Generative AI - LLM

Generating text

Group (1):
- Abdallah Elsharawy - 617292
- Dawit Admassu - 617303
- Iman Asfaw - 617332

Introduction

Text generation in Natural Language Processing (NLP) is the task of automatically producing **coherent** and **contextually appropriate** text from a given input.

The input: can range from a prompt, a set of keywords, a partial sentence, or structured data,
The output (generated text): can vary in length and complexity depending on the task.

The key goals of text generation include producing text that is **fluent, contextually relevant,** and **semantically accurate.**

Applications of Text Generation:

- Chatbots and Conversational Agents: Generating human-like responses in dialogue systems.
- Machine Translation: Translating text from one language to another.
- Summarization: Creating concise summaries of longer documents.
- Content Creation: Writing articles, stories, or other forms of content automatically.
- Code Generation: Generating code snippets from natural language descriptions.
- Question Answering: Providing answers to questions posed in natural language.

Advances in Text Generation

1. Recurrent Neural Networks (RNNs): Early models for text generation that processed sequences of words one at a time, maintaining a hidden state to keep track of context. Variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) were developed to handle longer dependencies better.

2. Seq2Seq Models with Attention: Sequence-to-Sequence models, often with an encoder-decoder architecture, where the encoder processes the input sequence and the decoder generates the output sequence. The attention mechanism allows the model to focus on different parts of the input sequence when generating each word of the output.

3. Transformers: The Transformer architecture introduced the use of self-attention mechanisms to handle sequences in parallel, allowing for more efficient training and better handling of long-range dependencies. Transformers are now the foundation of most state-of-the-art text generation models.

4. Pre-trained Language Models:
   - GPT (Generative Pre-trained Transformer): A series of models by OpenAI, including GPT-2 and GPT-3, which use large-scale pre-training on diverse text corpora followed by fine-tuning on specific tasks. These models generate text by predicting the next word in a sequence given the previous words, leveraging a vast amount of learned knowledge.
   - BERT (Bidirectional Encoder Representations from Transformers): Although BERT is primarily used for understanding and classification tasks, it has influenced text generation through its bidirectional context understanding, which is adapted in models like T5 (Text-to-Text Transfer Transformer).

Recurrent Neural Network (RNNs)

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to recognize patterns in sequences of data, such as time series or natural language.

RNNs are neural networks capable of handling sequential data.

More generally, RNNs can work on sequences of arbitrary lengths, rather than on fixed-sized inputs.

Unlike traditional feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a 'memory' of previous inputs by passing information through hidden states. This makes them particularly well-suited for tasks where the order of the data is important.

- $h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$
- $y_t = W_{hy}h_t$

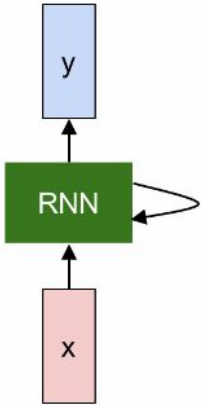
$$h_t = f_W(h_{t-1}, x_t)$$

new state

old state

input vector at some time step

some function with parameters W



Recurrent Neural Networks (RNNs)

Feedforward neural networks: the activations flow only in one direction, from the input layer to the output layer.

RNNs are composed of neurons receiving inputs, producing an output, and sending that output back to itself.

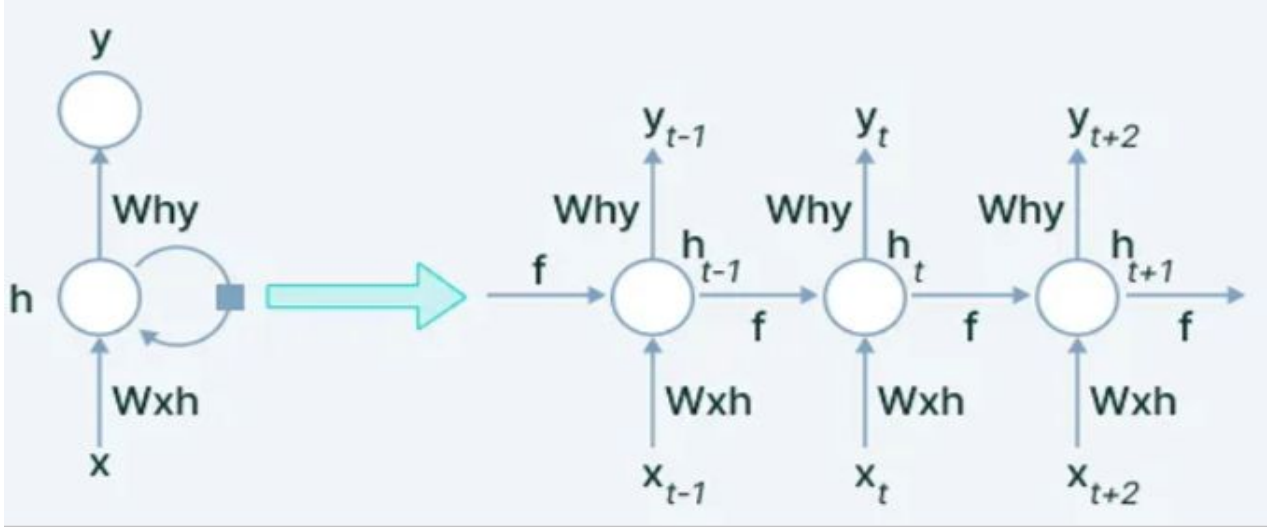


Figure 1. A layer of recurrent neurons (left) unrolled through time (right)

RNN Drawbacks

Vanishing/exploding gradients makes: *“standard RNNs fail to learn in the presence of time lags greater than 5 – 10 discrete time steps between relevant input events and target signals.”*

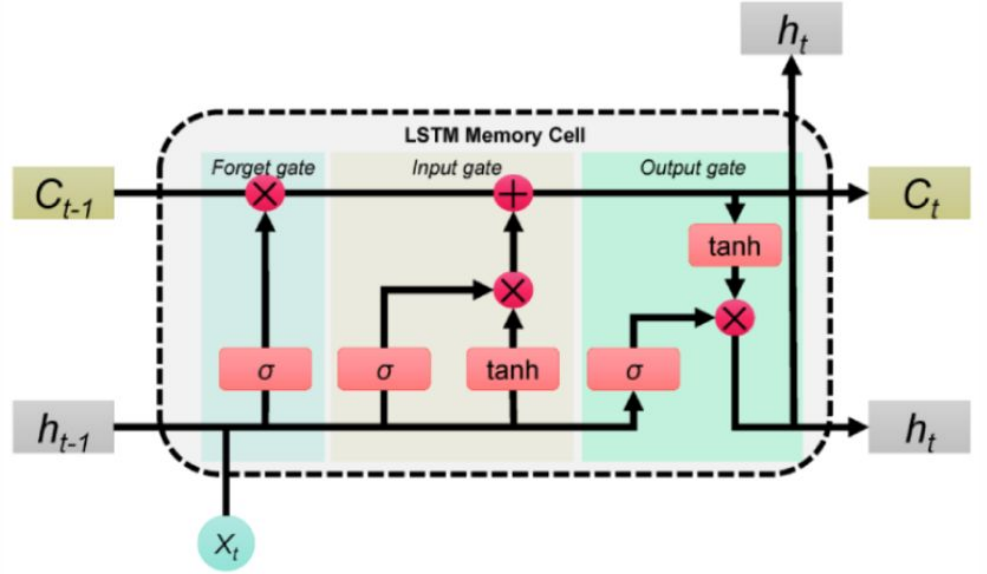
RNNs are very slow to train

Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) specifically designed to address the **vanishing and exploding gradient problems** encountered during the training of traditional RNNs. LSTMs introduce a memory cell and gating mechanisms that allow the network to maintain and update its state over long sequences, thereby improving its ability to learn long-term dependencies.

Cell State (Ct-1): Acts as a memory that carries information across the entire sequence, preserving long-term dependencies.

Hidden State (h_t)): Represents the output of the LSTM unit at each time step and is used for making predictions.

