
SOFTWARE-DEVELOPMENT
CIIT

CIIT SOFTWARE & TRAINING

AI TECH TALK

WIEN AM 26. SEPTEMBER 2024

09:15 - 10:15	How AI text is generated - Transformer architecture explained	Benjamin Rowley
10:15 - 10:30	Pause	
10:30 - 11:30	Spring AI - Intelligente Anwendungen mit Spring entwickeln	Bernhard Löwenstein
11:30 - 11:45	Pause	
11:45 - 12:45	Einführung in Machine Learning - Vom linearen Modell zum neuronalen Netz	Benedikt Hausberger
12:45 - 13:30	Mittagspause	
13:30 - 14:30	AI Tools für Entwickler:innen - Praktische Werkzeuge für moderne Software-Entwicklung	Michael Schaffler-Glößl
14:30 - 14:45	Pause	
14:45 - 15:45	Projektbeispiele bei CIIT	Benjamin Rowley & Benedikt Hausberger
15:45 - 16:00	Pause	
16:00 - 17:00	Datenschutz und Datensicherheit: Schutz sensibler Daten im Zeitalter künstlicher Intelligenz	Peter Stangl

LARGE LANGUAGE MODELS

VIENNA 26.09.2024

WHAT ARE LARGE LANGUAGE MODELS (LLM)

- Definition:
  - Large Language Models are AI models designed to understand and generate human language.
- Characteristics:
  - Trained on vast amounts of text data.
  - Capable of various natural language processing (NLP) tasks.
  - Examples: GPT, GEMINI, CLAUDE, etc.

WHAT IS A GPT

- **G**enerative
  - Generates an output of some kind, text/audio/image etc
- **P**re-Trained
  - Model is already trained but may be fine tuned
- **T**ransformer
  - The architecture at the heart of current AI trends
  - Originating from the Google paper “Attention is all you need”
  - In simplest terms – predicts which word comes next based on a probability distribution.

KEY COMPONENTS OF LLMs

- Transformer:
  - Backbone of LLMs.
  - Consists of encoder and decoder layers.
- Self-Attention:
  - Allows the model to focus on different parts of the input sequence.
- Positional Encoding:
  - Adds information about the position of tokens in the sequence.
- Feedforward Neural Networks:
  - Used within transformer layers to process information.

WHAT IS A TRANSFORMER

- Definition:
  - Transformers are a type of neural network architecture introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017.
- Purpose:
  - Designed to handle sequential data with high efficiency and scalability.
- Significance:
  - Foundation for state-of-the-art large language models like GPT, BERT, and T5.

TRANSFORMER ARCHITECTURE

- Components:
  - Encoder:
    - Processes input data and encodes it into context-aware representations.
  - Decoder:
    - Generates output data by decoding the encoder's representations, influenced by attention mechanisms.
- Flow:
  - Input -> Encoder -> Intermediate Representation -> Decoder -> Output
- Key Features:
  - Position encoding
  - Self-attention mechanism
  - Feed-forward neural networks

HOW LLMs WORK

- Tokenization:
- Text is broken down into tokens
    - Example: Break ing down text in to to ken s .
- Tokens can be words, sub-words or characters
- GPT-3 has a vocabulary size of 50,257 tokens
- GPT models use Byte Pair Encoding (BPE) to break the input text into tokens.
- Initial Tokenization: The input text is first broken down into the smallest units defined in the vocabulary, often starting with characters or sub-words.
- Sub-word Matching: The tokenizer matches sequences of characters or sub-words in the input text against the vocabulary. This involves:
- Longest Match First: The tokenizer tries to match the longest possible sub-words from the vocabulary first.
- Greedy Approach: For each part of the text, it finds the longest token that exists in the vocabulary and uses it.

