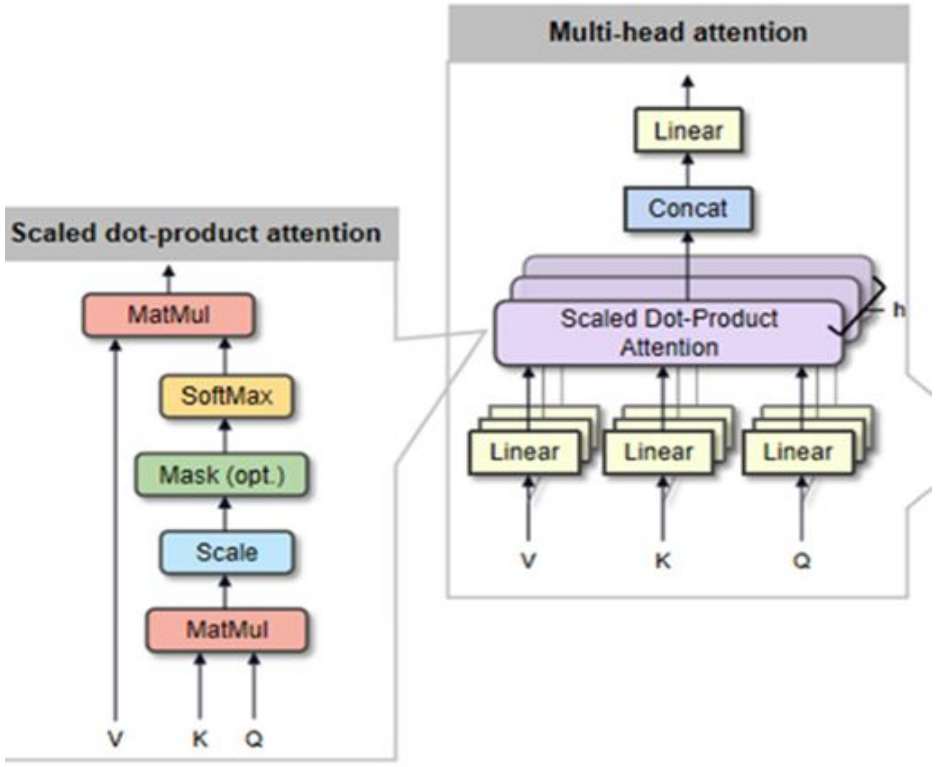
Transformer Architecture – Multi head attention

The linear layer before the scaled dot-product attention in each head of a **Multi-Head Attention** mechanism serves as a **learnable projection**.

Its purpose is to project the input queries (Q), keys (K), and values (V) into separate **lower-dimensional subspaces** for each attention head.

The size of Q, K,V embedding will be the d_{model} / no.of heads ex: d_{model} size = 512 and 8 heads then 64 size embedding's

After the scaled dot product, those get concatenated to come to original d_{model}



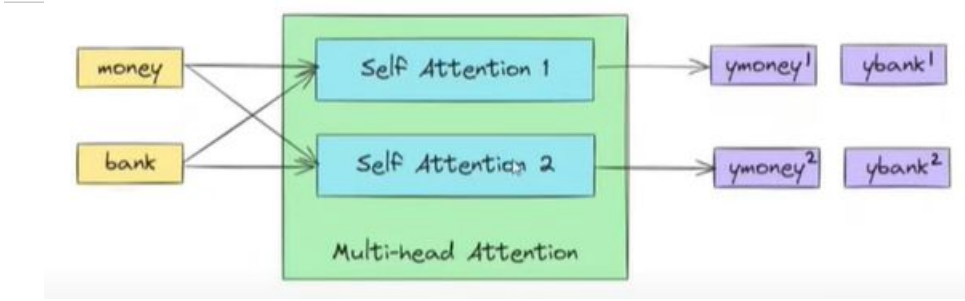
Mathematically

For each attention head i:

- $Q_i = QW_i^Q$
- $K_i = KW_i^K$
- $V_i = VW_i^V$

Where:

- $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ (or d_v) are the learnable weight matrices for the linear layers.
- d_{model} is the dimensionality of the input sequence.



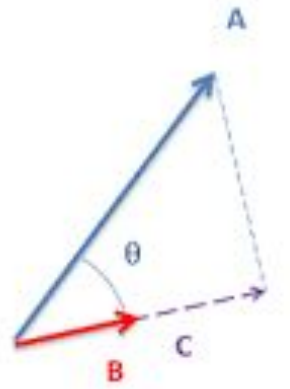
Scaled dot product

Dot Product of Query and Key:

- Calculate $Q \cdot K^T$, which measures how similar the **Query** is to each **Key**.
- This produces a **similarity score**

The dot product $A \cdot B$ calculates how aligned two vectors are, producing a scalar.
The **projection** uses the dot product to calculate the component of one vector along another.

$$\text{Projection of } \mathbf{A} \text{ onto } \mathbf{B} = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{B}|^2}\mathbf{B}$$



$$\vec{A} \cdot \hat{B} = |A||B| \cos(\theta)$$

if the magnitude of B is 1, then...