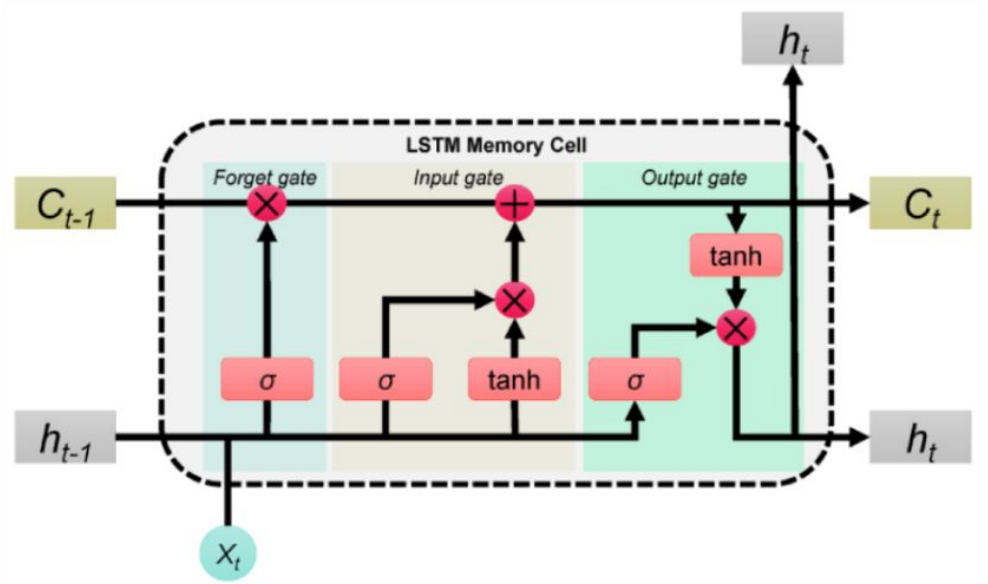


Long Short Term Memory LSTMs

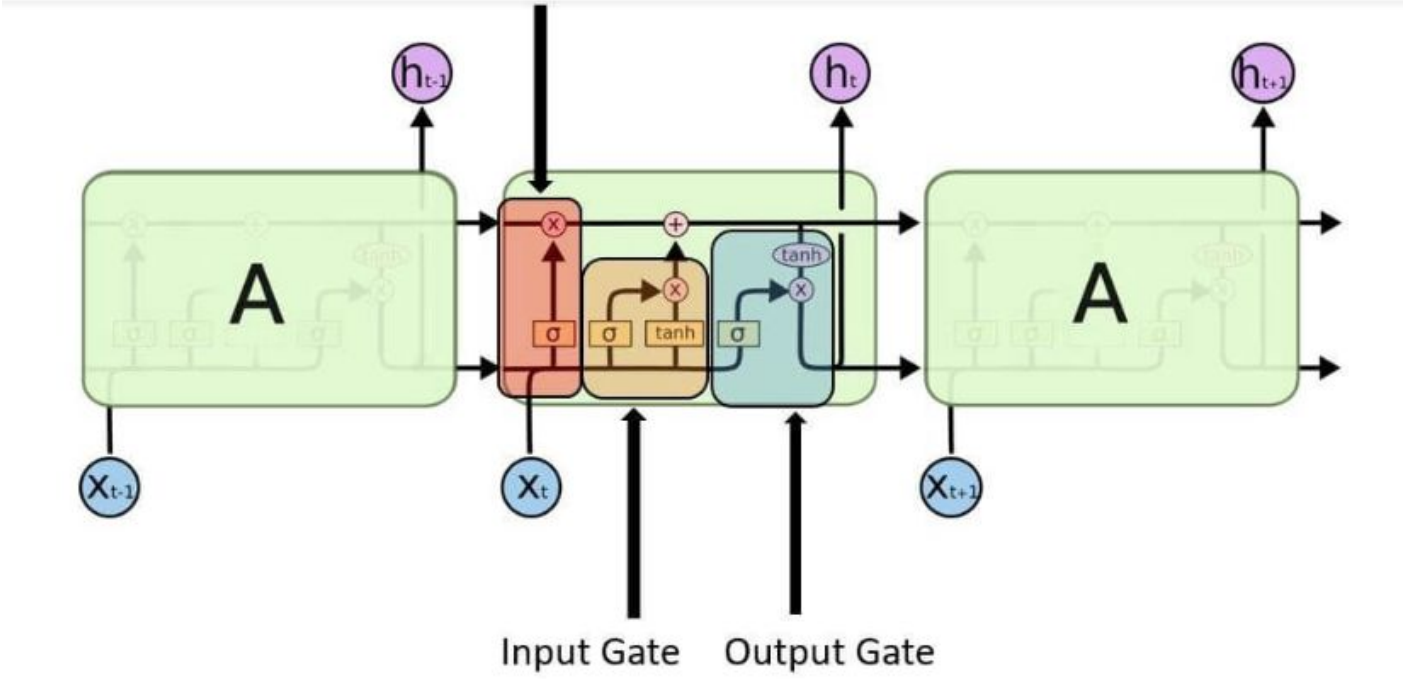
1. Forget Gate: Decides what % of Long Term memory to discard from the cell state.

2. Input Gate: Decides what new information to store in the cell state.

3. Output Gate: Decides what part of the cell state to output.



Long Short Term Memory (LSTMs)



Gated Recurrent Units (GRUs)

Gated Recurrent Units (GRUs) are simpler, having only two gates (reset gate and update gate). They combine the functionality of the forget and input gates from LSTMs into a single update gate.

They achieve this through a simpler architecture with fewer gates than LSTMs, leading to computational efficiency and ease of training.

GRUs are widely used in various applications involving sequence data, from natural language processing to time series forecasting.

Attention Mechanism

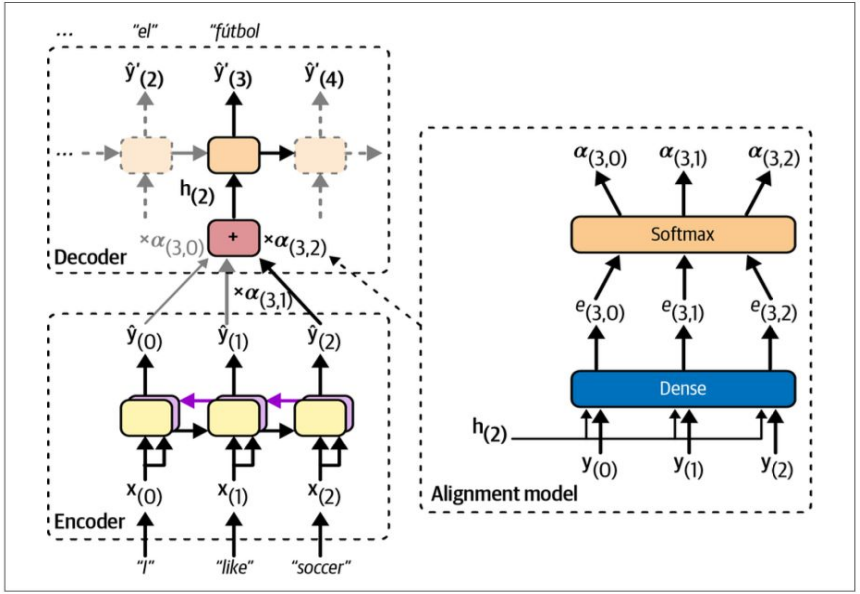
LSTMs or GRUs, still struggled to **maintain context over** long**er sequences**. e.g (30 words and above)

Resulted in a loss of
important contextual information for long-range dependencies.

Added on top of Encoder-Decoder Architecture to get Seq2Seq Models with Attention

Revolutionized text generation and machine translation

Foundation for Attention is All you need.
Paper that introduced Transformer



Attention Mechanism

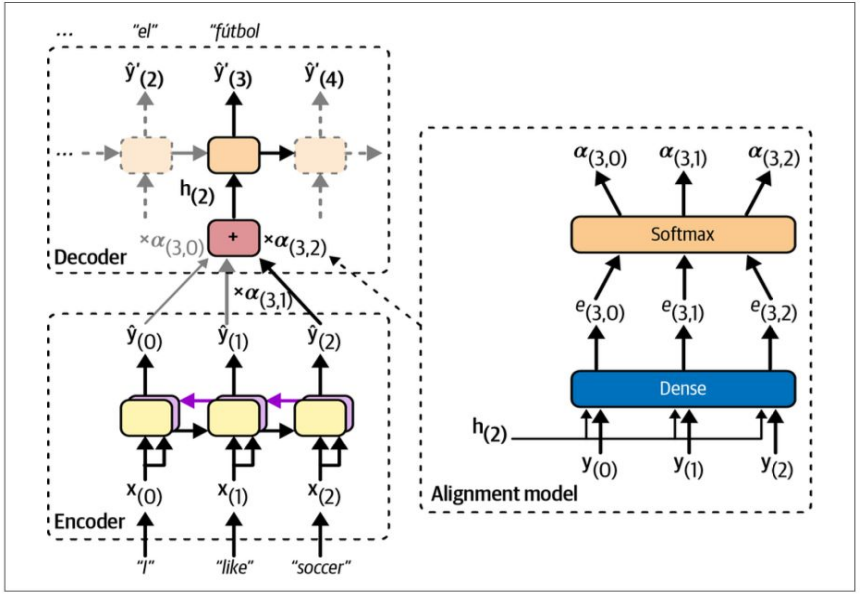
The attention mechanism allows the model to selectively focus on different parts of the input sequence when generating each token in the output sequence.

Instead of relying on a fixed-size context vector (the final hidden state of the encoder), attention provides a dynamic context that considers the entire input sequence.

Added Attention layer (alignment model) on Encoder-Decoder.

Allows the decoder to focus on the appropriate words at each time step.

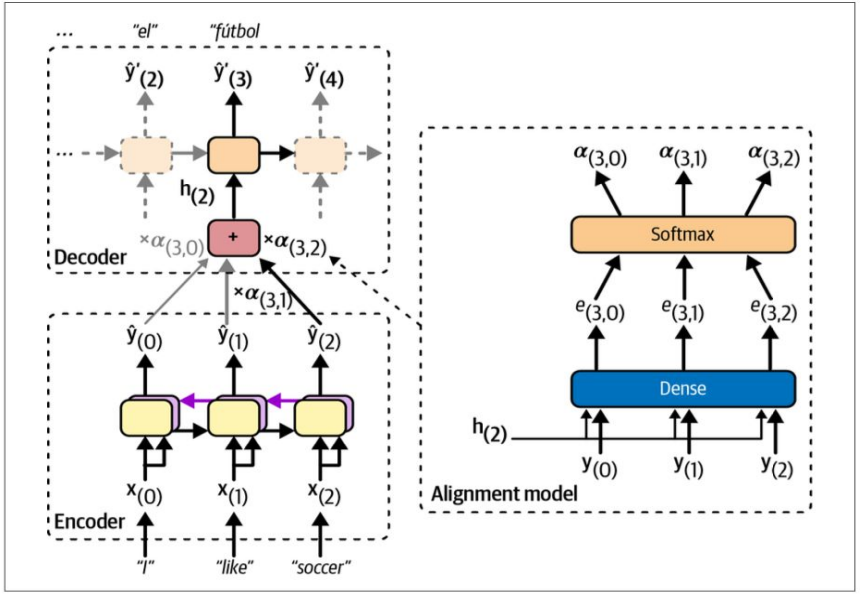
e.g focus on “soccer” when generating “futbol” i.e put more weight on alpha(3,2) at time step 2.



Attention Mechanism

The weight α(t,i) is the weight
of the ith encoder output at the tth decoder time step.

e.g, if the weight α(3,2) is much larger than the weights α(3,0) and α(3,1), then the decoder will pay much more attention to the encoder's output for word #2 ("soccer") than to the other two outputs, at least at this time step.



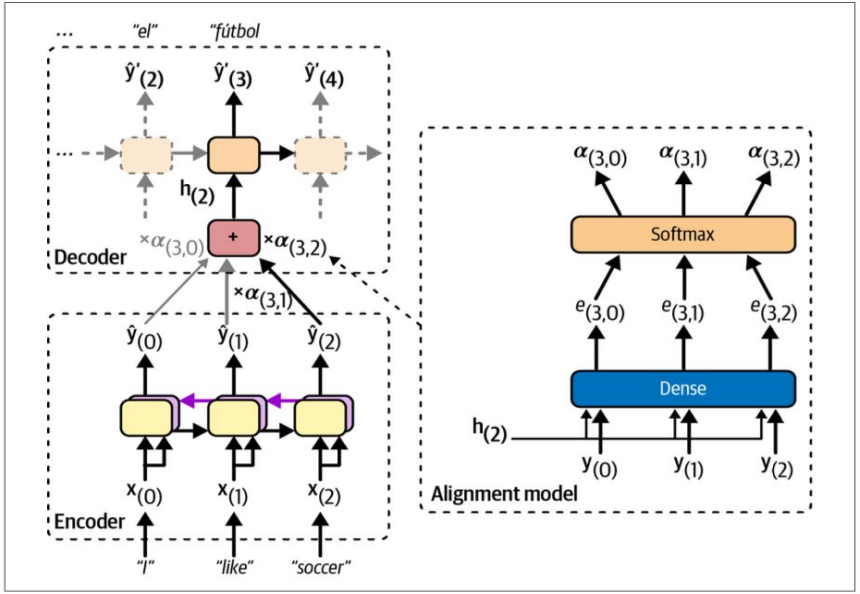
Attention Mechanism

Dense layer composed of a single neuron

a softmax layer to get a final weight for each encoder output.

All the weights for a given decoder time step add up to 1.

