Visual Attention

Visual Attention

- A model generated the caption “A woman is throwing a frisbee in a park” for the below image:

Visual Attention

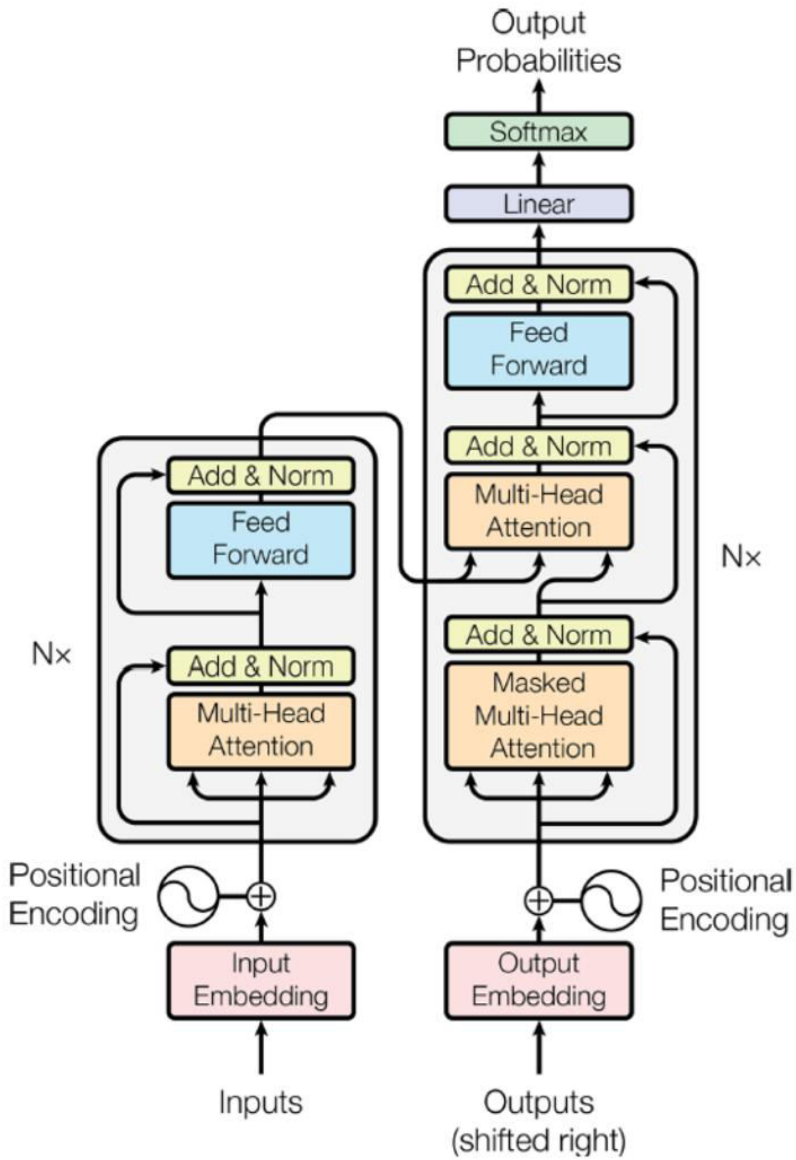
- CNN processes the image
- Outputs feature maps
- Then decoder RNN equipped with attention mechanism
    - Generates the caption, one word at a time

At each decoder time step (each word), the decoder uses the attention model to focus on just the right part of the image

The Transformer Architecture

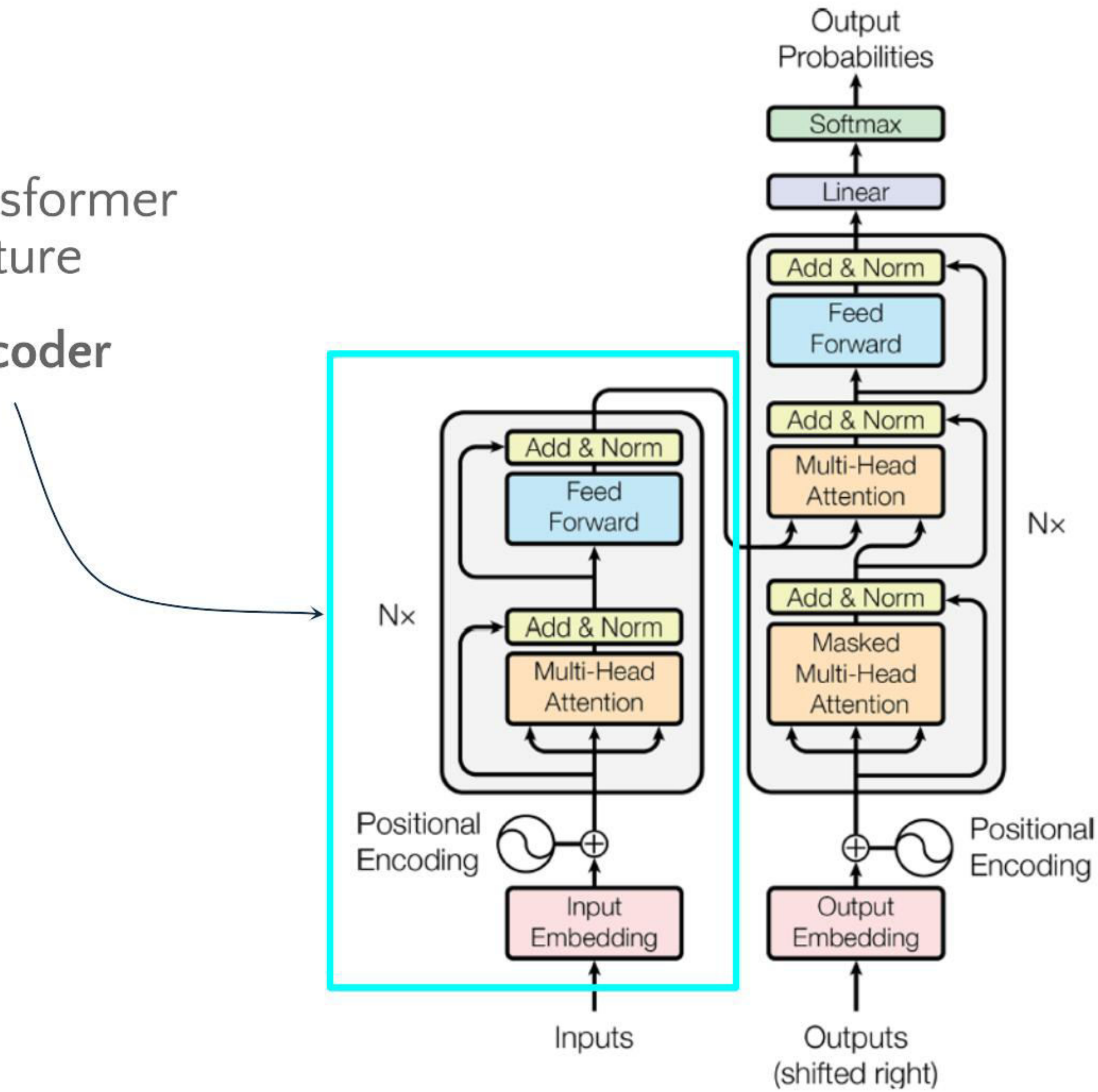
- In 2017, a team of Google researchers created an architecture called the **Transformer**
- It used only Attention Mechanism, no RNN or CNN to accompany it