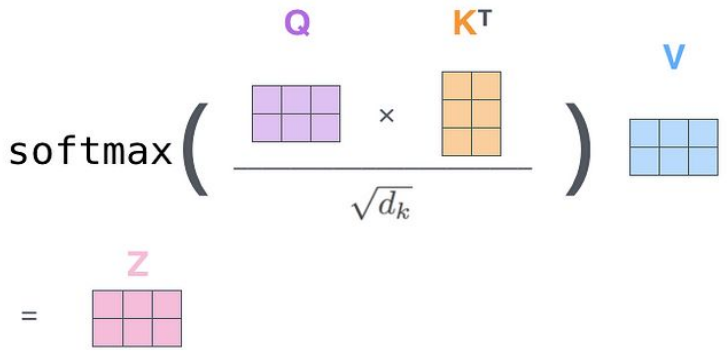
$$C = \vec{A} \cdot \hat{B} = |A| \cos(\theta)$$

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

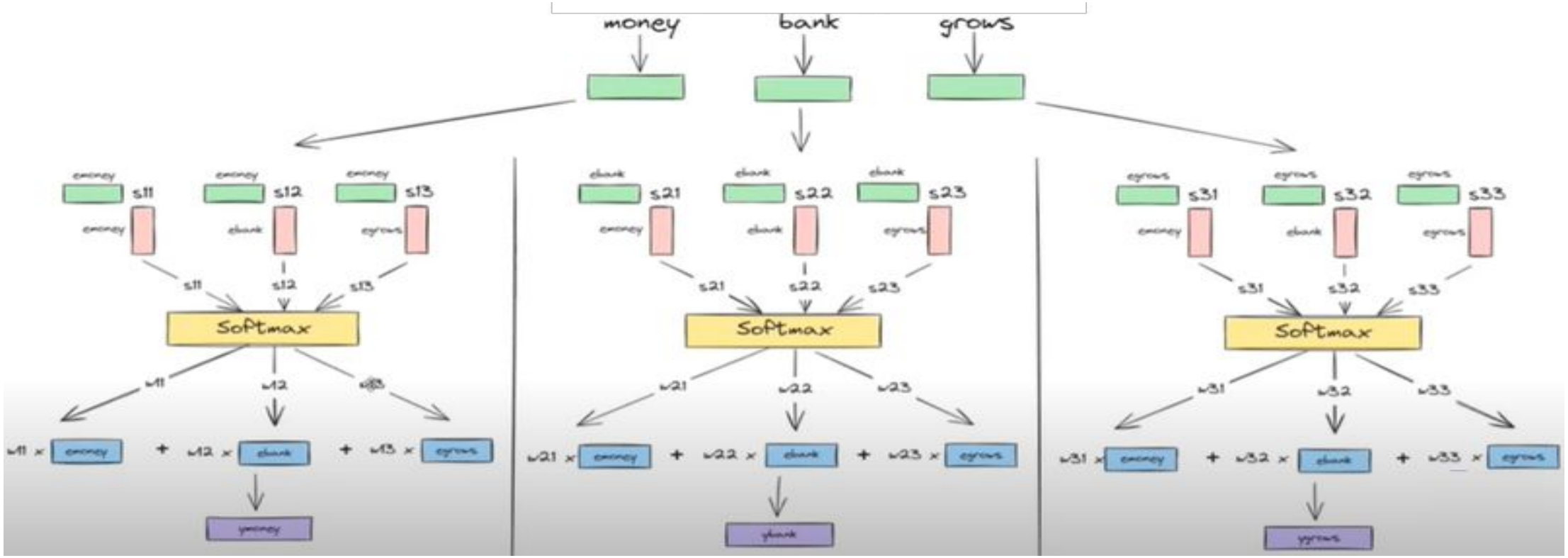
$$\text{Attention Weights} = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$$

$$\text{Output} = \text{Attention Weights} \cdot V$$

When d_k is large, the raw dot-product scores can become very large, leading to extremely small gradients after applying softmax. Scaling prevents this issue.



Scaled dot product



Residual connections

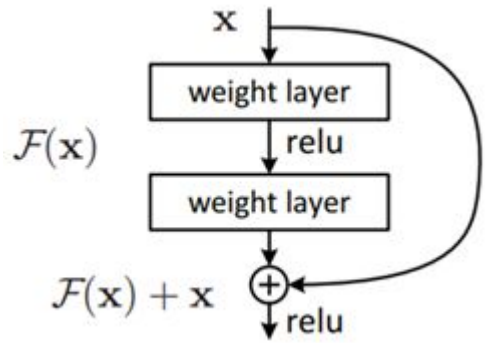
Residual connections involve **adding the input of a layer to its output** before passing the result to the next layer.

Output = Layer(x) + x

where x is the input and Layer(x) is the output from the layer.

Purpose:

1. **Mitigate Vanishing Gradient Problem**:
    1. Residual connections allow gradients to flow directly back through the network during backpropagation, ensuring that earlier layers receive meaningful updates.
2. **Preserve Input Information**:
    1. By adding the input, the network retains information from earlier layers and avoids "overwriting" it with transformations.
3. **Enable Deeper Architectures**:
    1. Transformers often have dozens or even hundreds of layers. Residual connections make training deep architectures feasible by stabilizing training dynamics.



Add Residual Connection:

The attention output is added to the original input of the attention layer.