Since it concatenates the encoder output with the decoder's previous hidden state, it is sometimes called concatenative attention (or additive attention)



Encoder-Decoder Architecture

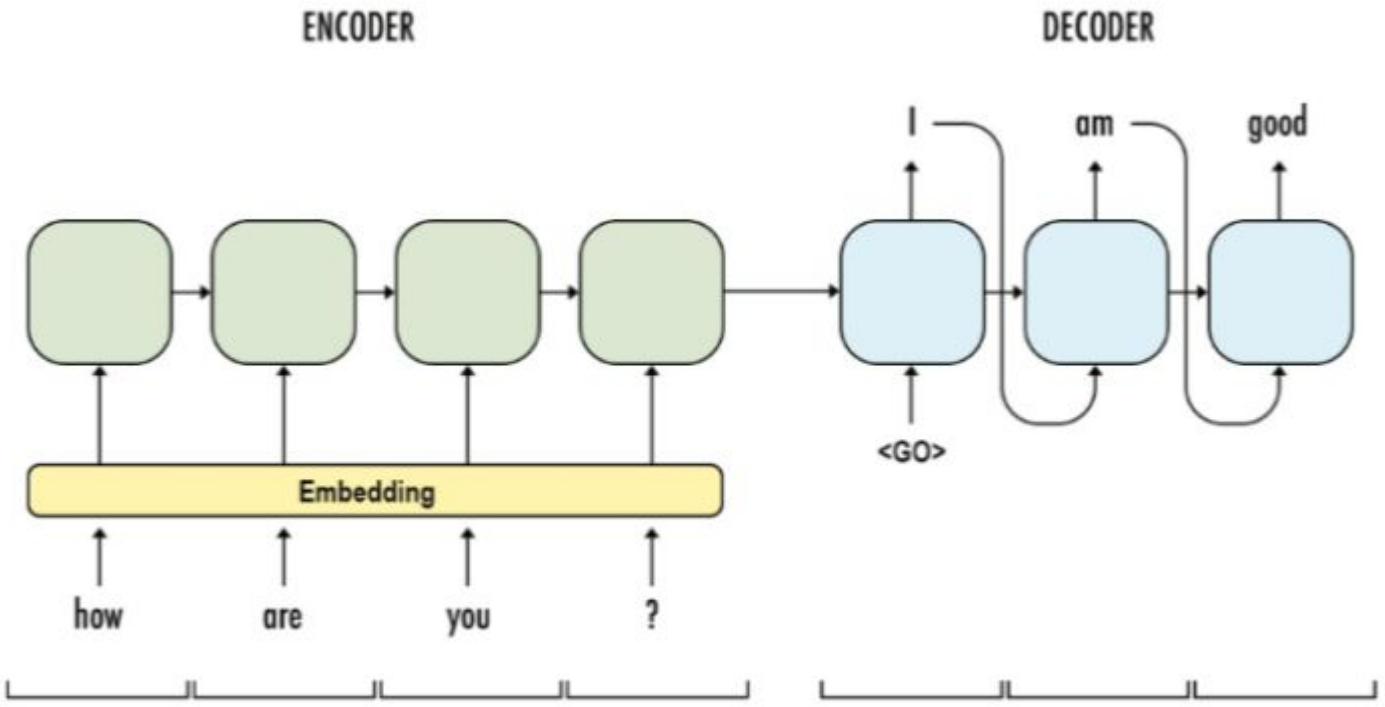
Implements Sequence to Sequence modelling.(Seq2Seq)
Allows RNNs to take a sequence as input and generate sequence as output
Popular in text generation, machine translation, text summarization tasks

1. Encoder-Decoder Framework:
   - Encoder: Processes the input sequence and compresses its information into a context vector
   - Decoder: Generates the output sequence from the context vector provided by the encoder.

2. Components:
   - Encoder RNN: Takes the input sequence and generates a hidden state for each token. Typically, LSTM (Long Short-Term Memory) or GRU (Gated Recurrent Unit) cells are used.
   - Context Vector: The final hidden state from the encoder serves as the context vector summarizing the input sequence.
   - Decoder RNN: Uses the context vector to generate the output sequence, one token at a time.

3. Attention Mechanism:
   - Helps the decoder focus on relevant parts of the input sequence during each step of the output generation, improving performance especially for long sequences.

Encoder-Decoder Architecture



Transformer

The Transformer architecture follows an encoder-decoder structure but does not rely on recurrence and convolutions in order to generate an output.

The encoder’s generate word representations (word embedding) of that aims to perfectly captures the meaning of

the word, in the context of the sentence.

The decoder’s role is to gradually transform each word representation (word embedding) into a word representation of the next word in the generation.

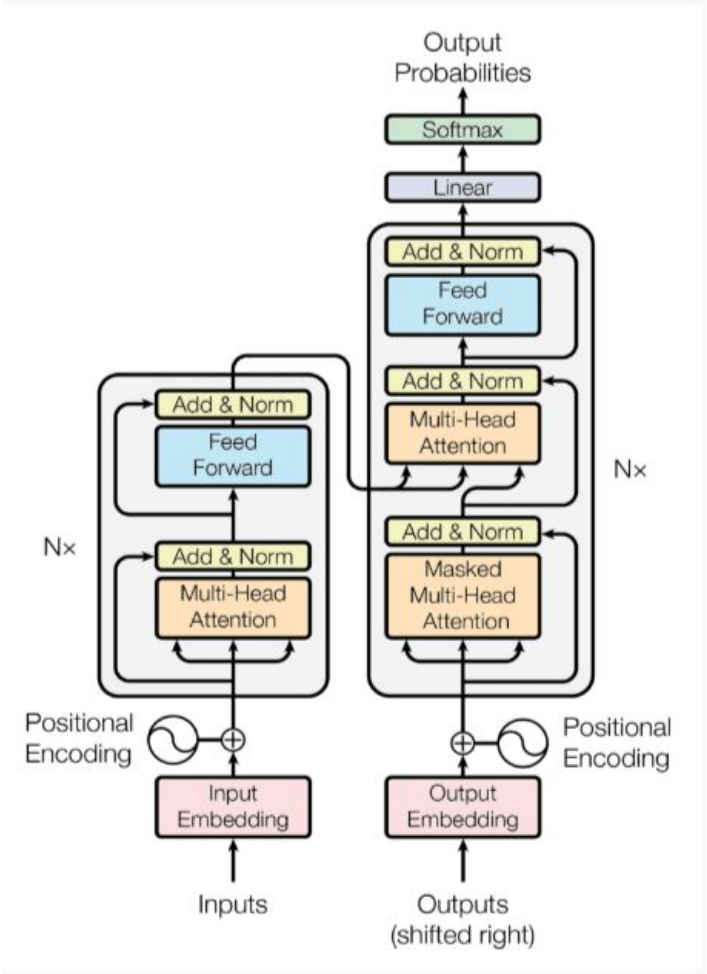
Transformer

Embedded layer : to learn word representation

Positional encoding: to pay attention to positions

of words in a sentence

Multi head Attention : to attend to important word



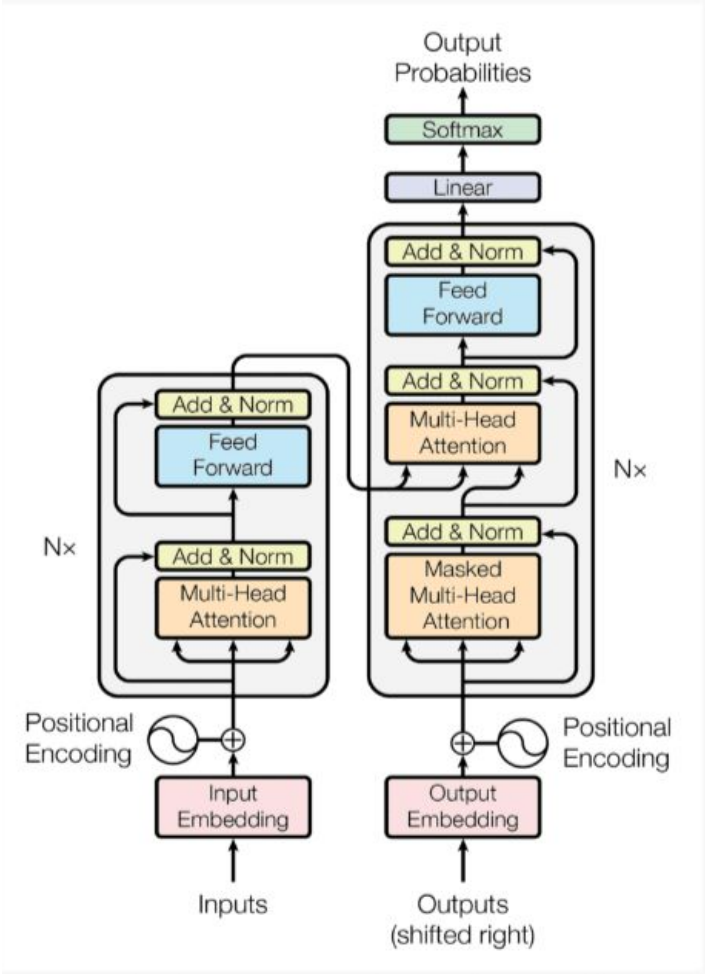
The encoder-decoder structure of the Transformer architecture Taken from "Attention Is All You Need"

Transformer

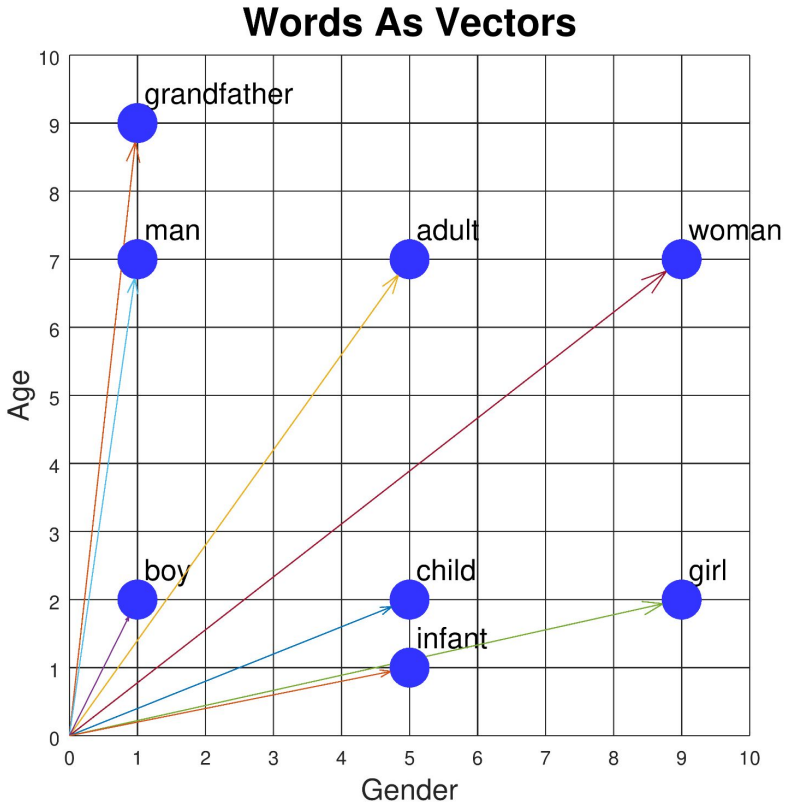
The encoder and the decoder contain modules that are

stacked 6 times.