HOW LLMs WORK

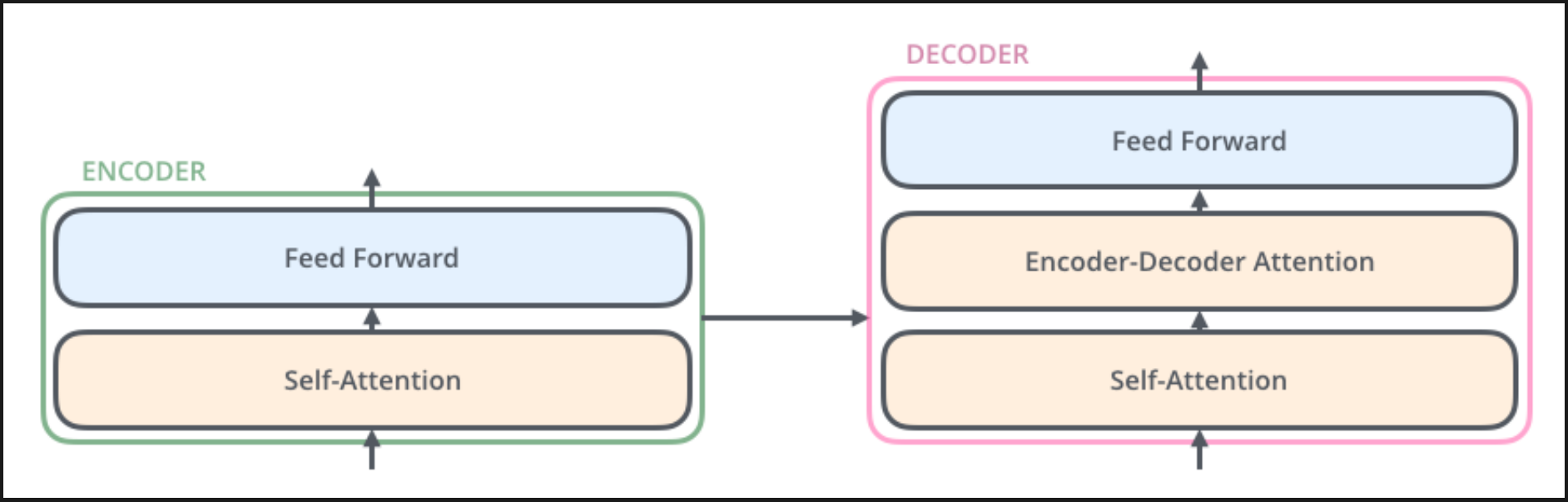
- Tokenization Output:
- Sequence of token IDs used to fetch the corresponding vectors from the Embedding Matrix.
- Open AI models use vectors of between 768 and 12288 dimensions, depending on the model size.
- Each vector captures complex semantic and syntactic information about a token.
- For each token in the input, a positional encoding is then added.
- The Transformer has no sense of a tokens position relative to the other tokens in the input.
- Tokens are processed in parallel so the positional encoding is essential.

MORE ON TOKENS

- Tokens are essential in how AI models process text.
- They are the bridge between words we understand and the format the model can work with.
- Tokens = money – cloud AI services will charge per token so its essential to track how many we are using.
- Token limits – AKA context window – the maximum number of tokens a model can handle in a request – includes all user inputs (system prompts, documents, list of functions, conversational context etc) and generated output.

TRANSFORMER ARCHITECTURE – SIMPLE OVERVIEW

- Encoder's inputs first flow through a self-attention layer that helps the encoder look at other words in the input sentence as it encodes a specific word.
- Outputs of the self-attention layer are fed to a feed-forward neural network (Multi-layer Perceptron). The exact same feed-forward network is independently applied to each position.
- The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence



TRANSFORMER LAYERS: ENCODER

- Purpose:
  - To convert input sequences into a context-aware representation.
- Structure:
  - Multiple identical layers, each containing:
    - Self-Attention Mechanism
    - Feed-Forward Neural Network
    - Residual Connections and Layer Normalization
- Process:
  1. Input tokens are embedded and position encoded.
  2. Self-attention mechanism processes the embeddings.
  3. Feed-forward neural network (Multi-layer Perceptron) refines the processed embeddings.
  4. Residual connections ensure gradient flow meaning the original input is preserved across layers which helps maintain strong contextual information and prevents original meaning being lost.
  5. Normalization ensures that each layer's output is stable, preventing drastic changes in the representation as the model processes each token, allowing it to produce text that is contextually relevant and consistent with previous tokens.

INPUT EMBEDDING AND POSITION ENCODING

- Input Embedding:
  - Transforms input tokens into dense vectors of fixed size.
  - Example: "The cat sat" -> [E_The, E_cat, E_sat]
- Position Encoding:
  - Problem:
    - Transformers have no inherent sense of token order.
  - Solution:
    - Use position encoding to inject information about the token positions.
  - Adds positional information to embeddings to retain the order of tokens.
  - Types:
    - Sinusoidal Encoding: Uses sine and cosine functions.
    - Learned Encoding: Learned during training.

WHY GPT USES LEARNED POSITIONAL ENCODING?

- Flexibility: Learned positional encodings allow the model to find the most useful way to represent positional information during training. The model can decide what kind of positional relationship matters for the specific data and task.
- Integration with Language Modelling: Since GPT models are pre-trained on large text corpora, learned positional encodings can adapt to linguistic patterns and textual data in a way that might be more effective than using fixed sinusoidal functions