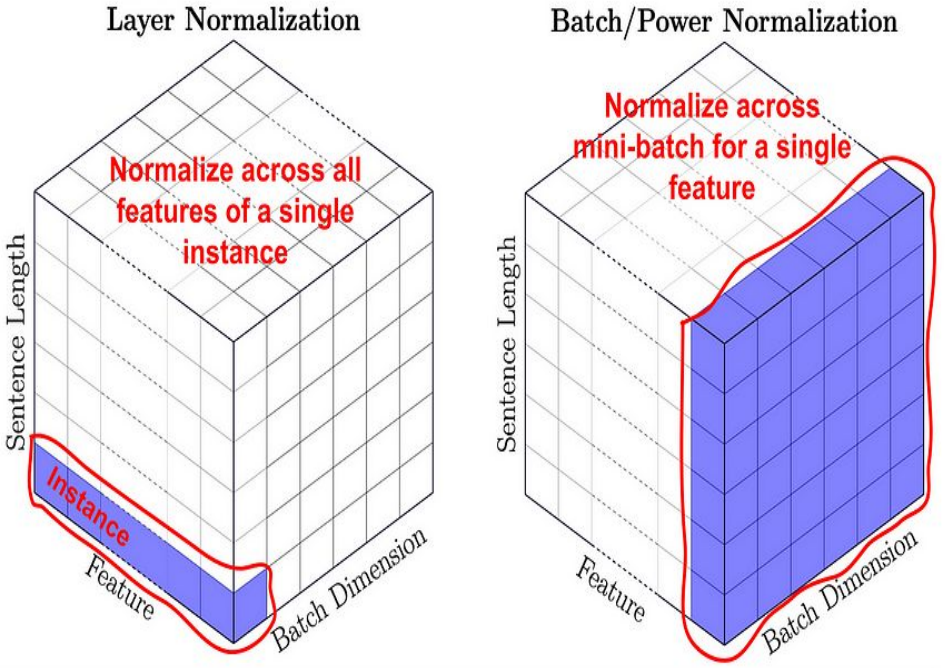
Output = MultiHeadAttention(Q,K,V) + x

Layer normalization

Layer Normalization in Transformers
What Is Layer Normalization?
Layer normalization (LayerNorm) normalizes the **features of each layer** independently, ensuring the mean and variance are consistent.
It normalizes across the **features** within each training sample.

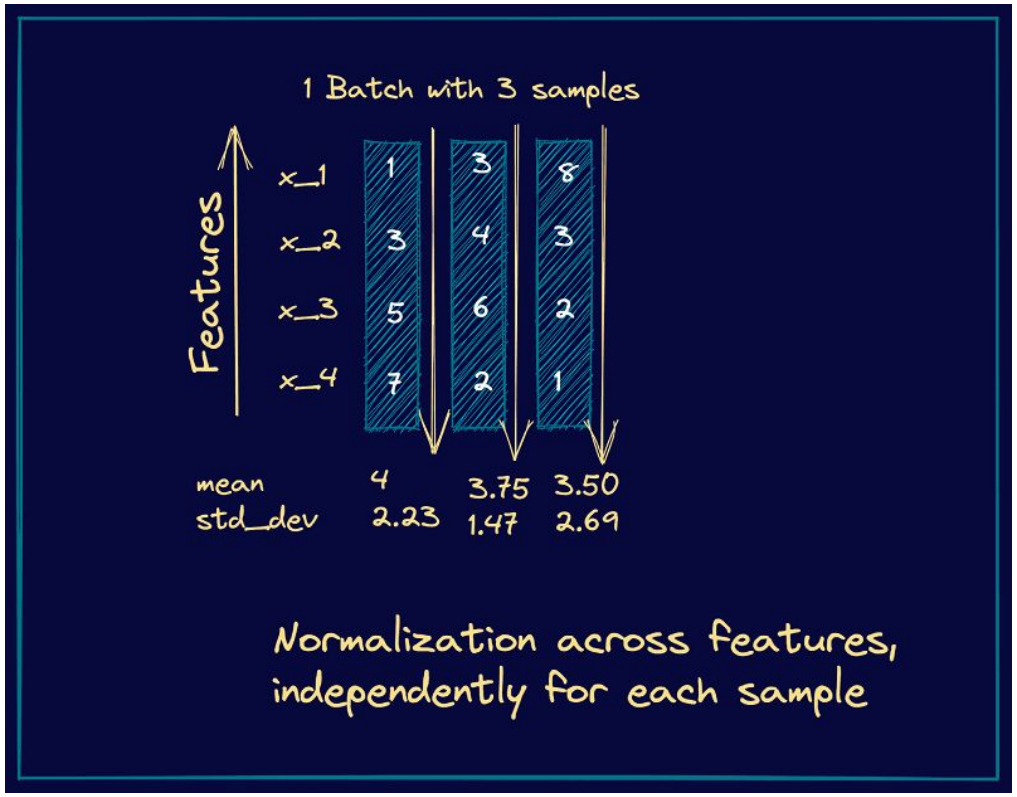
Purpose:
Stabilize Training:
Normalizing features ensures that they are centered and have similar scales, improving convergence during training.
Reduce Internal Covariate Shift:
LayerNorm addresses the issue where the distribution of features changes as gradients propagate back, making training more stable.
Promote Generalization:
By normalizing activations, LayerNorm helps the model generalize better to unseen data.