The Transformer Architecture



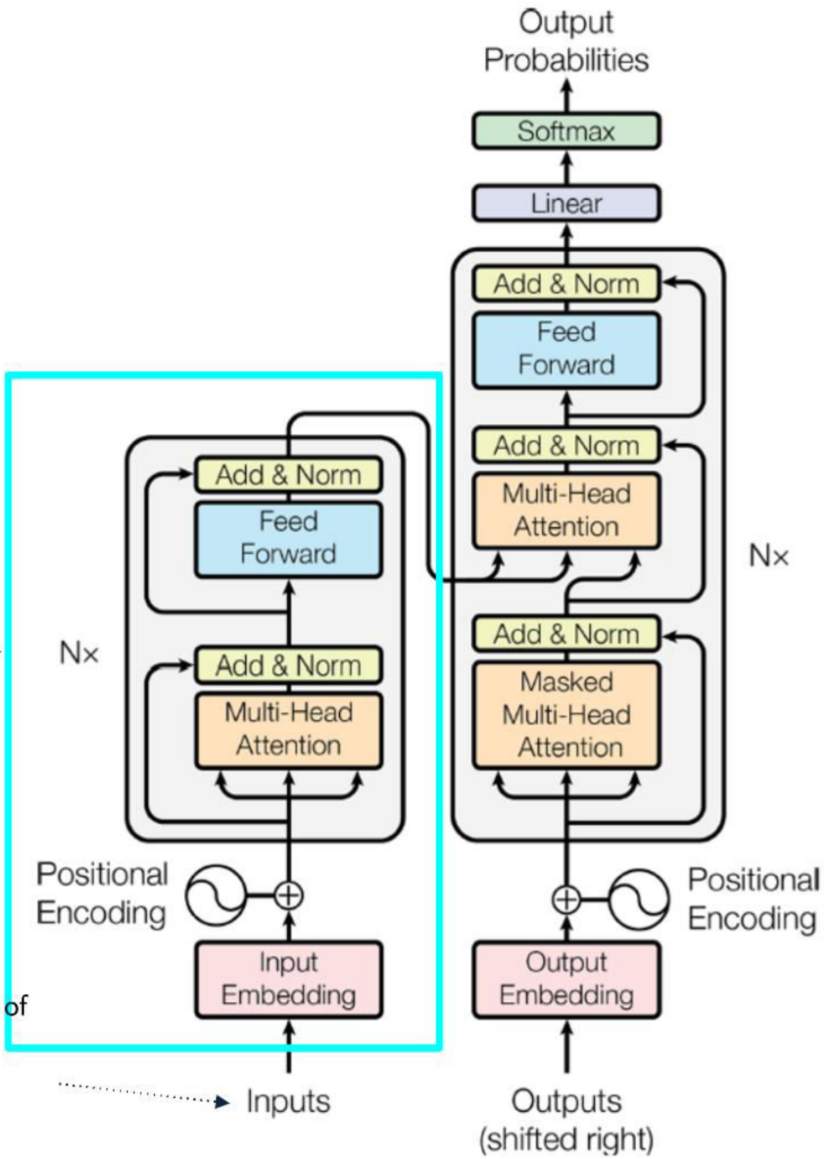
The Transformer Architecture

Encoder



The Transformer Architecture

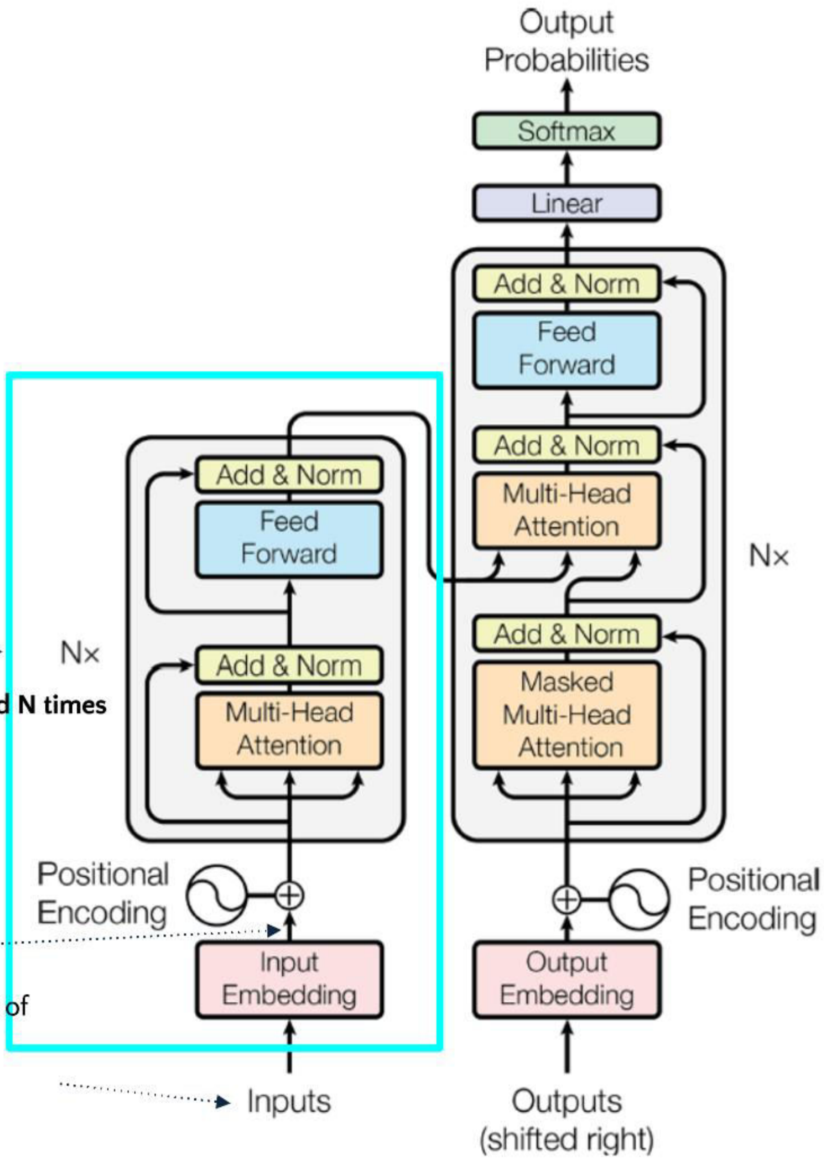
Encoder



A batch of sentences represented as sequences of word IDs.