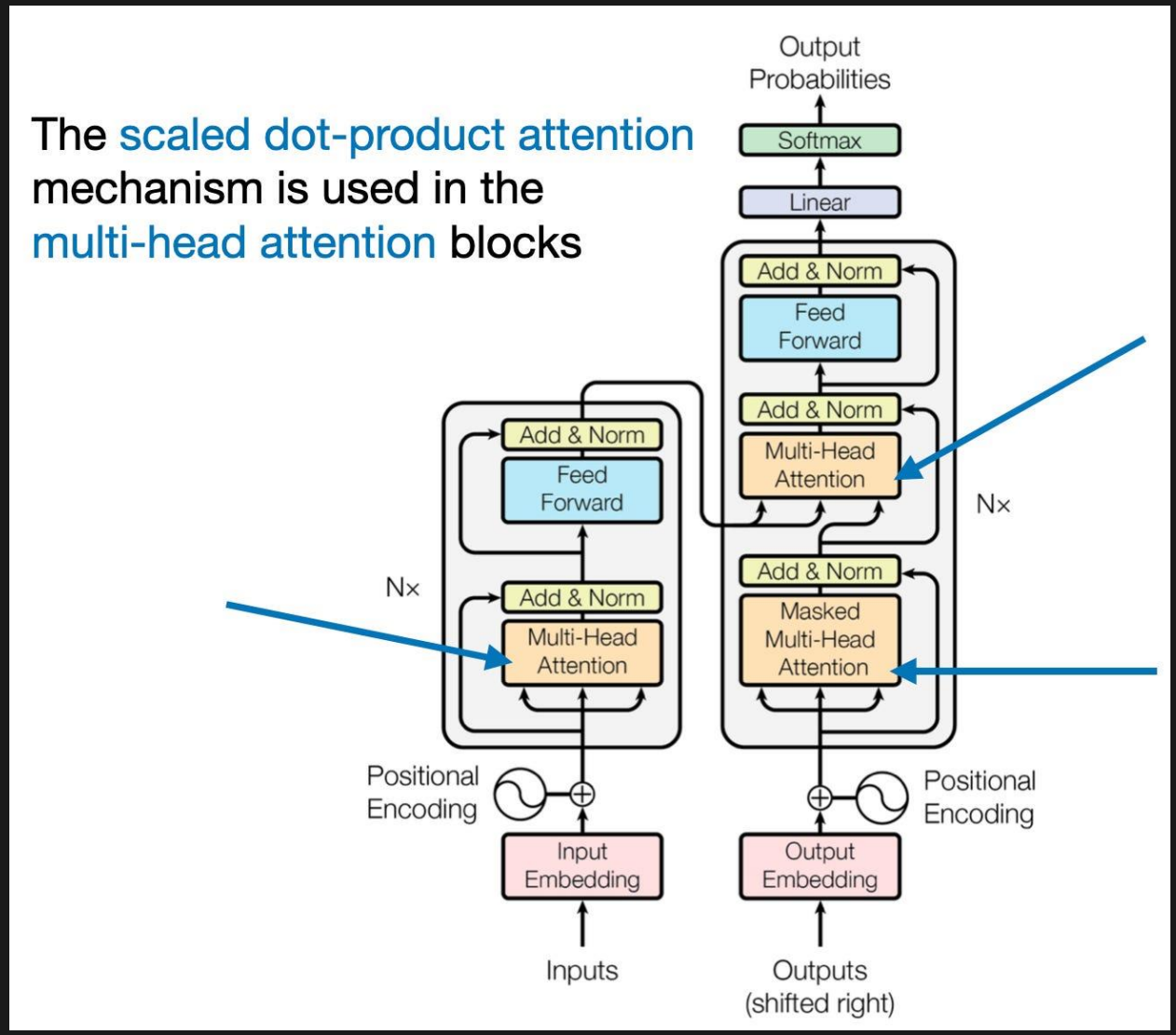
SELF ATTENTION MECHANISM

Input Embedding:

- The input tokens (words or sub-words) are first converted into embeddings, which are dense vector representations. These embeddings capture the semantic information of the tokens.

Positional Encoding:

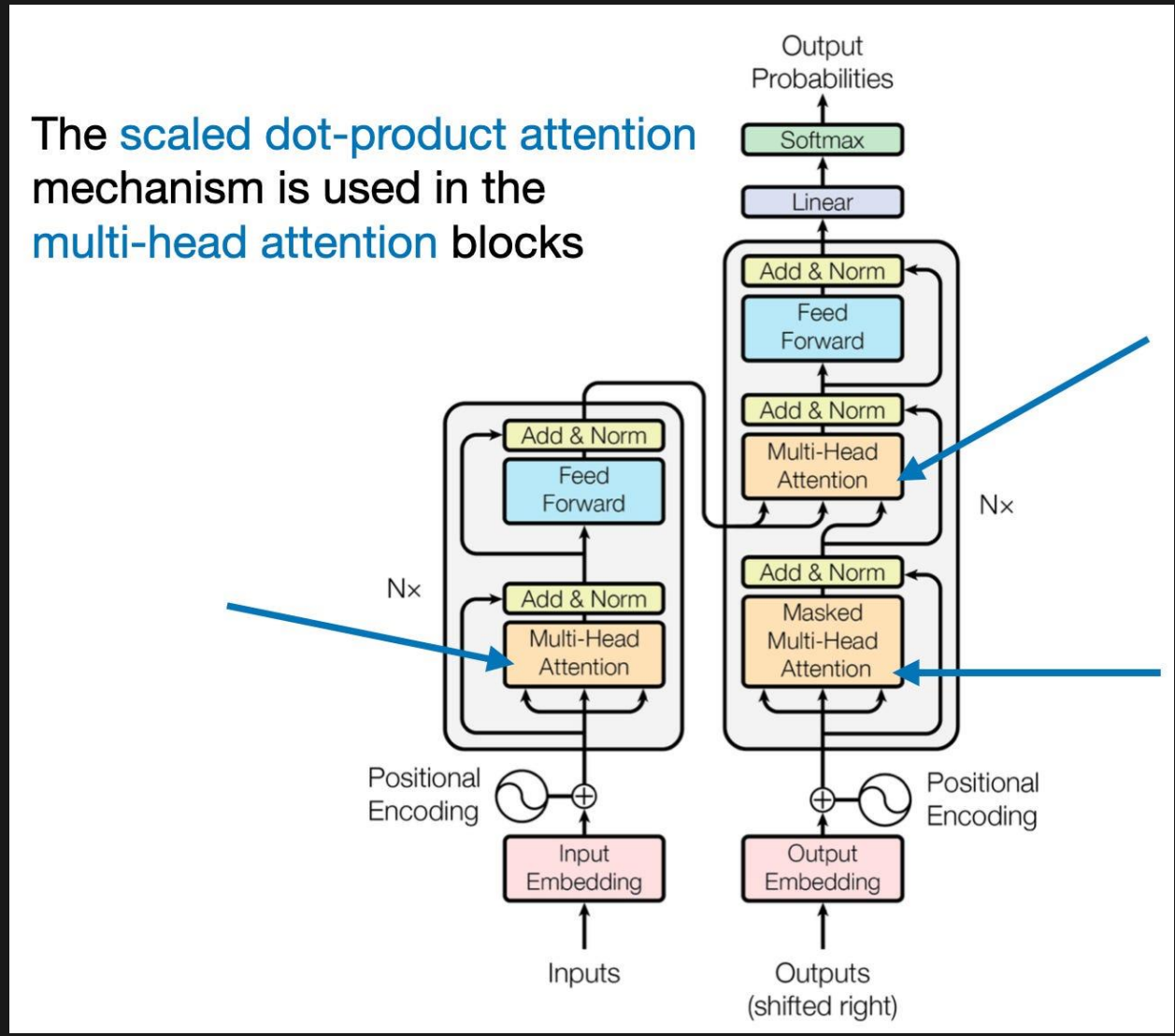
- Since the Transformer architecture does not inherently understand the order of the sequence, positional encodings are added to the input embeddings. This allows the model to incorporate the position information of each token in the sequence.



SELF ATTENTION MECHANISM

Multi-Head Attention:

- **Scaled Dot-Product Attention**: This is the core mechanism of the attention layer. It computes attention scores based on the dot product of query (Q), key (K), and value (V) vectors, which are derived from the input embeddings.
- **Multi-Head Attention**: Instead of performing a single attention function, the model uses multiple attention heads. Each head operates in a different subspace of the embeddings, allowing the model to capture different aspects of the relationships between tokens. The outputs of these attention heads are concatenated and linearly transformed.