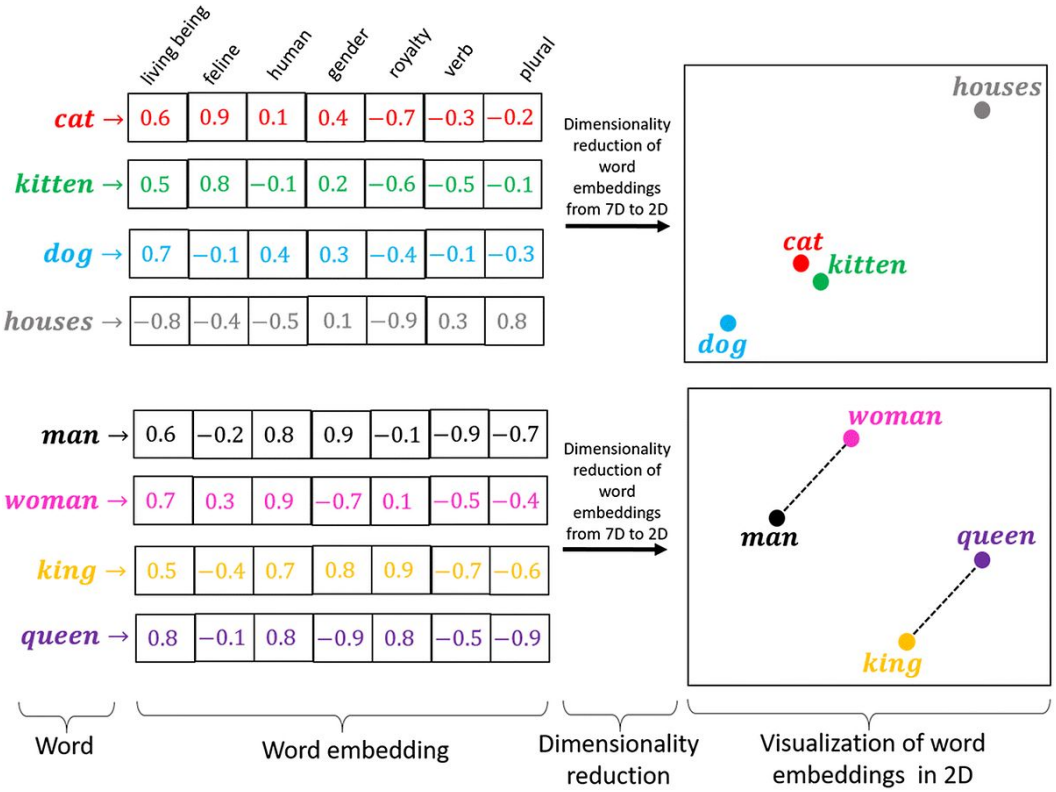
1. Embeddings
2. Self-Attention Mechanism
3. Layer Normalization
4. Feed-Forward Network
5. Residual Connections:
6. Output Layer