Apply Layer Normalization:
LayerNorm is applied to the result of the residual connection to normalize the features:
NormedOutput = LayerNorm(AttentionOutput)

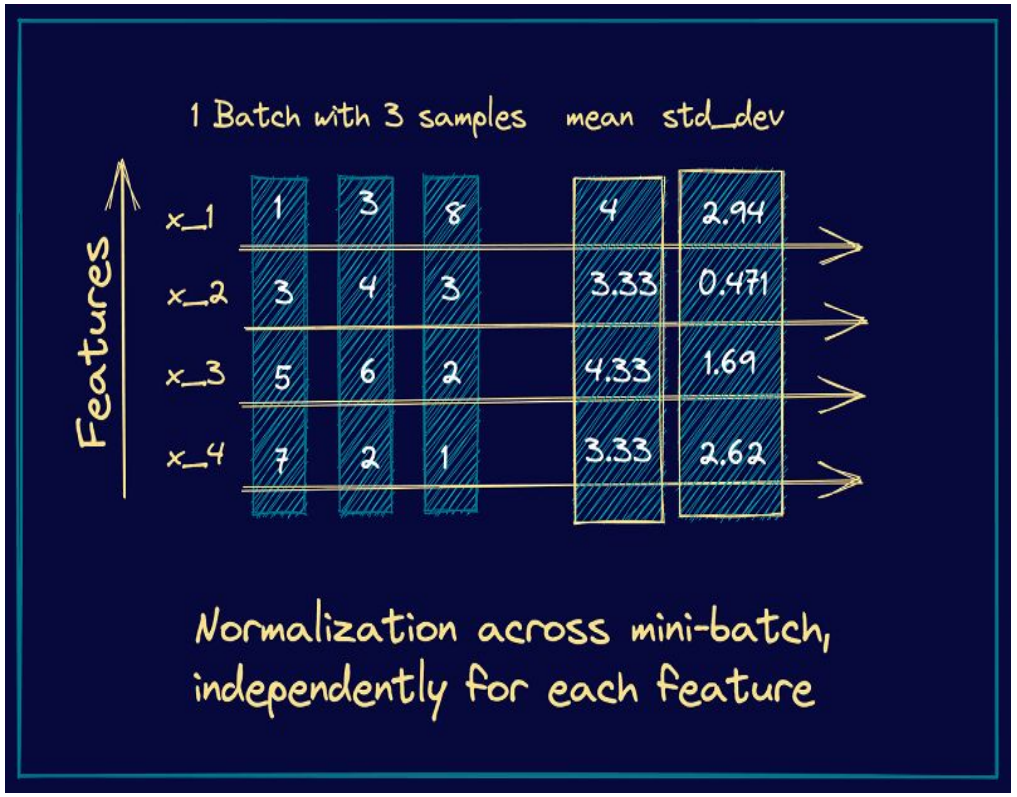


Layer normalization

Layer Normalization



Batch Normalization



Decoder

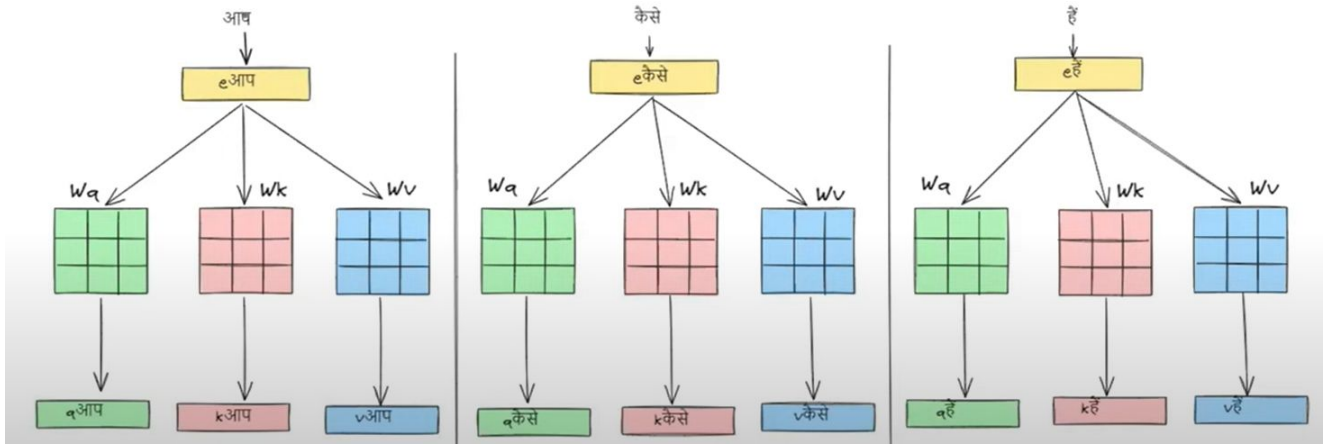
In decoder there are two major components

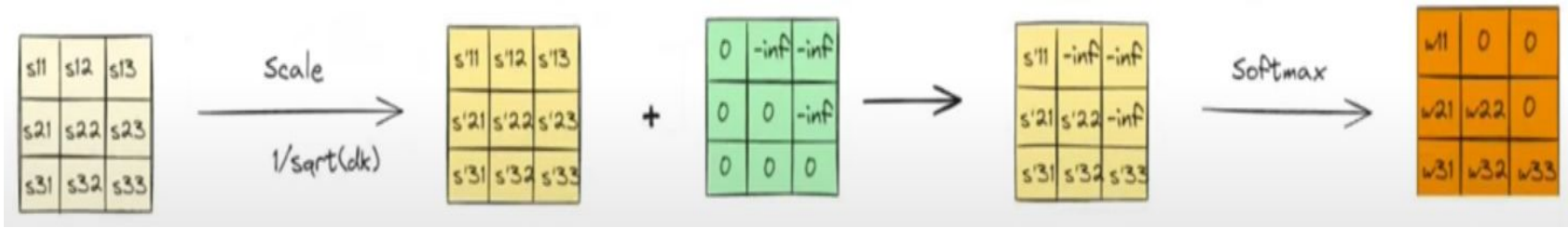
1. Masked multi head attention
2. Cross Attention