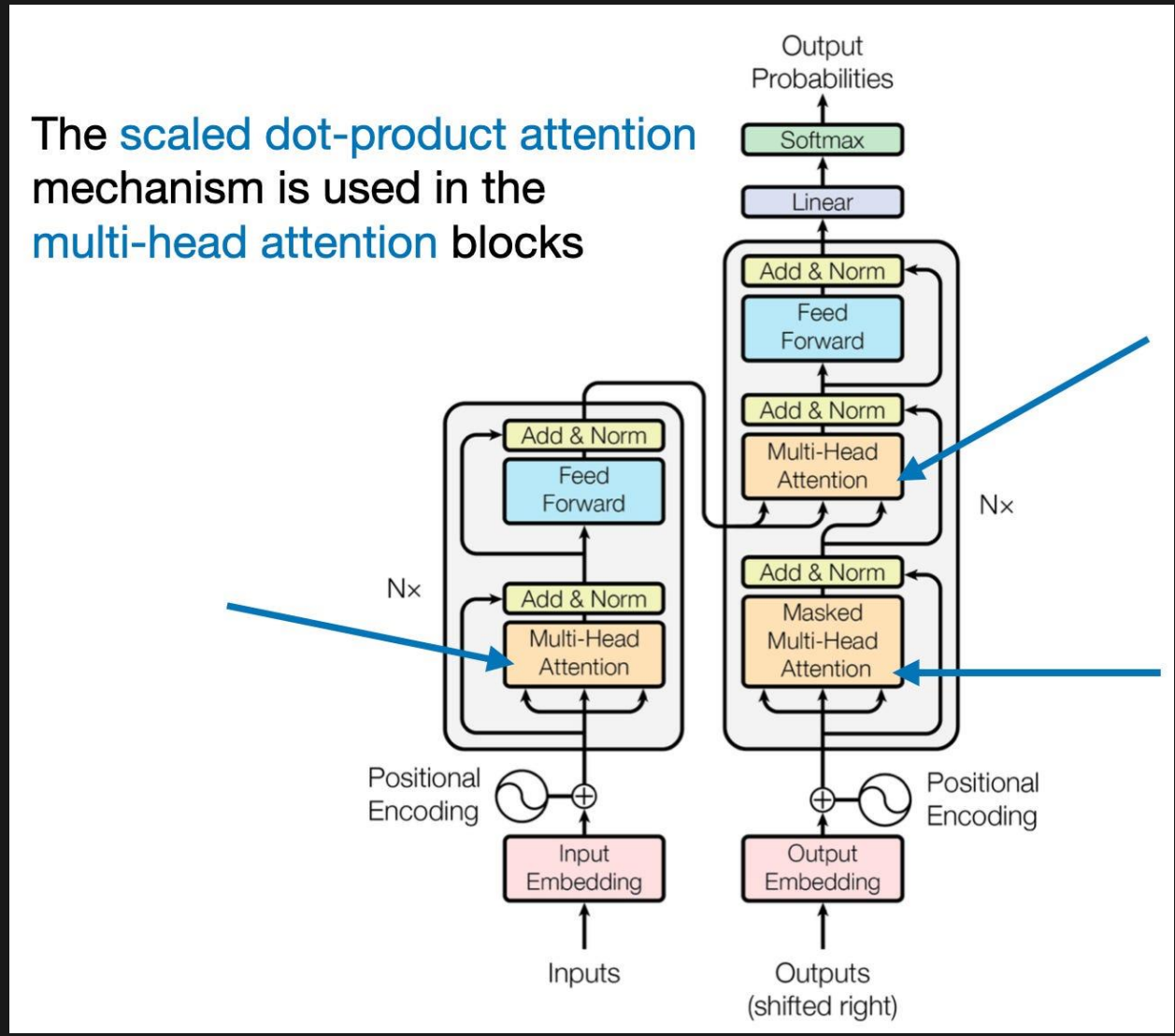


SELF ATTENTION MECHANISM

Add & Norm:
- After the multi-head attention, a residual connection is applied where the input of the attention layer is added to its output. This is followed by layer normalization. This helps in stabilizing and accelerating the training.

Feed Forward:
- A fully connected feed-forward neural network is applied to each position independently. It consists of two linear transformations with a ReLU (Rectified Linear Unit) activation in between.

Encoder Stack:
- The encoder consists of multiple (Nx) identical layers, each containing a multi-head attention mechanism and a feed-forward network.