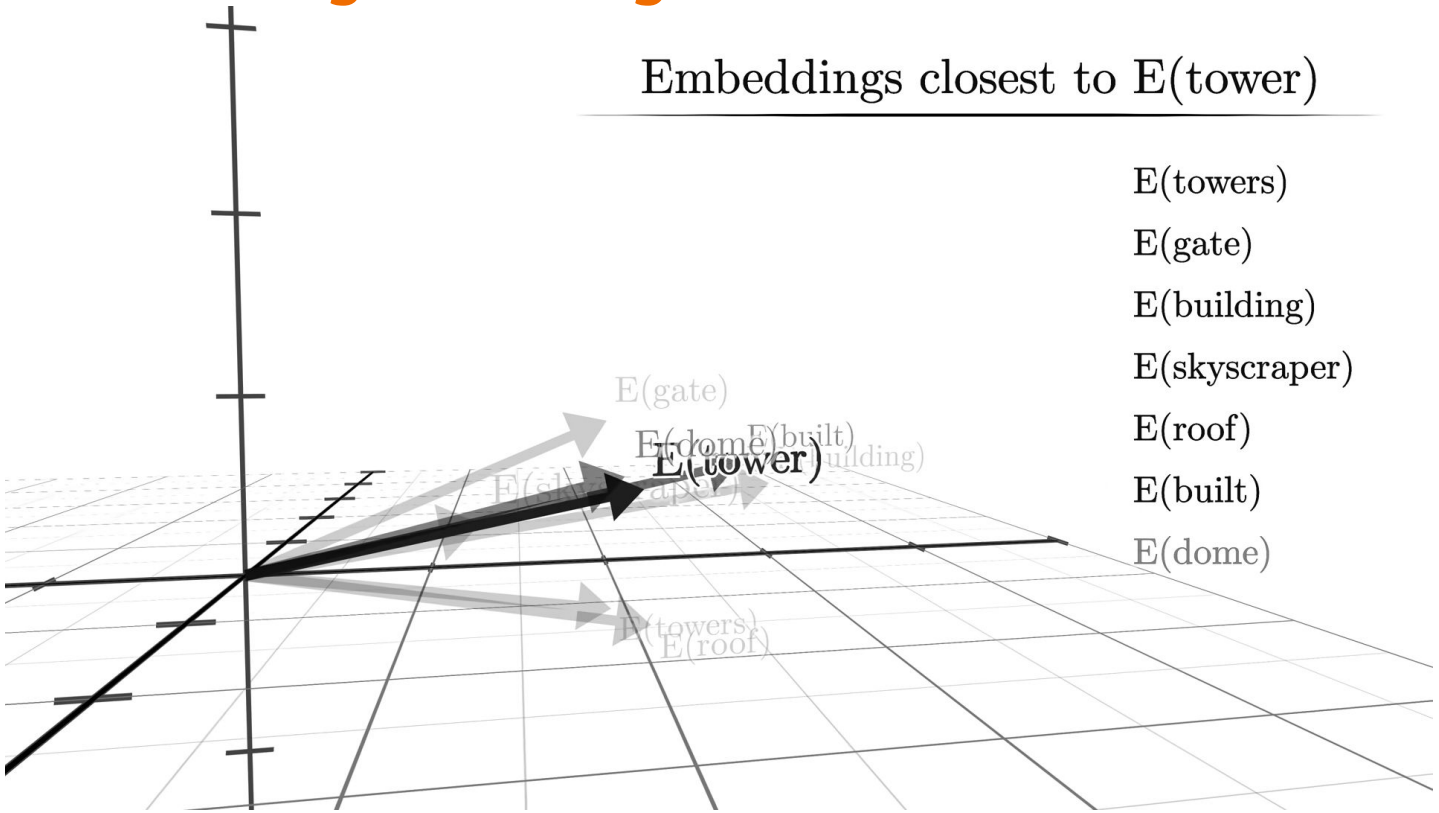
Word Embedding



Word Embedding: Representation

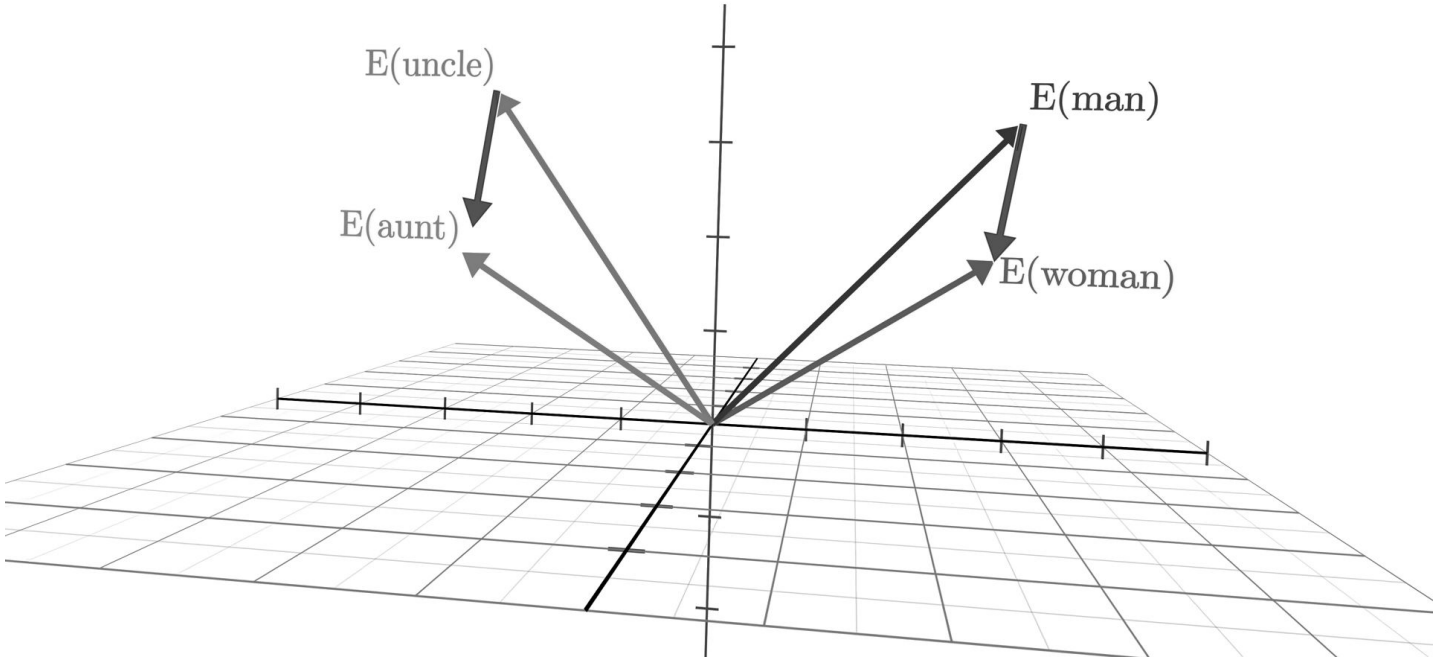


Word Embedding: Meaning



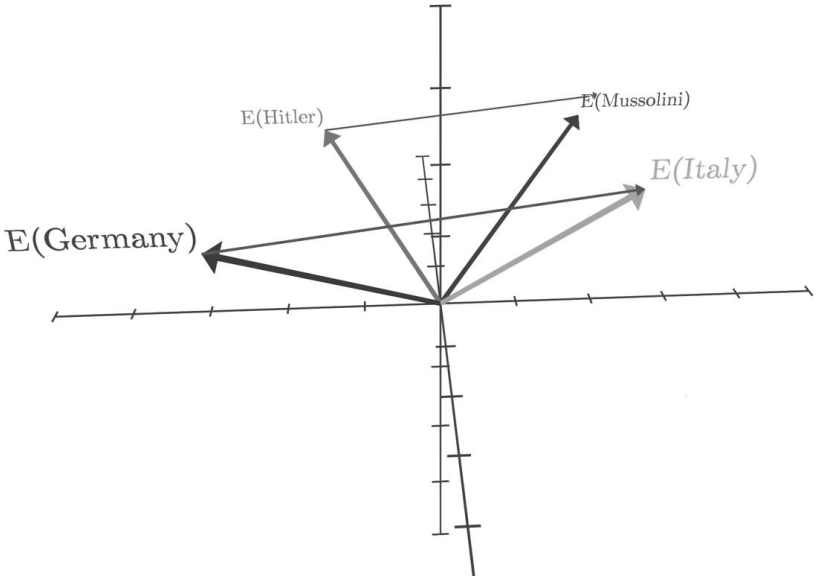
Word Embedding: Meaning

$$E(\text{aunt}) - E(\text{uncle}) \approx E(\text{woman}) - E(\text{man})$$



Word Embedding: Meaning

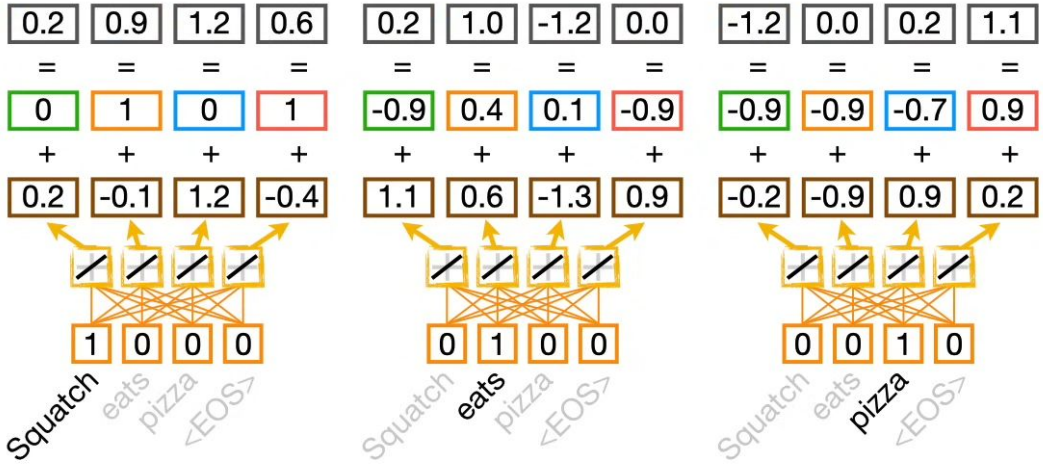
$$\mathrm{E}(\text{Hitler}) + \mathrm{E}(\text{Italy}) - \mathrm{E}(\text{Germany}) \approx \mathrm{E}(\text{Mussolini})$$

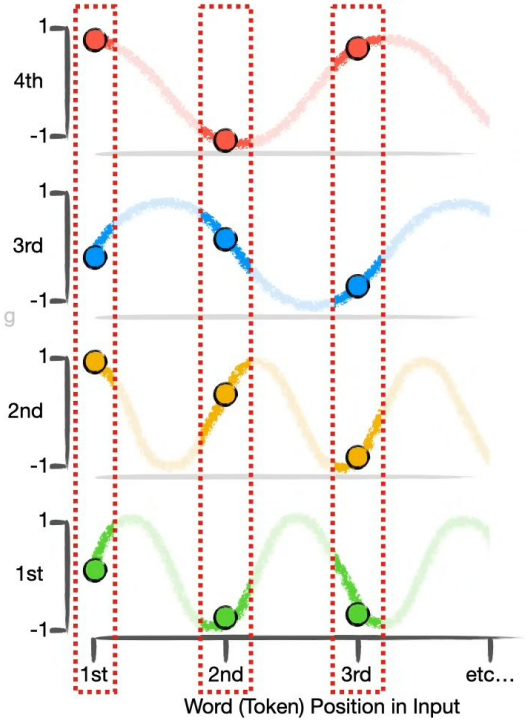


Word Embedding:

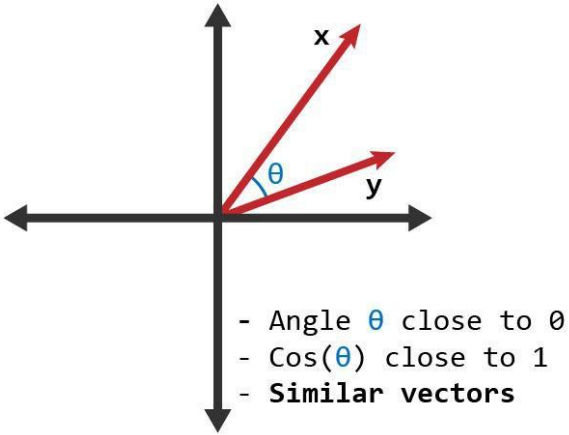
Word embeddings transform the input text (words or tokens) into dense vectors of fixed size. These vectors capture the semantic meanings and relationships between words. The embeddings serve as the input to the Transformer model, enabling it to understand and process text data.

Positional Encoding

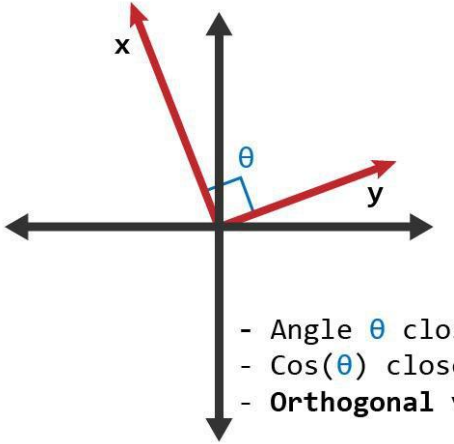




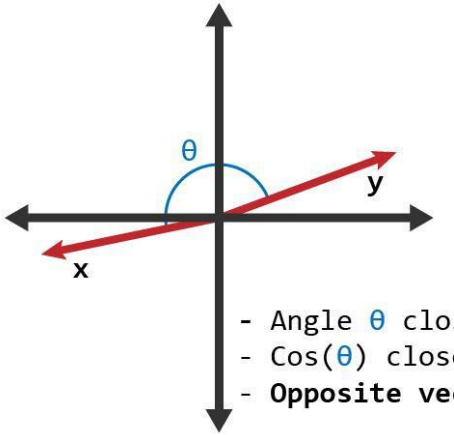
Cosine Similarity



- Angle θ close to 0
- Cos(θ) close to 1
- **Similar vectors**



- Angle θ close to 90
- Cos(θ) close to 0
- **Orthogonal vectors**



- Angle θ close to 180
- Cos(θ) close to -1
- **Opposite vectors**

Transformer: The Encoder

The encoder’s multi-head attention layer updates each word representation by

attending to (i.e., paying attention to) all other words in the same sentence.

That’s where the vague representation of the word “like” becomes a richer

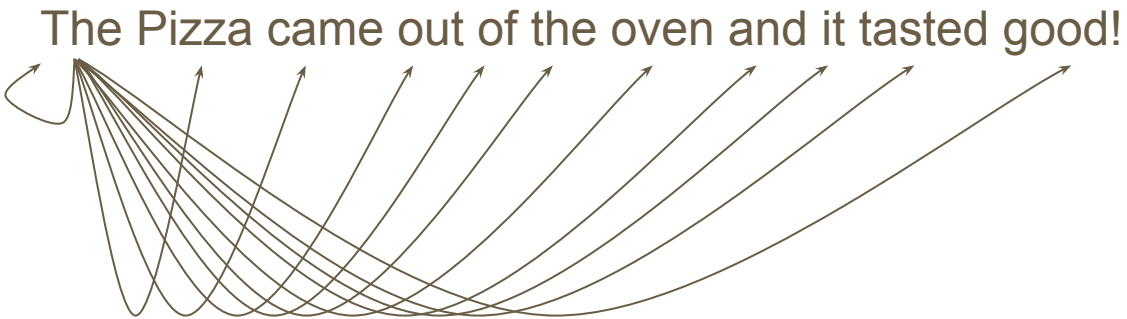
and more accurate representation, capturing its precise meaning in the given

sentence

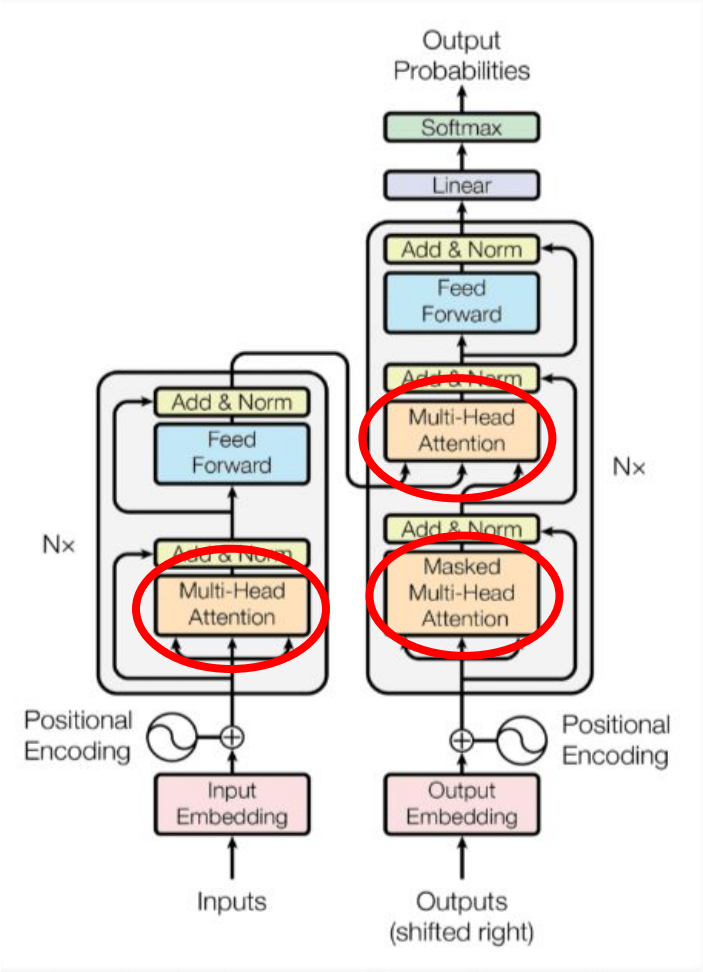
Transformer: Self Attention

The **Pizza** came out of the **oven** and **it** tasted good!