At training it is Non auto regressive(achieved with masked attention) at inference it is autoregressive

Masked attention:

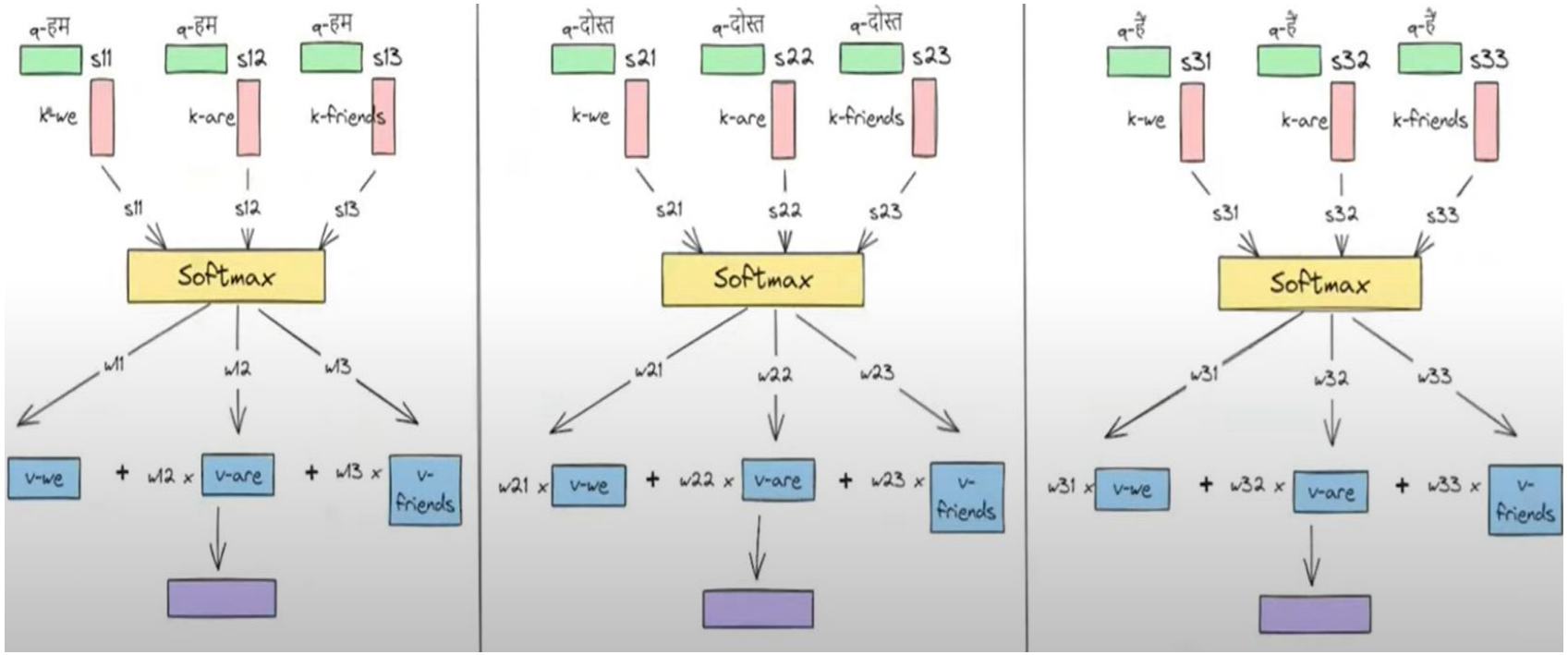
How are you	आप कैसे हो

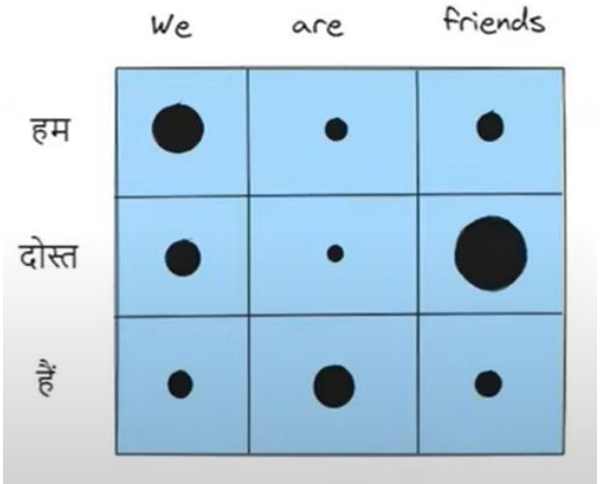




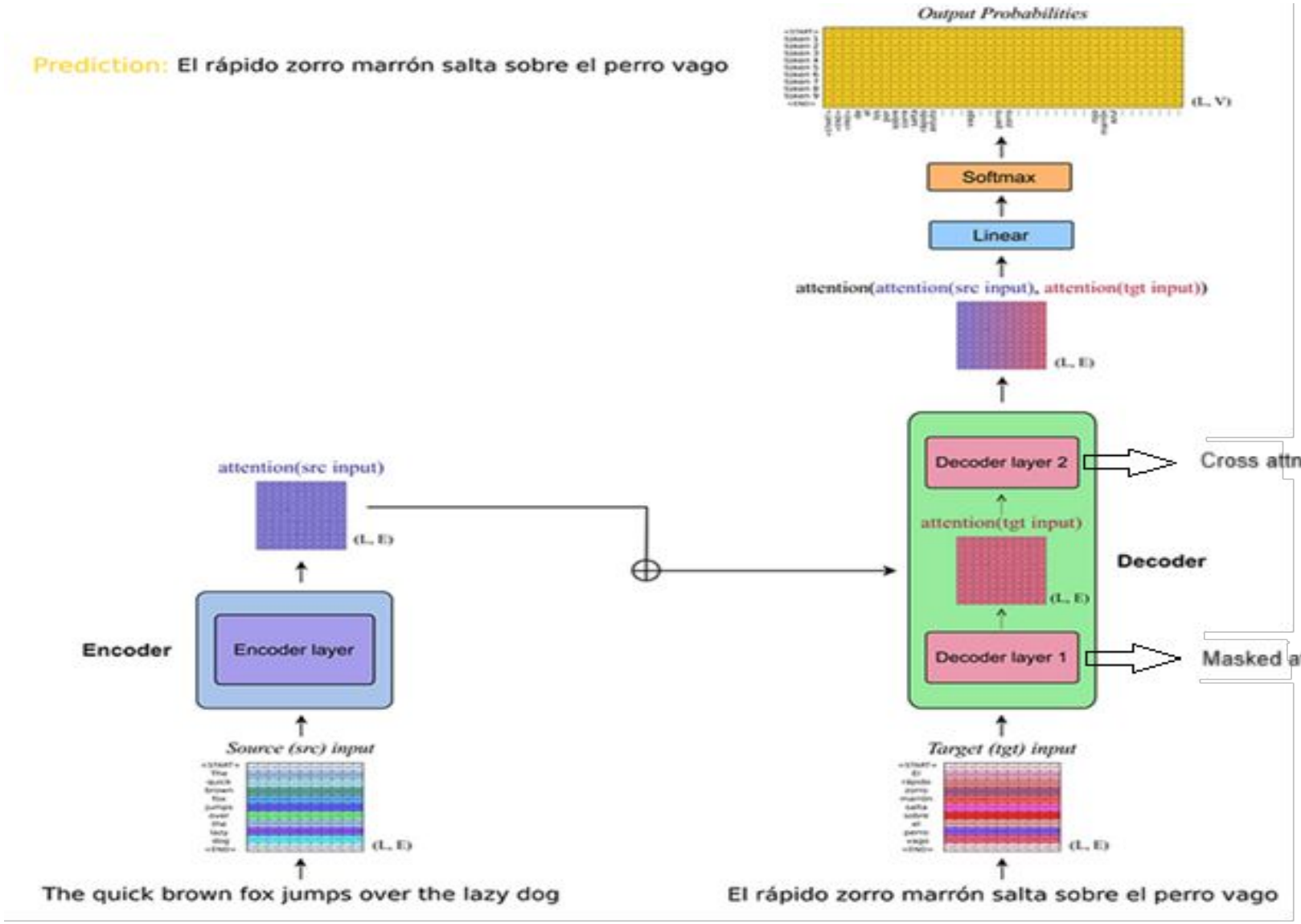
Decoder – cross attention