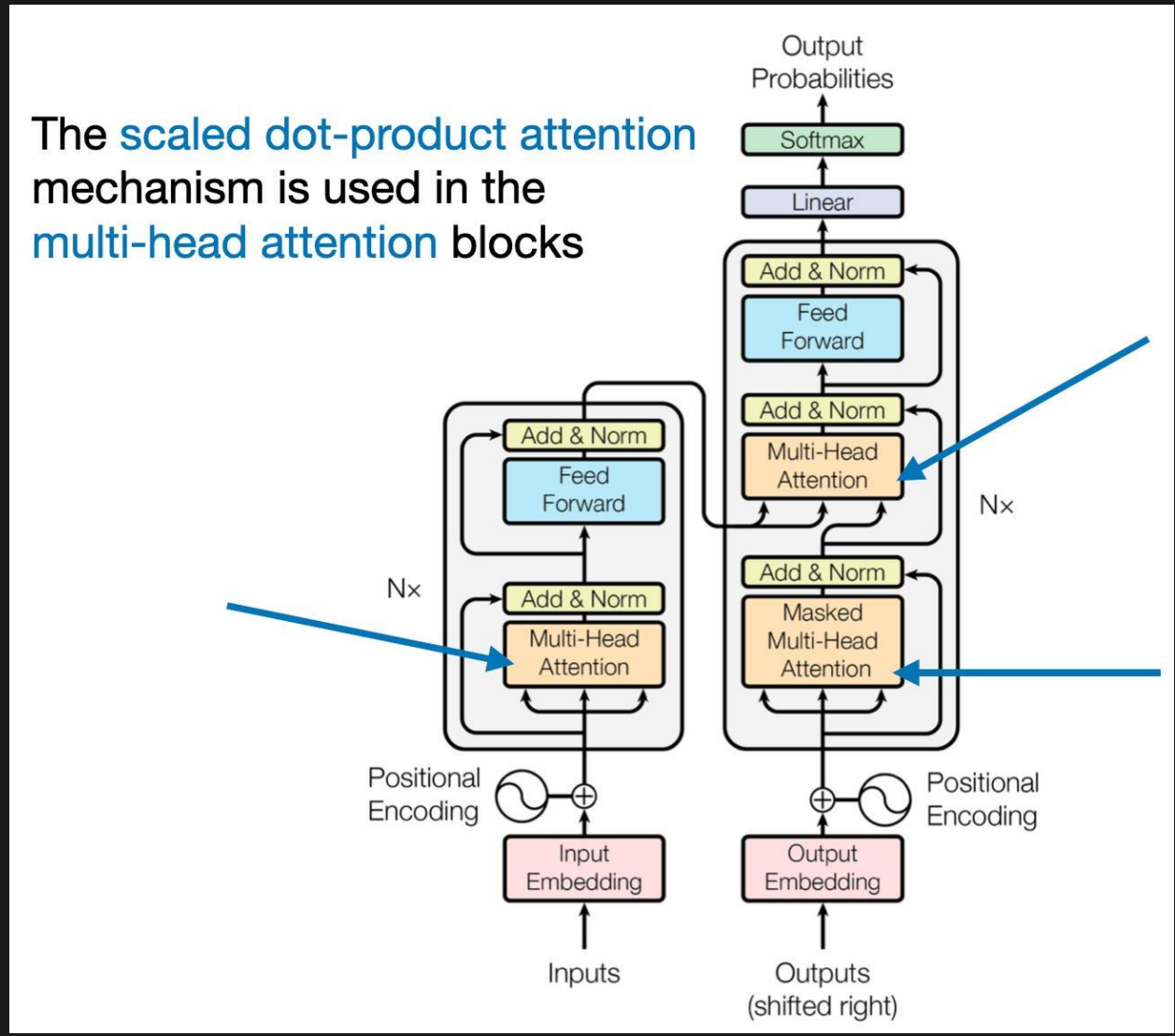
SELF ATTENTION MECHANISM

Decoder Stack:

- Similar to the encoder, the decoder also consists of multiple (Nx) identical layers. However, it includes an additional masked multi-head attention mechanism before the regular multi-head attention and feed-forward layers.
- Masked Multi-Head Attention: In the decoder, the masked attention ensures that the predictions for a particular token depend only on the known outputs preceding it (to prevent the model from cheating by looking ahead).

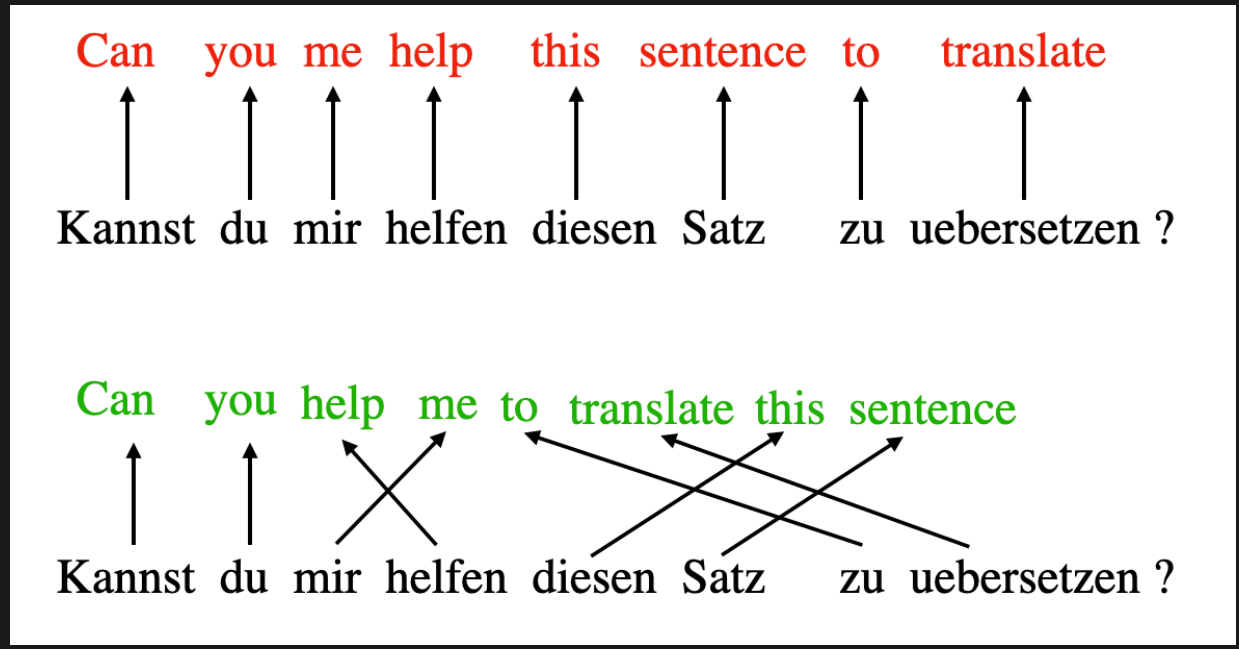
Output Embedding:

- The output from the decoder is processed through a linear transformation and a Softmax layer to generate the final probabilities of the next token in the sequence.



SELF ATTENTION MECHANISM

- Initially developed to improve machine translation algorithms.
- Translating a sentence word by word ignores complex gramatical structures and idiomatic expressions unique to each language.
- To overcome this issue, attention mechanisms were introduced to give access to all sequence elements at each time step.

