2025

Lesson 02

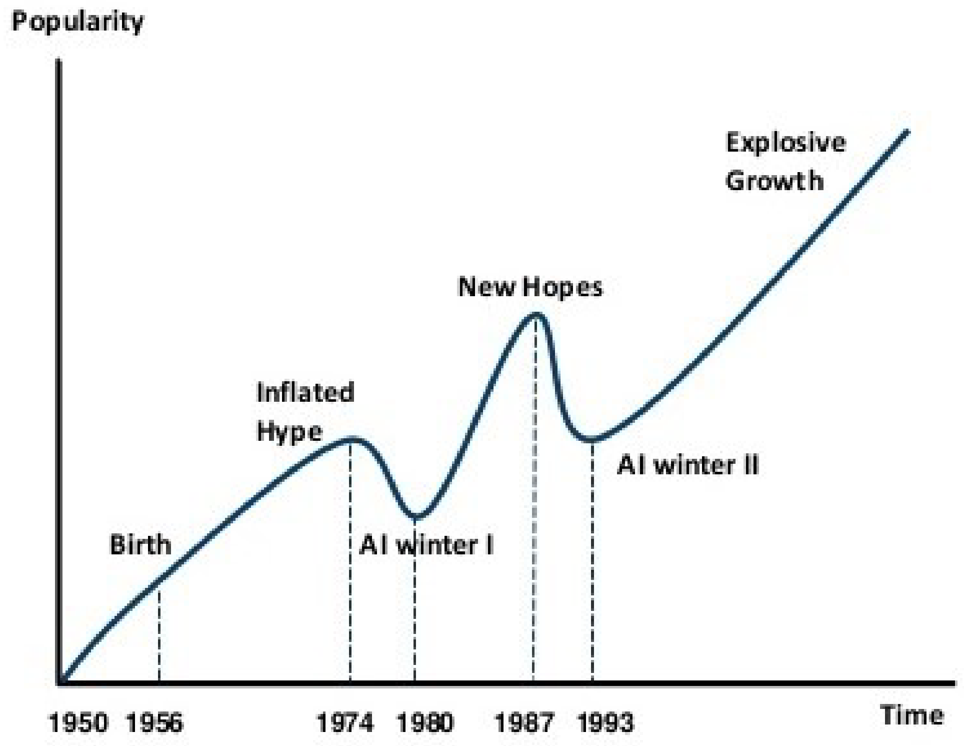
Transformer – An Introduction

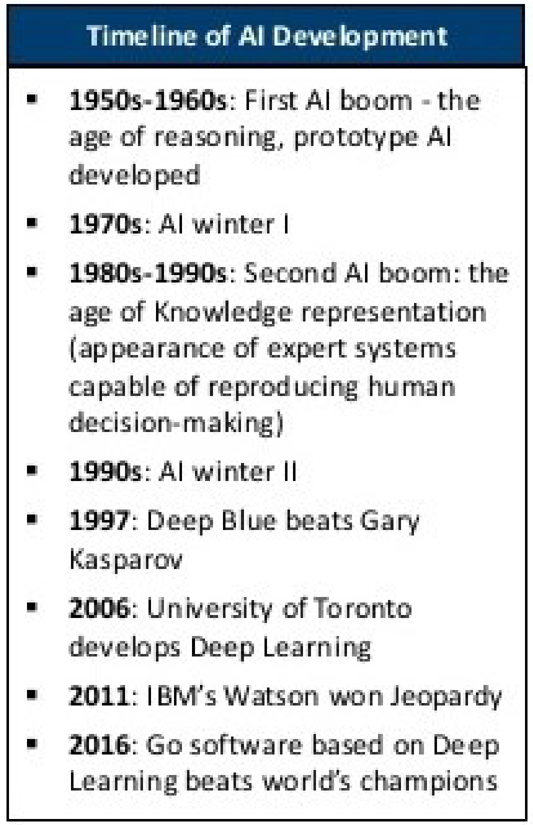
AI History

- AI comes a long way.
- The concept was first proposed almost 70 years ago.