Shape: **[batch size, max input sentence length]**

The Transformer
Architecture

Encoder

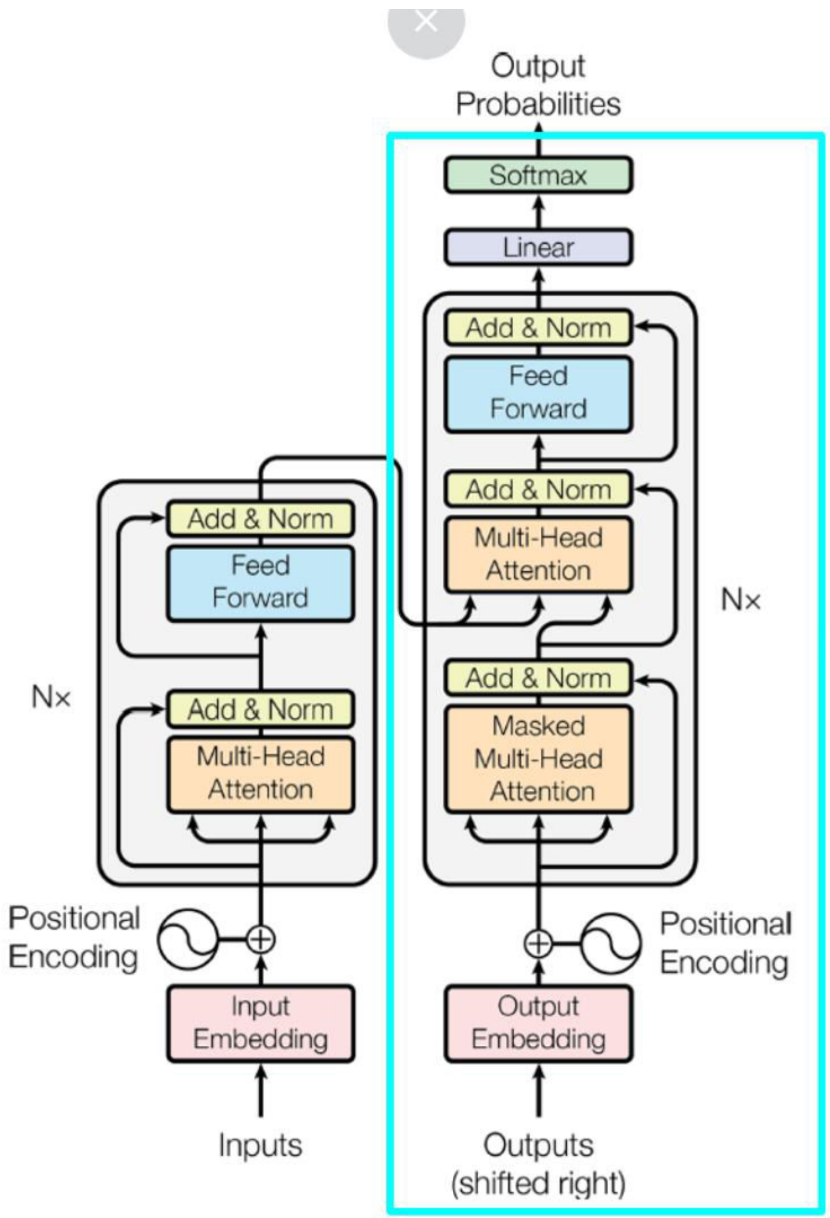


Stacked N times
N = 6

[batch size, max input sentence length, 512]