We are friends	हम दोस्त हैं

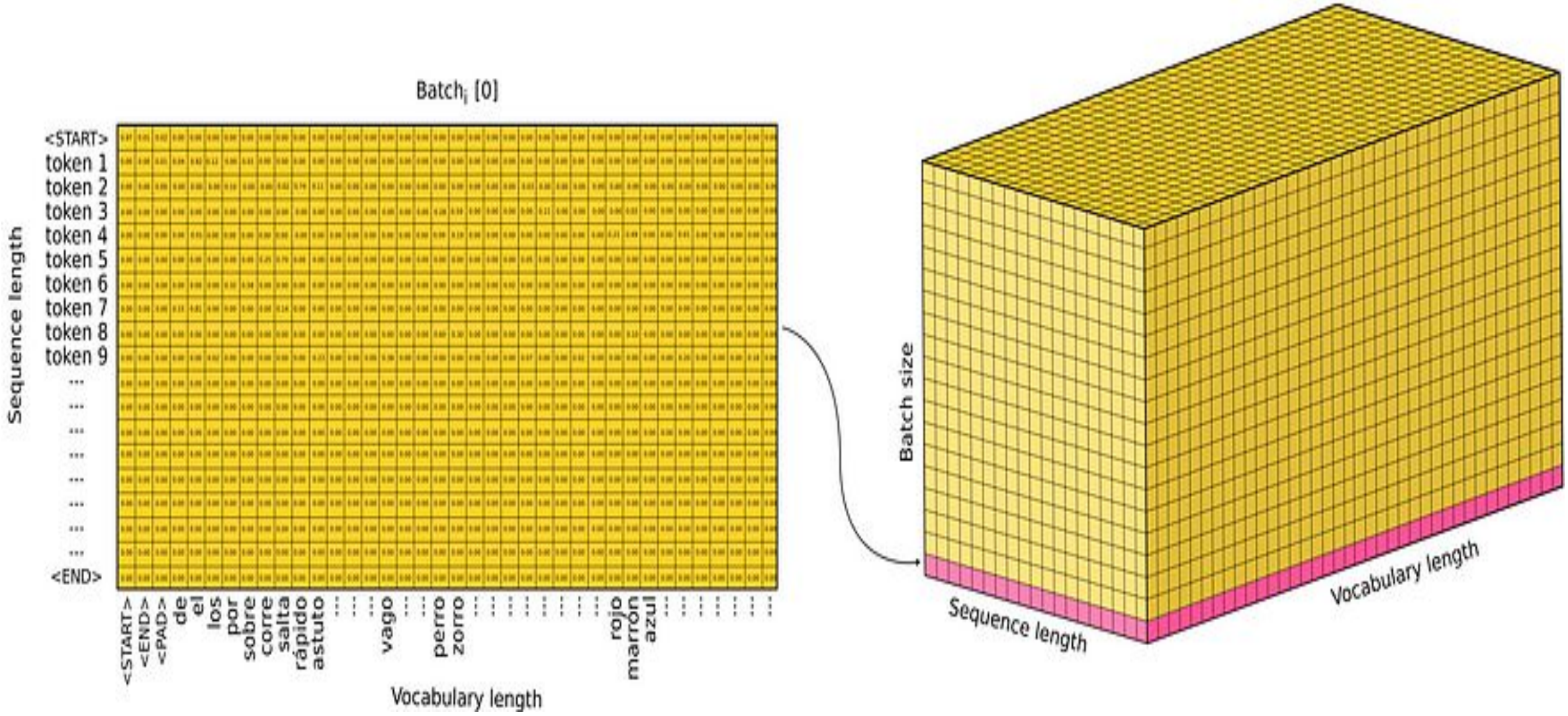




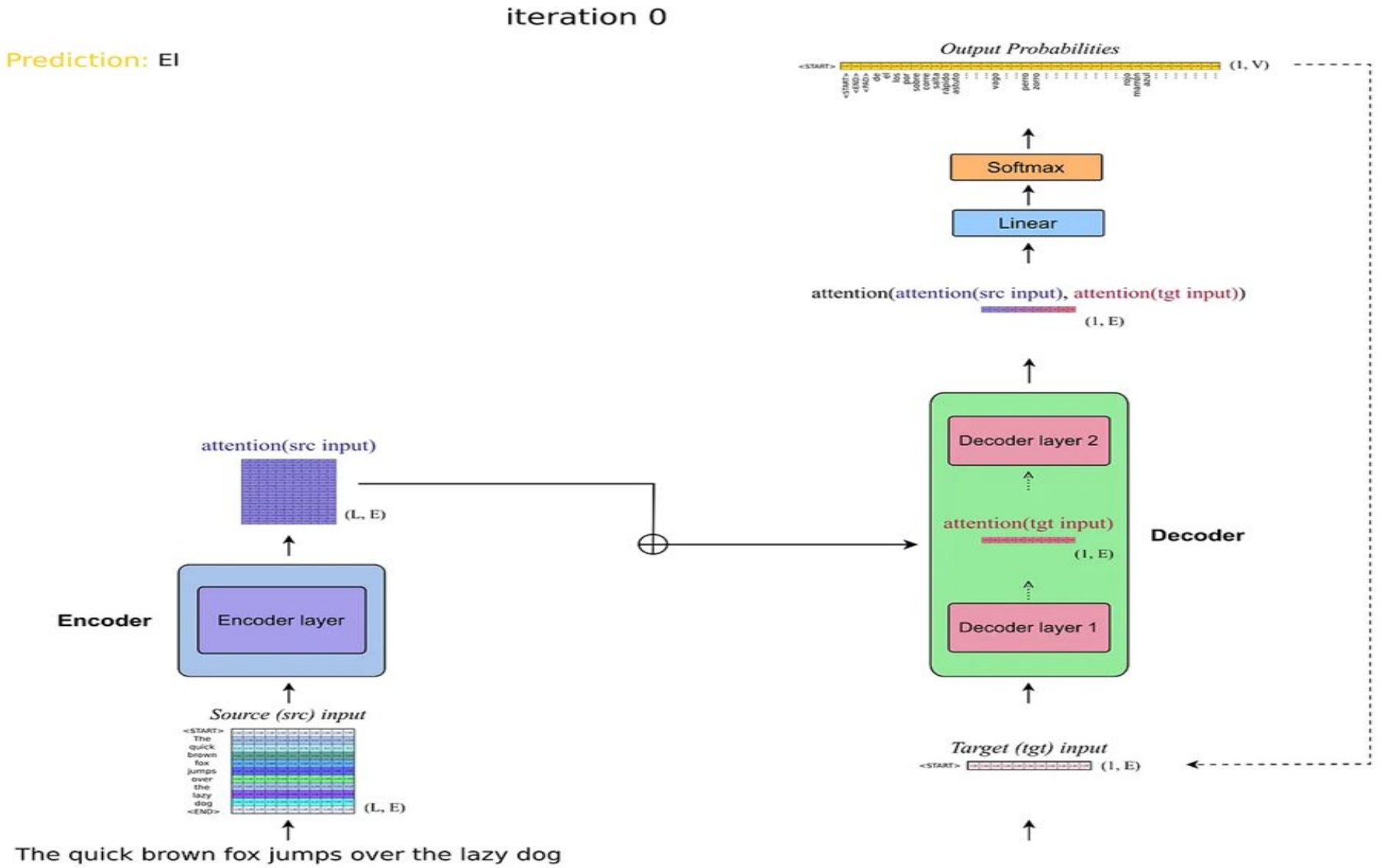
Training


Prediction: El rápido zorro marrón salta sobre el perro vago
Output Probabilities
(L, V)
Softmax
Linear
attention(attention(src input), attention(tgt input))
(L, E)
Decoder layer 2
Cross attn
attention(tgt input)
(L, E)
Decoder
Decoder layer 1
Masked attn
attention(src input)
(L, E)
Encoder
Encoder layer
Source (src) input
(L, E)
Target (tgt) input
(L, E)
The quick brown fox jumps over the lazy dog
El rápido zorro marrón salta sobre el perro vago