AI HAS A LONG HISTORY OF BEING "THE NEXT BIG THING"...

https://www.actuaries.digital/2018/09/05/history-of-ai-winters/





Timeline of AI Development

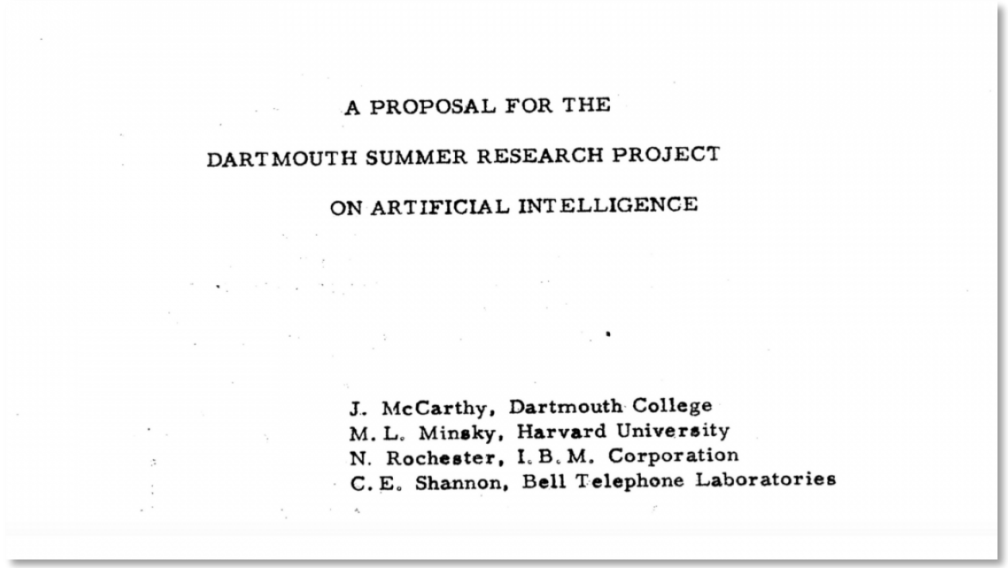
- **1950s-1960s**: First AI boom - the age of reasoning, prototype AI developed
- **1970s**: AI winter I
- **1980s-1990s**: Second AI boom: the age of Knowledge representation (appearance of expert systems capable of reproducing human decision-making)
- **1990s**: AI winter II
- **1997**: Deep Blue beats Gary Kasparov
- **2006**: University of Toronto develops Deep Learning
- **2011**: IBM's Watson won Jeopardy
- **2016**: Go software based on Deep Learning beats world's champions

1956 Dartmouth Conference: The Birth of AI

- Coining of "Artificial Intelligence": The Dartmouth Conference is where the term "Artificial Intelligence" (AI) was first officially used. John McCarthy, one of the conference's organizers, is credited with coining the term.
- Foundational Moment for AI: This event is often considered the founding moment of AI as a field of study. It was the first time researchers from various disciplines came together to explore the concept of machine intelligence.

Key Participants

- John McCarthy: Known for later developing the LISP programming language, a cornerstone of AI research
- Marvin Minsky: A cognitive scientist and AI pioneer who would later co-find the MIT AI Lab
- Claude Shannon: Known as the father of information theory, he brought a deep understanding of communication and information processing to the discussion
- Nathaniel Rochester: An IBM researcher who contributed to the development of the first computers and was interested in the possibilities of programming machines to think



A PROPOSAL FOR THE

DARTMOUTH SUMMER RESEARCH PROJECT

ON ARTIFICIAL INTELLIGENCE

J. McCarthy, Dartmouth College
M. L. Minsky, Harvard University
N. Rochester, I.B.M. Corporation
C. E. Shannon, Bell Telephone Laboratories

https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth

DARTMOUTH ENGINEERING https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth NVIDIA. https://developer.nvidia.com/gen-ai-teaching-kit-syllabus

1958 Perceptron By Rosenblatt at Cornell

Foundational Model for Neural Networks:

The perceptron, introduced by Frank Rosenblatt in 1958, is one of the earliest models of an artificial neuron, laying the groundwork for the development of modern neural networks. It was inspired by biological neurons and aimed to mimic the way the brain processes information.