A batch of sentences represented as sequences of word IDs.
Shape: **[batch size, max input sentence length]**

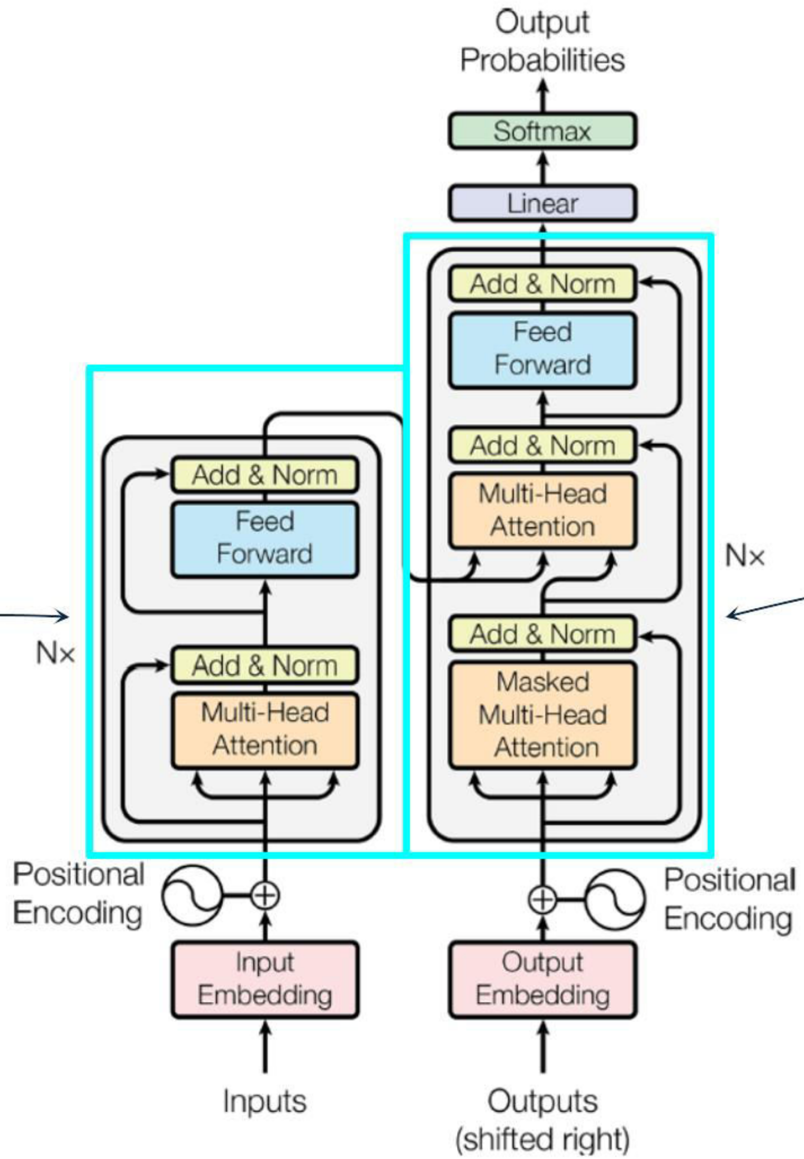
The Transformer Architecture



Decoder

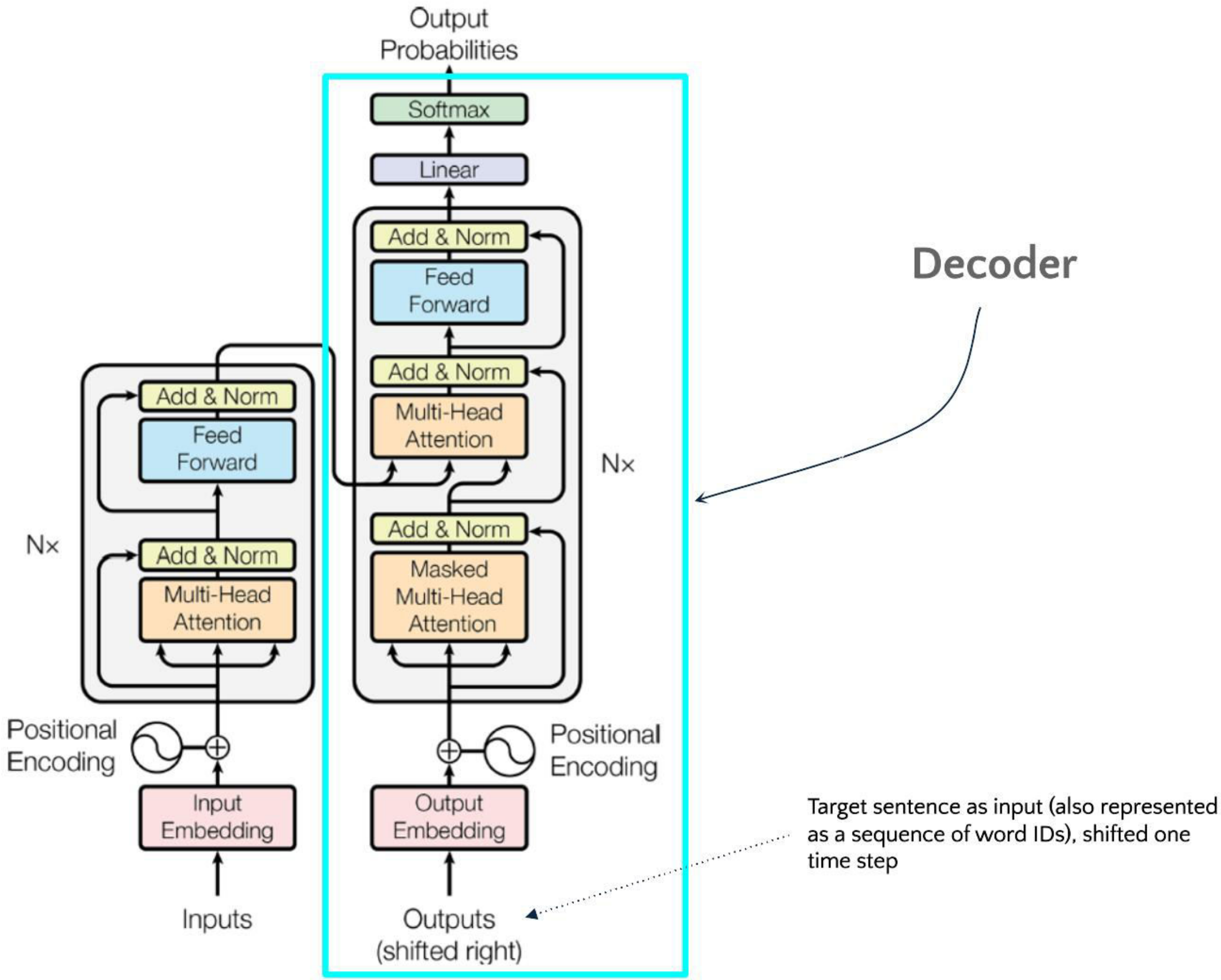
The Transformer Architecture