Transformer: Self Attention

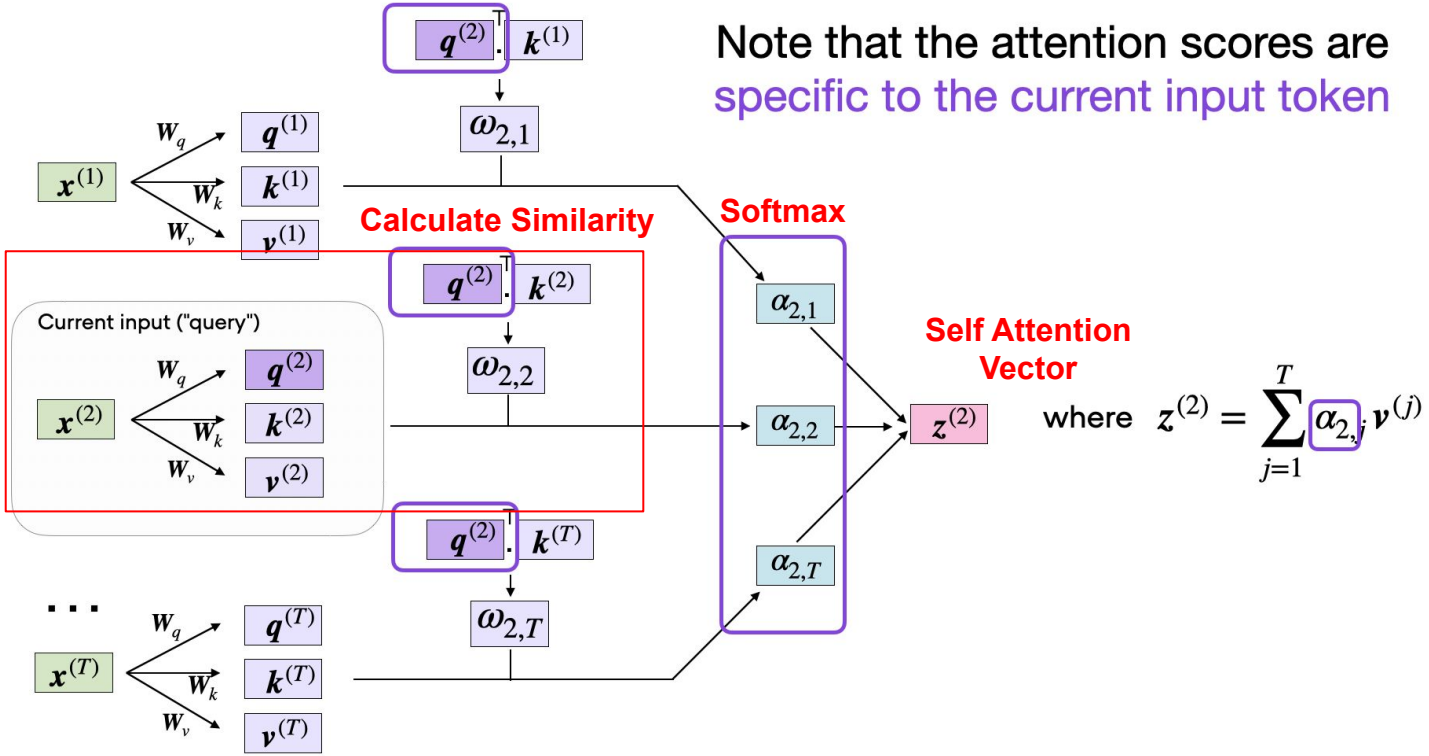


Transformer

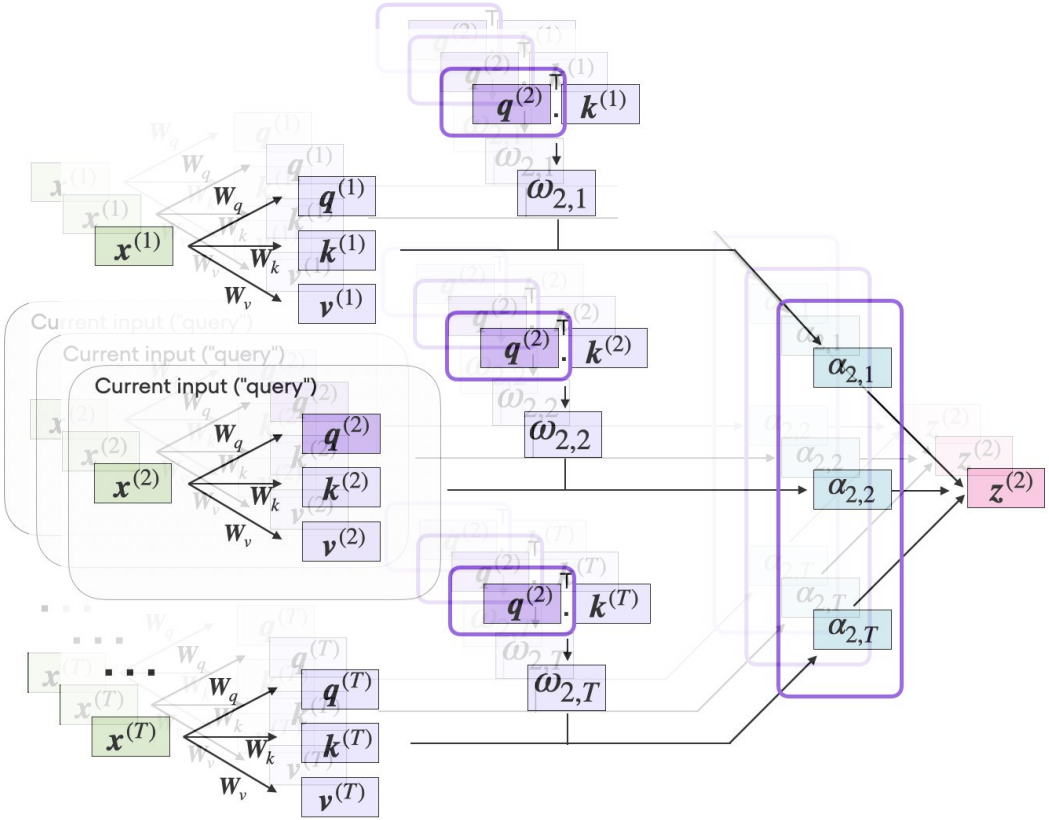


The encoder-decoder structure of the Transformer architecture Taken from "Attention Is All You Need"