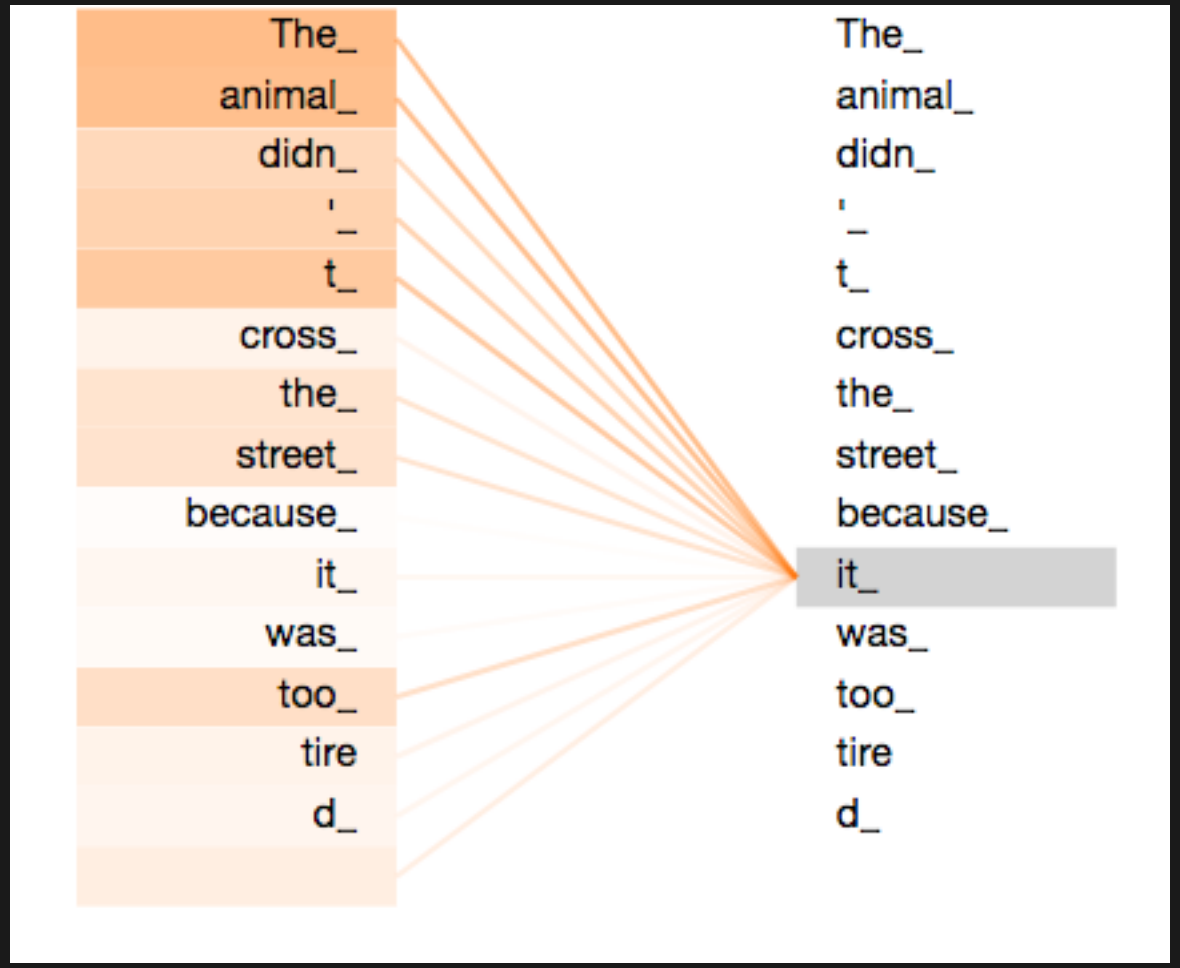


SELF ATTENTION MECHANISM

- Purpose:
  - Allows the model to weigh the importance of different parts of the input sequence dynamically.
  - Each token gains contextual semantic meaning from the other tokens within the context window.
- Steps:
  1. Query, Key, Value Vectors:
     - For each input token, generate three vectors: Query (Q), Key (K), and Value (V).
     - Query (Q): Represents what a token is "looking for" in other tokens. It asks, "How much attention should I give to each other token?"
     - Key (K): Encodes information about each token, answering, "How relevant am I to other tokens?"
     - Value (V): Contains the actual content or information of the token that will be passed on if it's deemed important (based on Q and K).
  2. Attention Scores:
     - Calculate scores using the dot product of Q and K, scaled by the square root of the dimension.
  3. Softmax:
     - Apply Softmax to obtain attention weights.
  4. Weighted Sum:
     - Compute the weighted sum of V vectors using the attention weights.