Encoder

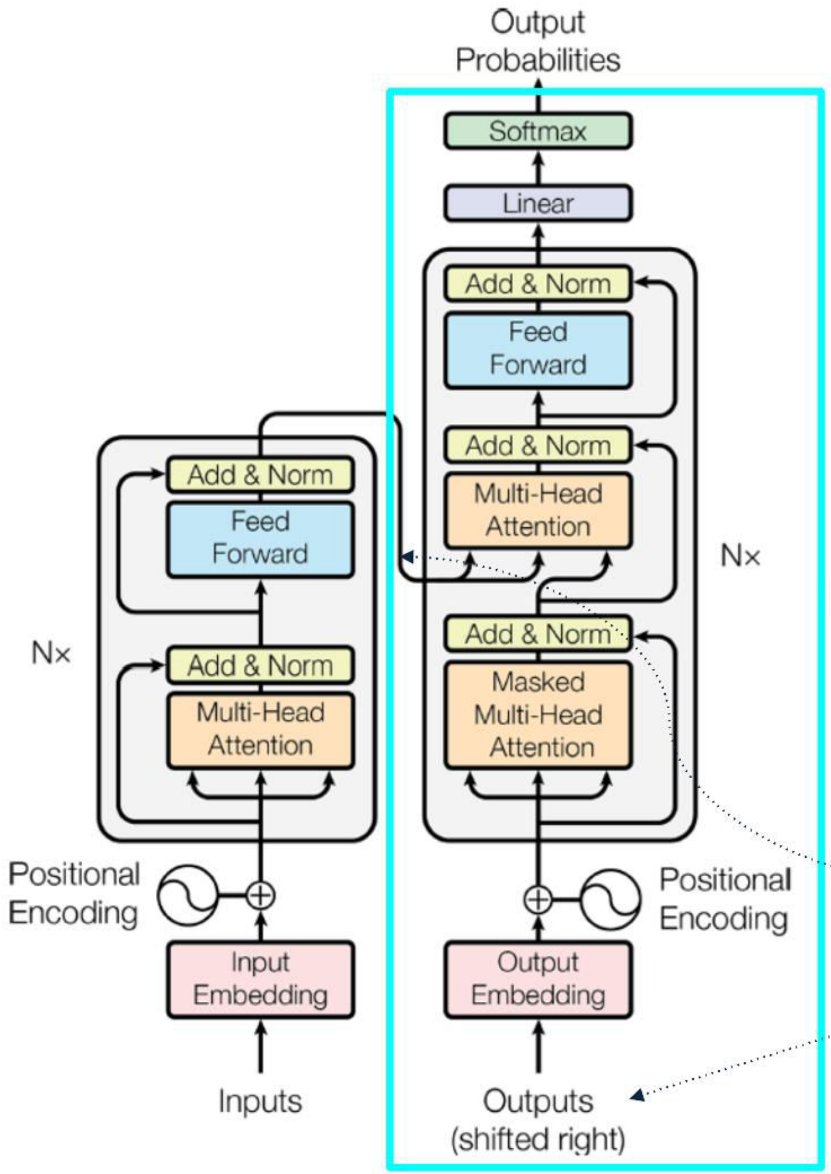


Decoder

The Transformer Architecture



The Transformer Architecture

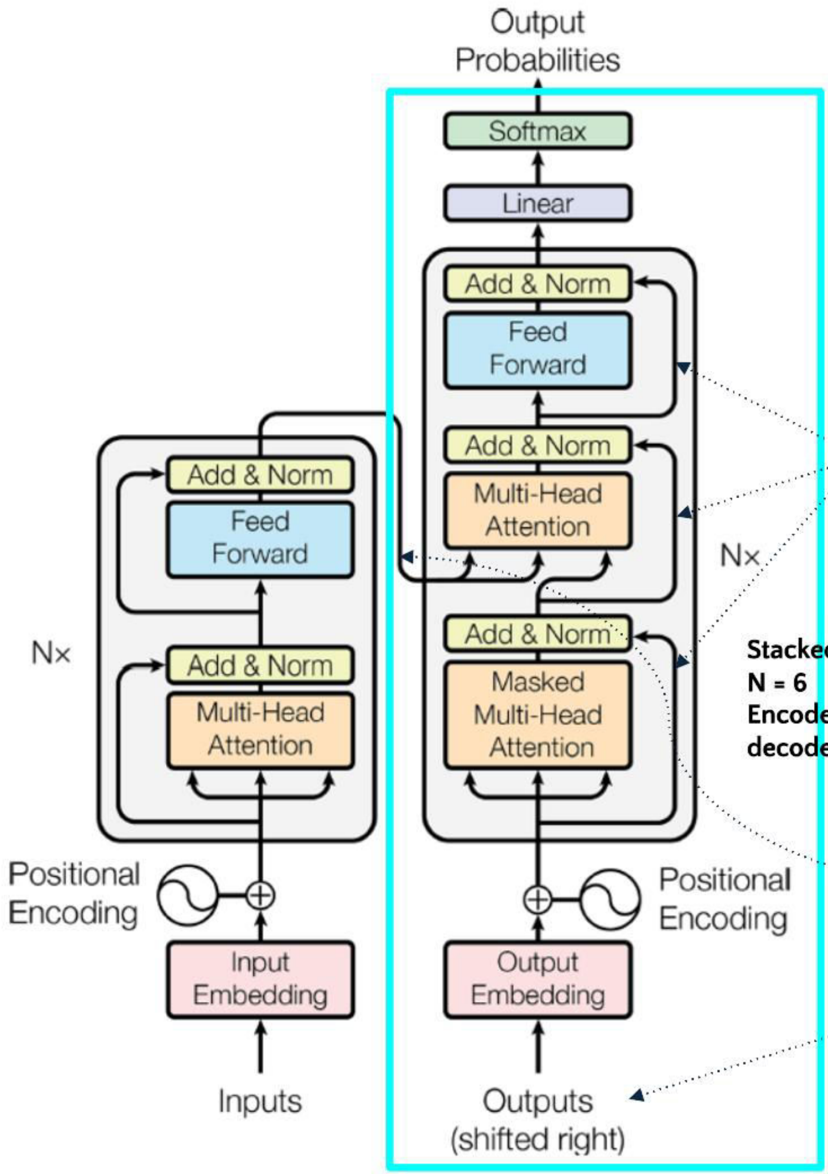


Decoder

Receives input from encoder too.