Limitations and Influence on Future Research:

While the perceptron was a breakthrough, it had significant limitations, most notably its inability to solve problems that are not linearly separable, such as the XOR problem.

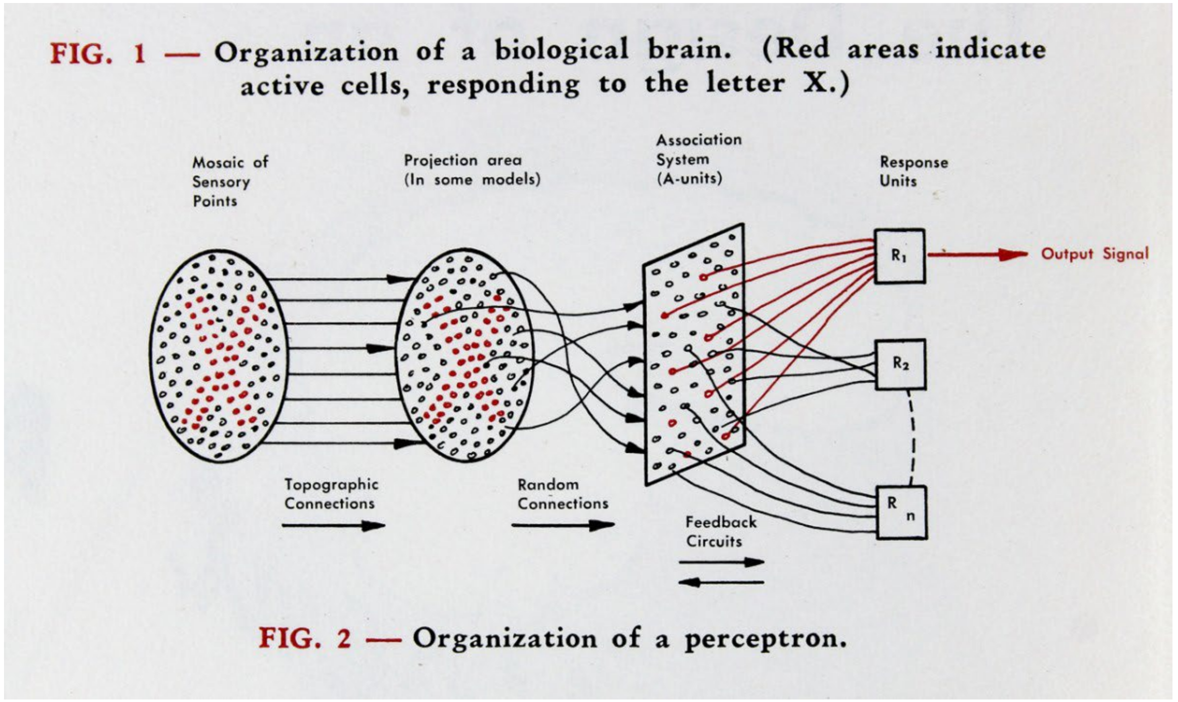


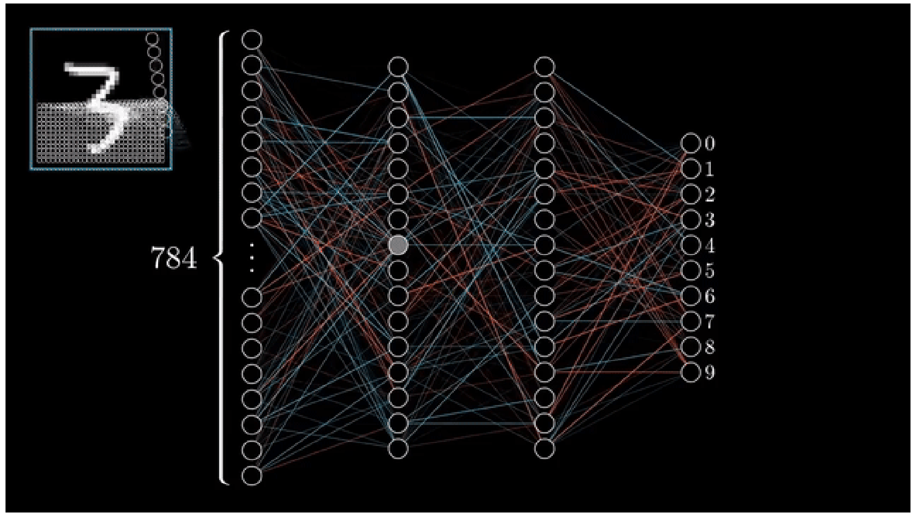
FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)
Mosaic of Sensory Points
Projection area (In some models)
Association System (A-units)
Response Units
R₁
R₂
Rn
Output Signal
Topographic Connections
Random Connections
Feedback Circuits
FIG. 2 — Organization of a perceptron.