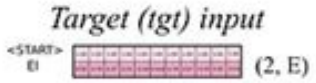
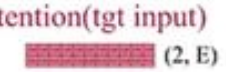
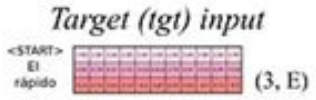
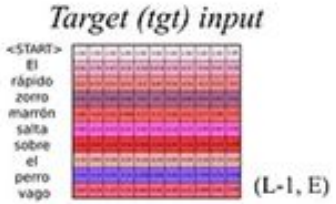
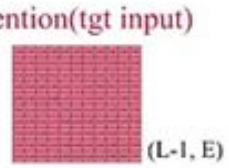
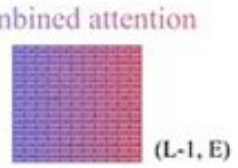
Output Probabilities tensor



Inference


iteration 0
Prediction: El
Output Probabilities
(1, V)
Softmax
Linear
attention(attention(src input), attention(tgt input))
(1, E)
Decoder layer 2
attention(tgt input)
(1, E)
Decoder layer 1
Decoder
attention(src input)
(L, E)
Encoder
Encoder layer
Source (src) input
(L, E)
Target (tgt) input
(1, E)
The quick brown fox jumps over the lazy dog

Inference

Tgt input text	Embedded tgt input	Att tgt input	Combined att	Prediction
iteration 1 El				El rápido
iteration 2 El rápido				El rápido zorro
⋮	⋮	⋮	⋮	⋮
iteration L El rápido zorro marrón salta sobre el perro vago				El rápido zorro marrón salta sobre el perro vago

Attention Vs Self Attention

Attention (Sequence-to-Sequence):
Attention operates between two different sequences — the **query** sequence attends to the **key-value** pairs of another sequence. Works across two sequences (e.g., input sequence ↔ output sequence).

Input Sequence (e.g., "The cat sat") -> Attention -> Output Sequence (e.g., "Le chat s'est assis")
Ex: cross attention

Self-Attention (Within-Sequence):
Self-attention operates within a single sequence — each token attends to all tokens (including itself) in the same sequence.
Input Sequence (e.g., "The cat sat") -> Self-Attention -> Contextualized Representations of each token
i.e The Vs The; The vs Cat ; The Vs sat ; cat Vs sat

Attention: Links information between different sequences.
Self-Attention: Links information within the same sequence.

BERT & GPT

BERT (Bidirectional Encoder Representations from Transformers) Architecture

BERT is a pre-trained language model developed by Google that focuses on understanding the context of words in relation to all other words in a sentence. BERT's architecture is based on the **Transformer Encoder** architecture.

Key Features of BERT:
Bidirectional Context: Unlike traditional models that read text left-to-right or right-to-left, BERT reads text in both directions (i.e., it's bidirectional). This allows it to understand the context of a word based on its surrounding words.
Masked Language Model (MLM): BERT uses a masked language model objective for pre-training. In this, a certain percentage of input tokens are randomly masked, and the model tries to predict them based on their context.
Next Sentence Prediction (NSP): BERT is also trained on a task called Next Sentence Prediction, where the model predicts whether two sentences follow each other in a given context.

BERT Model Applications:
Sentence-level classification tasks (e.g., sentiment analysis, question answering).
Token-level classification tasks (e.g., Named Entity Recognition, part-of-speech tagging).

BERT & GPT

GPT (Generative Pre-trained Transformer) Architecture
GPT, developed by OpenAI, is a generative language model that focuses on language generation. GPT's architecture is based on the **Transformer Decoder** architecture.

Key Features of GPT:
Unidirectional Context: GPT is unidirectional, meaning it processes text in a left-to-right manner, i.e., it generates text word by word from left to right.
Pre-training Objective: GPT is pre-trained using a causal language modeling objective, meaning it predicts the next word given the previous words.
Autoregressive Model: GPT generates each word based on previously generated words, which makes it suitable for text generation tasks.

GPT Model Applications:
Text generation (e.g., writing articles, generating code).
Text completion and augmentation.
Language modeling tasks.
Few-shot learning (especially with GPT-3).

BERT & GPT

Feature	BERT	GPT
Architecture	Transformer Encoder	Transformer Decoder
Training Objective	Masked Language Modeling + Next Sentence Prediction	Causal Language Modeling (Autoregressive)
Context	Bidirectional (Considers both left and right context)	Unidirectional (Only considers left context)
Task Type	Mostly for understanding tasks (e.g., classification, NER)	Primarily for generative tasks (e.g., text generation)
Input Representation	WordPiece + Segment + Positional Embeddings	Token + Positional Embeddings
Fine-Tuning	Fine-tuned for specific tasks (classification, question answering)	Fine-tuned for text generation and few-shot tasks
Model Objective	Predict masked tokens + sentence relationship	Predict next word in the sequence

Thank you