Target sentence as input (also represented as a sequence of word IDs), shifted one time step

The Transformer Architecture



Decoder

Skip Conn.

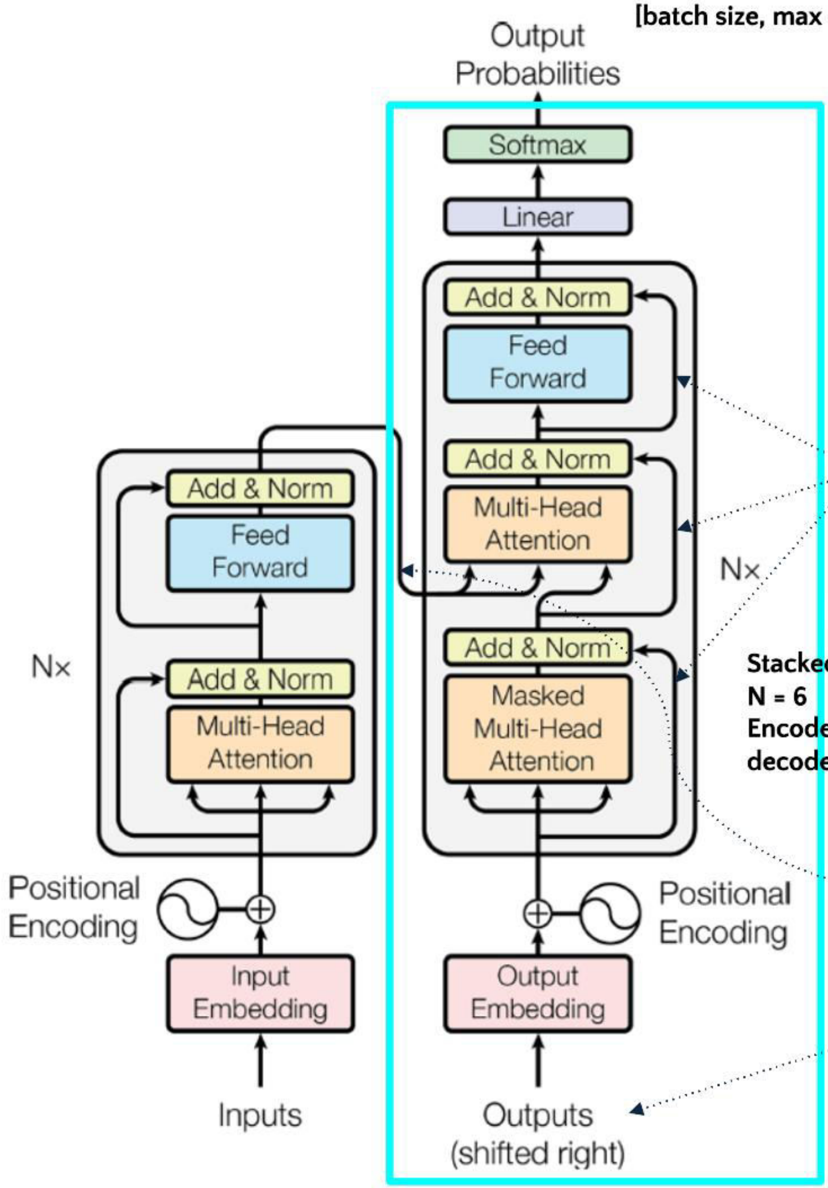
Stacked N times
N = 6
Encoder stack's final outputs are fed to the decoder at each of these N levels.