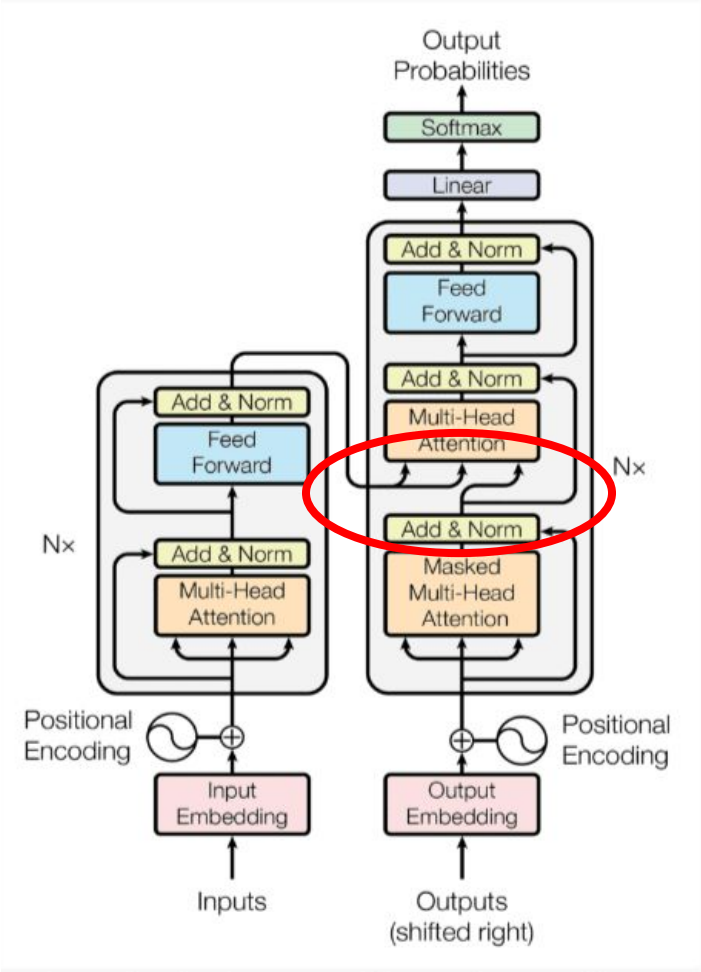
Transformer: Single-Head Attention



Transformer: Multi-Head Attention

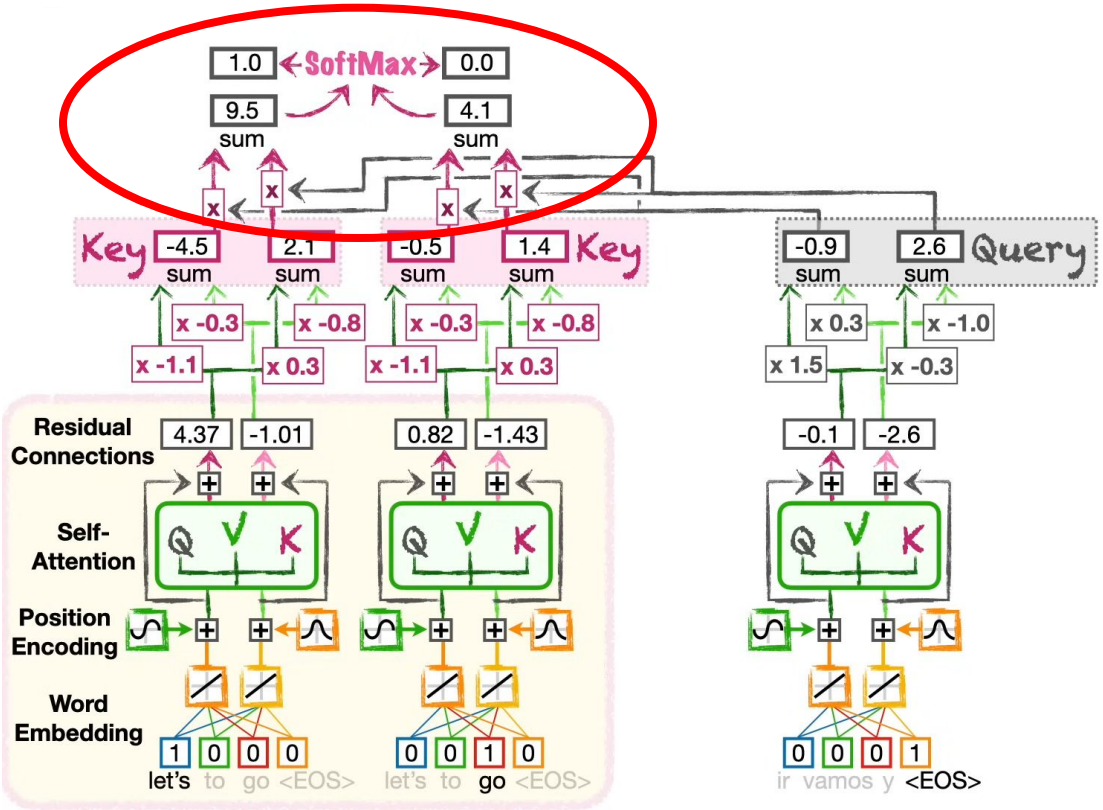


Transformer

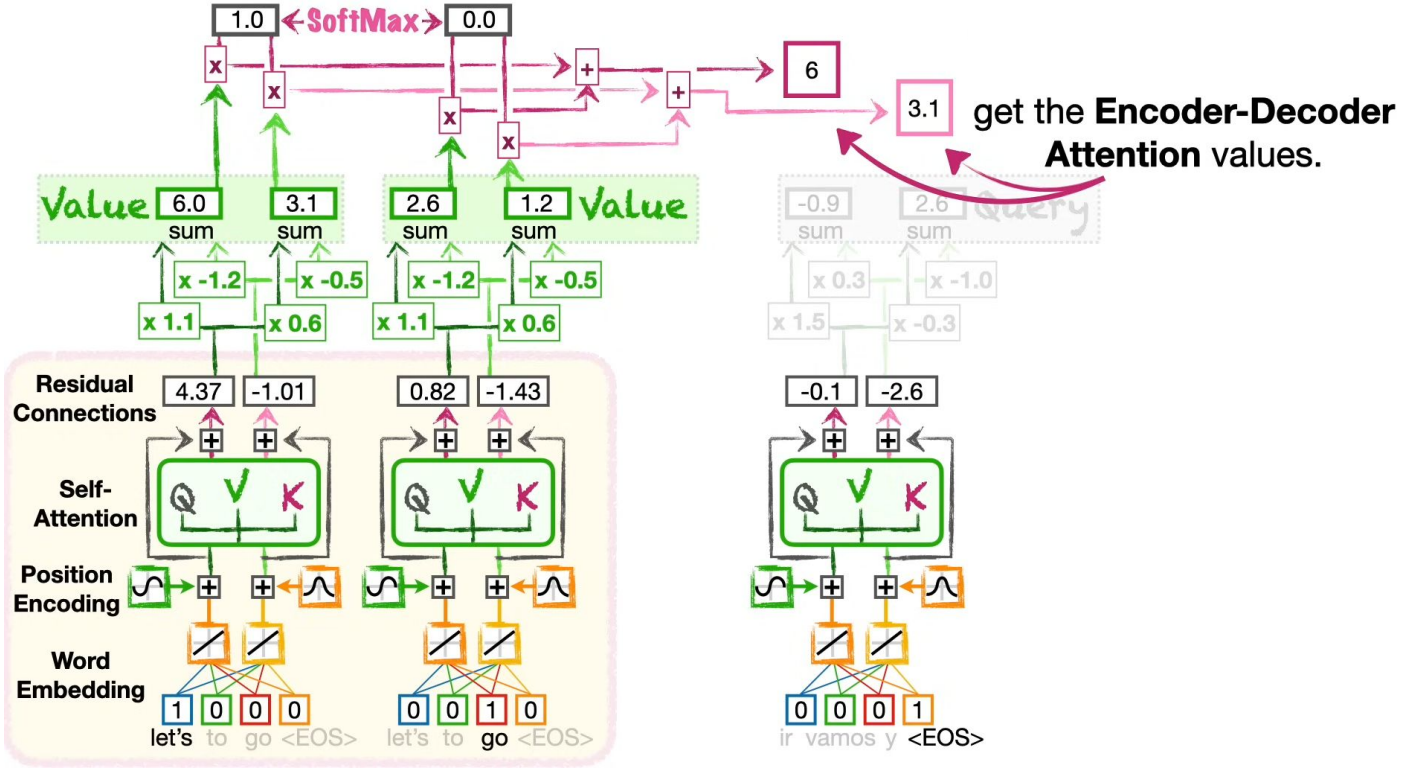


The encoder-decoder structure of the Transformer architecture Taken from "Attention Is All You Need"

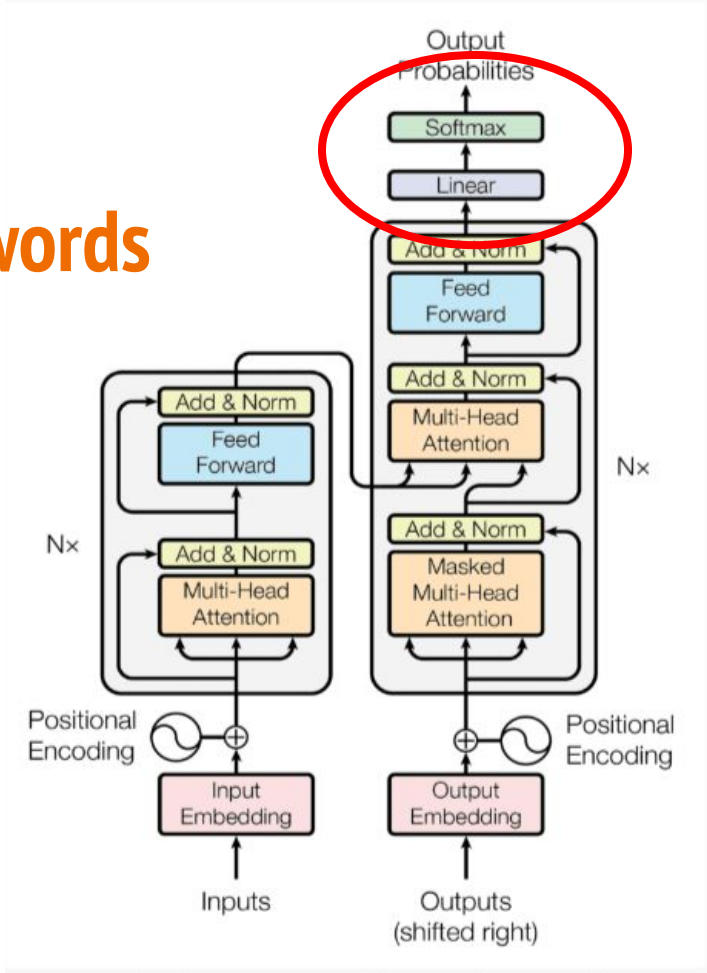
Transformer: Encoder-Decoder Attention



Transformer: Encoder-Decoder Attention

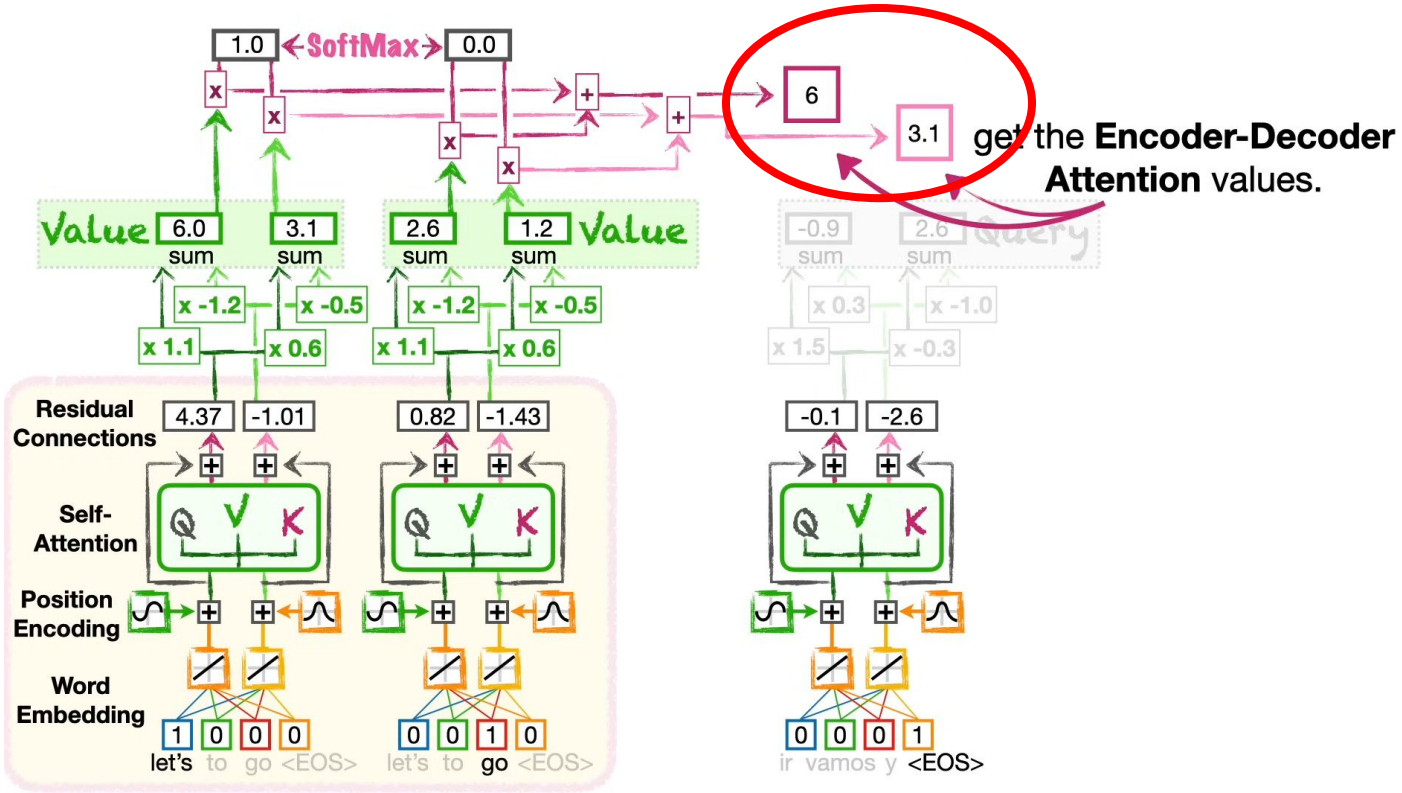


Transformer: Decoding into words

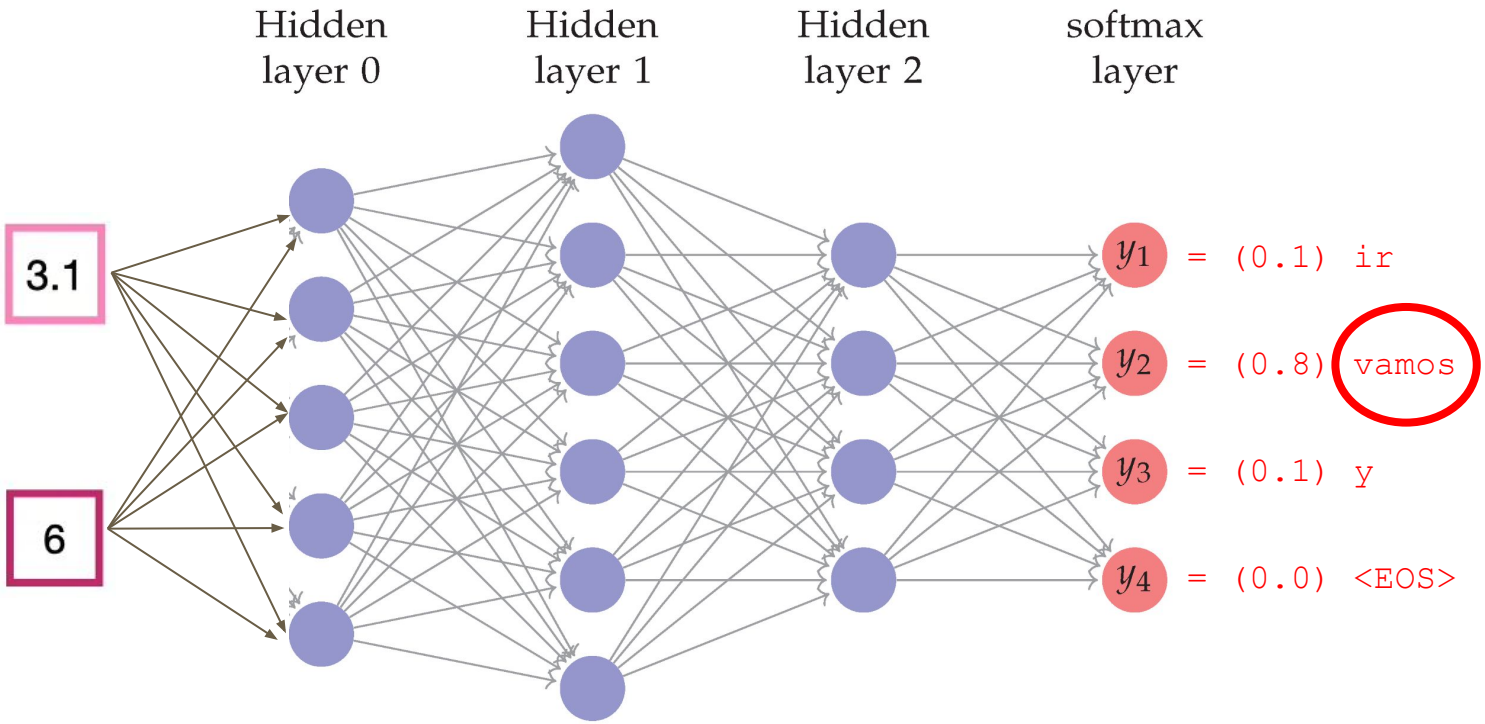


The encoder-decoder structure of the Transformer architecture Taken from "Attention Is All You Need"