SELF ATTENTION MECHANISM

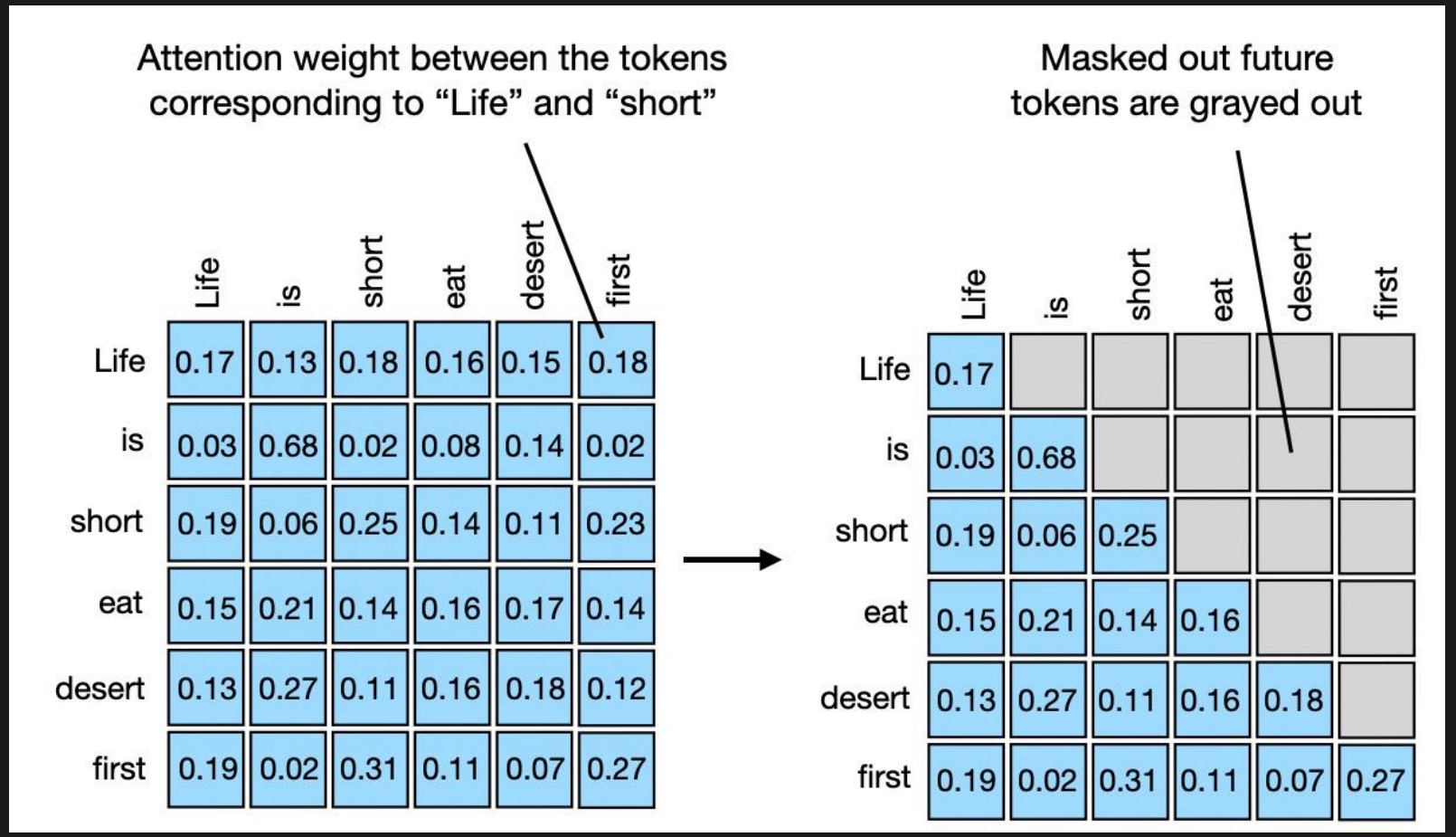
- As the model processes each word, self attention allows it to look at other words in the input sequence for clues that can help lead to a better encoding for this word.
- Self-attention is the method the Transformer uses to bake the “understanding” of other relevant words into the one we’re currently processing.
- As we encode the word “it” part of the attention mechanism was focussing on “The animal”.
- This contextual meaning is used to update the encoded value for “it”



ENCODER – DECODER DIFFERENCES

- The self attention layers in the decoder operate in a slightly different way than the one in the encoder:
- In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to -∞) before the SoftMax step in the self-attention calculation.
- The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack.

VISUALISING MASKED SELF-ATTENTION



FINAL LINEAR AND SOFTMAX LAYER

- Decoder stack outputs a vector of floating point numbers.
- How do we turn that into a word?
- Final Linear layer followed by SoftMax Layer.
- The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits vector.
- If the model knows 10,000 English words (output vocabulary), learned from its training dataset.
- This would make the logits vector 10,000 cells wide – each cell corresponding to the score of a unique word.
- That is how we interpret the output of the model followed by the Linear layer.
- The SoftMax layer then turns those scores into probabilities (all positive, all add up to 1.0)
- The cell with the highest probability is chosen.
- The word associated with it is produced as the output for this time step.

REFERENCES

- https://jalammar.github.io/illustrated-word2vec/
- https://platform.openai.com/docs/concepts
- https://platform.openai.com/docs/concepts
- https://github.com/spring-projects/spring-ai?tab=readme-ov-file
- https://jalammar.github.io/illustrated-transformer/

VIELEN DANK!