Receives input from encoder too.

Target sentence as input (also represented as a sequence of word IDs), shifted one time step

The Transformer Architecture

[batch size, max output sentence length, vocabulary length]



Decoder

Skip Conn.

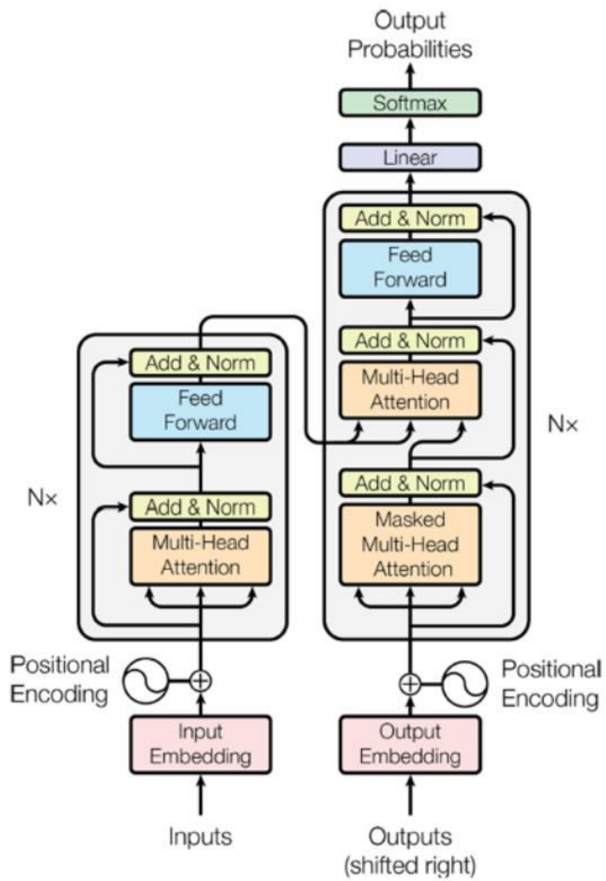
Stacked N times
N = 6
Encoder stack's final outputs are fed to the decoder at each of these N levels.

Receives input from encoder too.

Target sentence as input (also represented as a sequence of word IDs), shifted one time step

The Transformer Architecture (More)

- Two embedding layers
- 5 × N skip connections
- Each of them followed by a layer normalization layer
- 2 × N "Feed Forward" – 2 dense layers each
  - First one using the ReLU
  - second with no activation function
- The output layer is a dense layer using the softmax
- All of these layers are time-distributed.
  - So, each word is treated independently



The Transformer Architecture

- We are familiar with most components, except 2 of them:
  - Multi-Head Attention layer
  - Positional embeddings

Positional embeddings

References:

1. https://arxiv.org/pdf/1706.03762.pdf
2. https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
3. https://www.tensorflow.org/tutorials/text/nmt_with_attention

Positional embeddings

Consider the 2 following sentences:

> I **do not** like the story of the movie, but I **do** like the cast

> I **do** like the story of the movie, but I **do not** like the cast

What is the difference between these 2 sentences?

Souce: https://is.gd/positionalembedding

Office on the web Frame

Multi-Head Attention

Multi-Head Attention

- Encodes each word's relationship with every other in a sentence
- Paying more attention to the most relevant ones
- Called **Self-Attention**

For example:

- "They welcomed the Queen of the United Kingdom"
- The output for "Queen" will depend on all the words in the sentence,
- but it will probably pay more attention to
    - "United" and "Kingdom" than
    - "They" or "welcomed."

Multi-Head Attention

- Before we get into Multi-Headed Attention, we should first look at the concept of **Scaled Dot-Product Attention**
- **Example:**

Multi-Head Attention - Scaled Dot-Product Attention

- Say, encoder analyzed “They played chess,” and understood:
  - “They” → Subject
  - "Played" → verb
- This is encoded in the representations of the words
- Say, decoder has already translated the subject
  - and it thinks that it should translate the verb next
  - For this, it needs to fetch the verb from the input sentence.
- This is similar to a dictionary lookup:
  - Look up “verb” in {“subject”: “They”, “verb”: “played”, ...}

Multi-Head Attention

- Compared to the standard form of Attention, Scaled Dot-Product Attention utilizes Scaled Dot-Product to calculate similarity

$$\text{Attention}\,(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_{keys}}}\right)\mathbf{V}$$

Multi-Head Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_{keys}}}\right)\mathbf{V}$$

- Q is a matrix containing one row per query of shape $[n_{queries}, d_{keys}]$
- $n_{queries}$ is the number of queries
- d_{keys} is the number of dimensions of each query and each key

Multi-Head Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_{keys}}}\right)\mathbf{V}$$

- K is a matrix containing one row per key of shape [n_{keys}, d_{keys}]
- n_{keys} is the number of keys and values

Multi-Head Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_{keys}}}\right)\mathbf{V}$$

- V is a matrix containing one row per value of shape [n_{keys}, d_{values}]
- d_{values} is the number of each value

Multi-Head Attention

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{d_{keys}}}\right)\mathbf{V}$$

- The shape of QK^T is $[n_{queries}, n_{keys}]$
- It contains one similarity score for each query/key pair

Multi-Head Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_{keys}}}\right)\mathbf{V}$$

- The output of the softmax function has the same shape, but all rows sum up to 1
- The final output has a shape of [$n_{queries}$, d_{values}], there is one row per query, where each row represents the query result (a weighted sum of the values)

Multi-Head Attention

- The `keras.layers.Attention` layer implements Scaled Dot-Product Attention
- Its inputs are just like Q, K, and V, except with an extra batch dimension (the first dimension)

Multi-Head Attention

- It is a bunch of Scaled Dot-Product Attention layers
- Each preceded by linear transformation of the values, keys, and queries
- All outputs are concatenated
- And they go through a final linear transformation

Multi-Head Attention

But why?