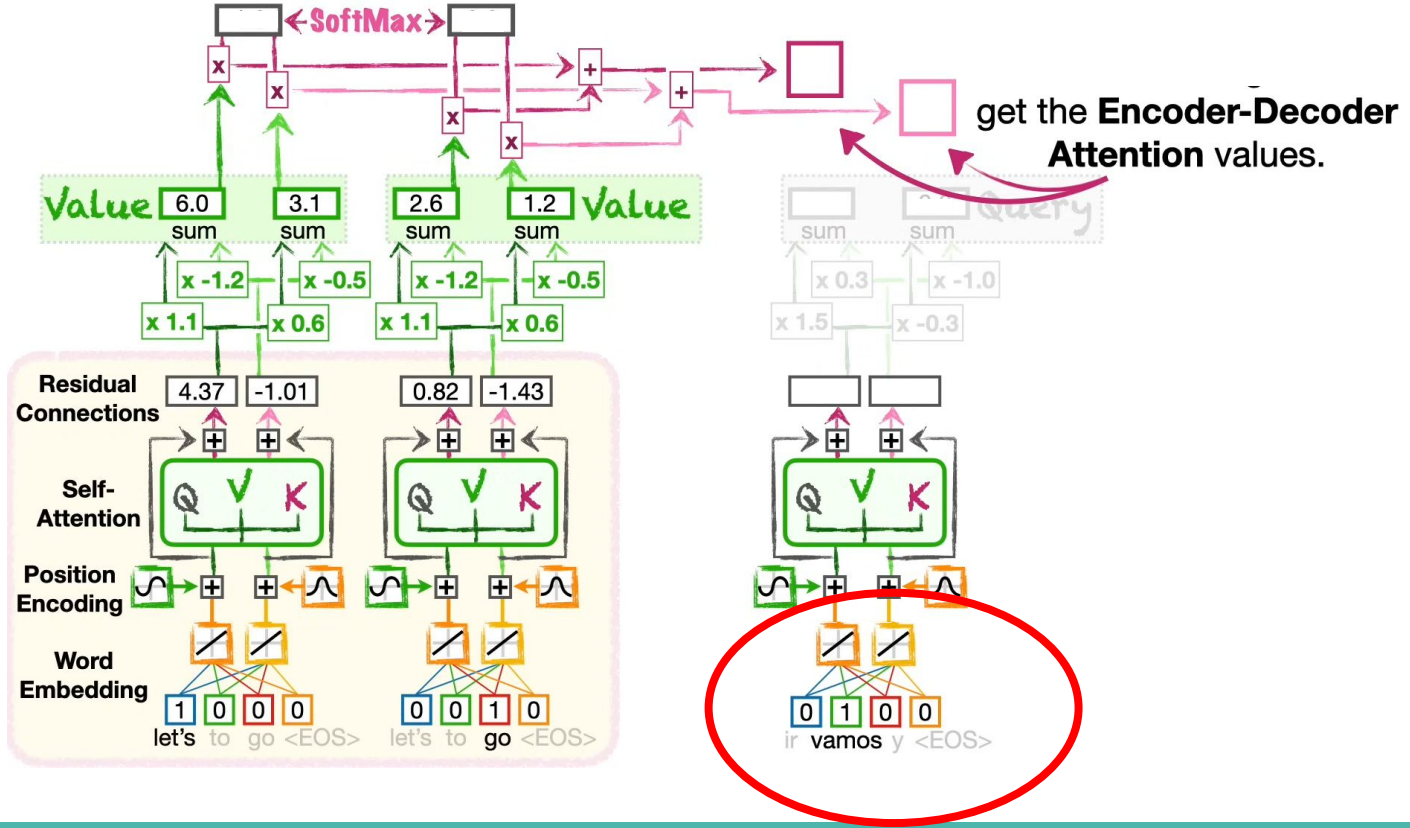
Transformer: Decoding into words



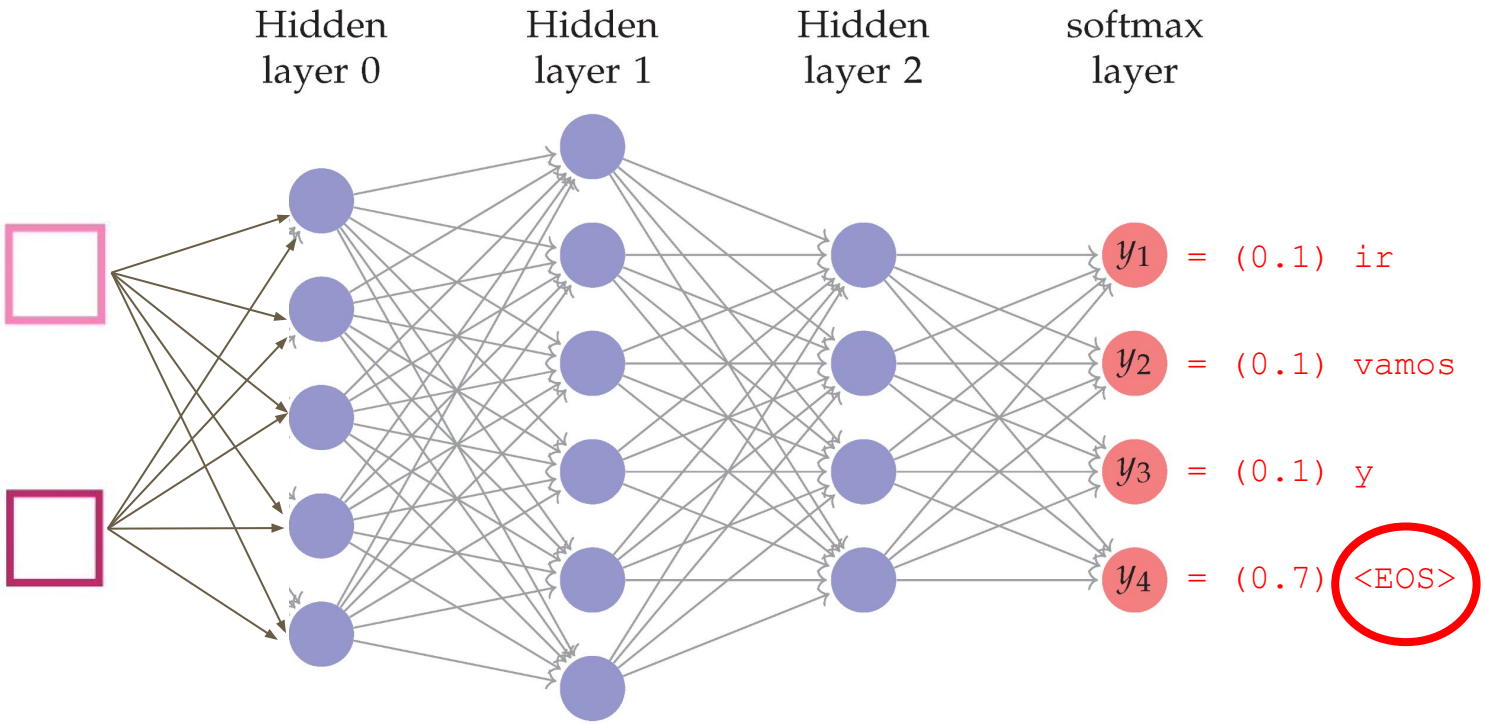
Transformer: Decoding into words



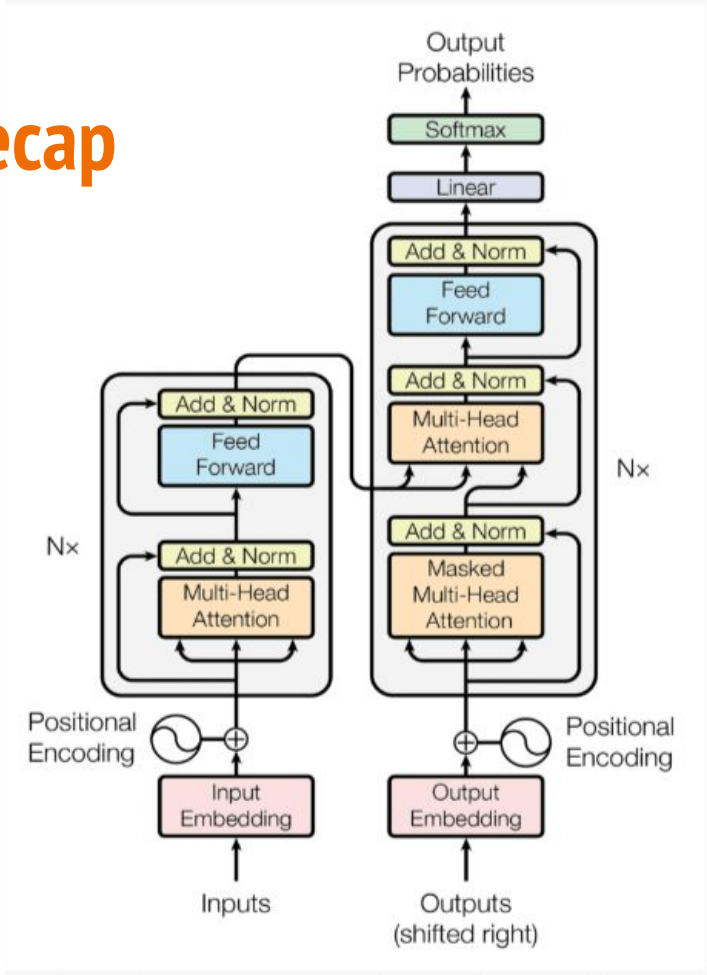
Transformer: Decoding into words - Repeat with previous output



Transformer: Decoding into words - Until getting <EOS>



Transformer: Recap



The encoder-decoder structure of the Transformer architecture Taken from "Attention Is All You Need"

Thank You!