What is the intuition behind this architecture?

Multi-Head Attention

- The word representation encodes many different characteristics of the word
- With a single Scaled Dot-Product Attention layer this is not possible

Multi-Head Attention

- This is why Multi-Head Attention layer applies multiple different linear transformations of values, keys, and queries
- It gives the attention layer multiple "representation subspaces"

Next Word Prediction

Next Word Prediction

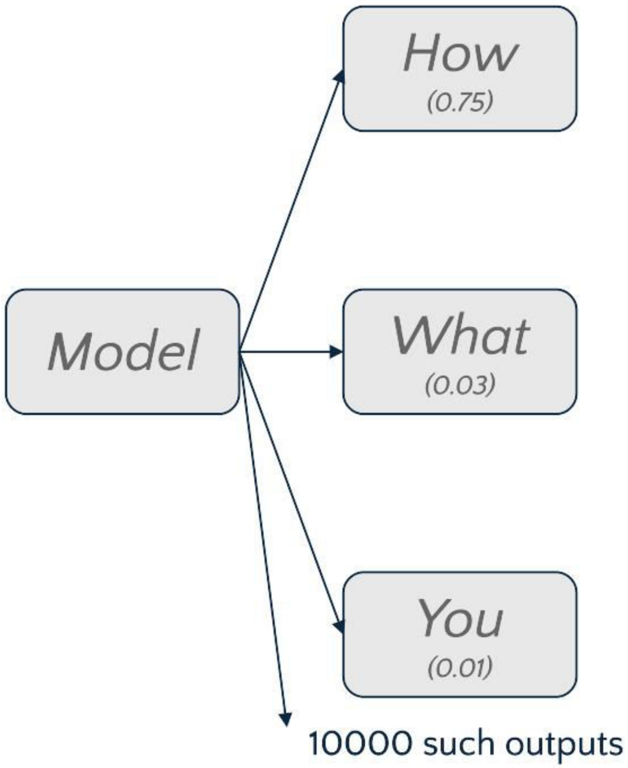
- Generating text requires
  - ensuring overall coherence,
  - grammar and
  - diversity in the sequence.
- A balance between local and global optimization.
  - Local optimization focuses on immediate choices,
  - while global optimization aims to find the overall best sequence.

Beam Search

Beam Search

- It keeps track of k (**beam width**) most promising sentences
- At each decoder step
    - It tries to extend them by one word
    - Keeping only the k most likely sentences

Beam Search (width - 3)

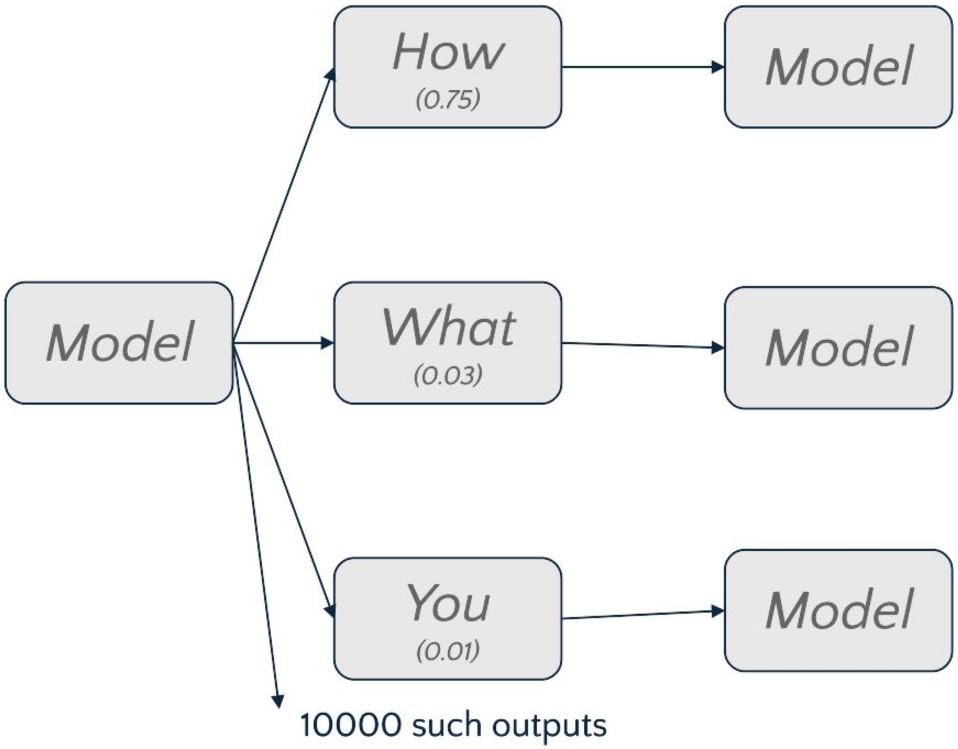


Beam Search: Working

Step 2

- Expansion:
    - Predict the next words to generate multiple possibilities.
    - Create new sequences by adding these words to the existing ones.

Beam Search (width - 3)



Beam Search: Working

Step 3

- Scoring:
  - Assign scores to sequences based on the probability of each word.
  - Calculate the overall probability for each sequence.