https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon

Backpropagation (By Hinton et. Al)

Backpropagation, or “backward propagation of errors,” is an algorithm used to train neural networks by efficiently computing the gradient of the loss function with respect to the network’s weights. This process allows the network to adjust its weights to minimize errors, thereby improving its performance.

- It uses the chain rule from calculus to compute gradients layer by layer, starting from the output layer and moving backward through the network.
- Before backpropagation, neural networks were primarily limited to linear models with simple learning capabilities.
- The introduction of backpropagation demonstrated that neural networks could approximate any continuous function, given sufficient data and appropriate network architecture.



https://www.youtube.com/watch?v=Ilg3gGewQ5U

https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth

https://developer.nvidia.com/gen-ai-teaching-kit-syllabus

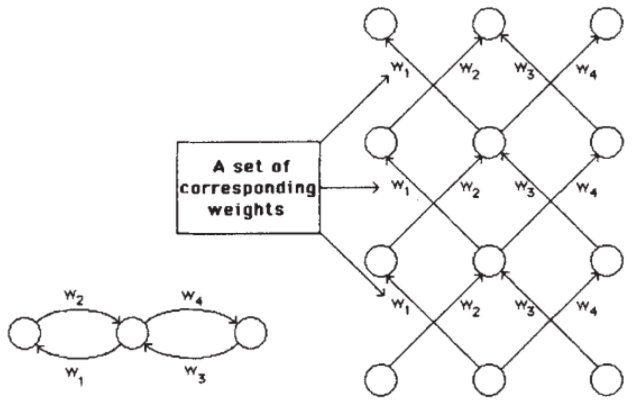
1986 – Multi-layer Perceptron

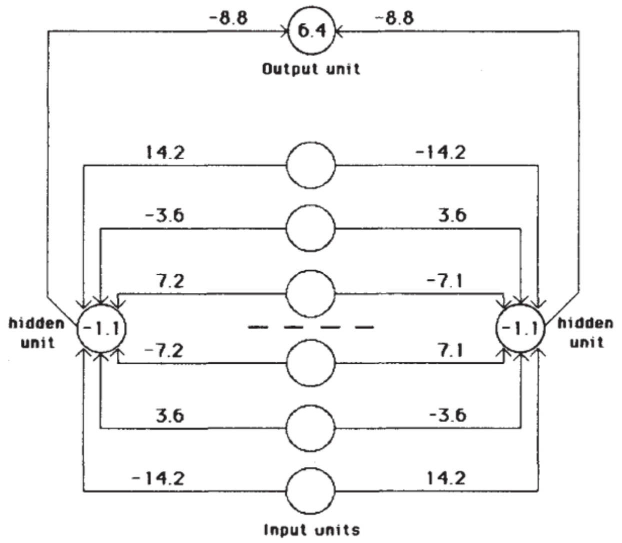
In the same backpropagation paper from 1986, the multi-layer perceptron was demonstrated to train to classify classes.

Learning Internal Representations: The authors demonstrated that MLPs could learn internal representations in hidden layers, which are crucial for capturing complex features in data. This ability to learn hierarchical representations was a significant advancement over previous models.

Non-Linear Capabilities: The paper highlighted that MLPs with non-linear activation functions in the hidden layers could solve problems that linear models could not, such as XOR, which was a known limitation of single-layer perceptrons.

Empirical Validation: The 1986 paper provided empirical results showing that MLPs, trained using backpropagation, could successfully perform tasks like pattern recognition, proving the effectiveness of the approach in practical applications.





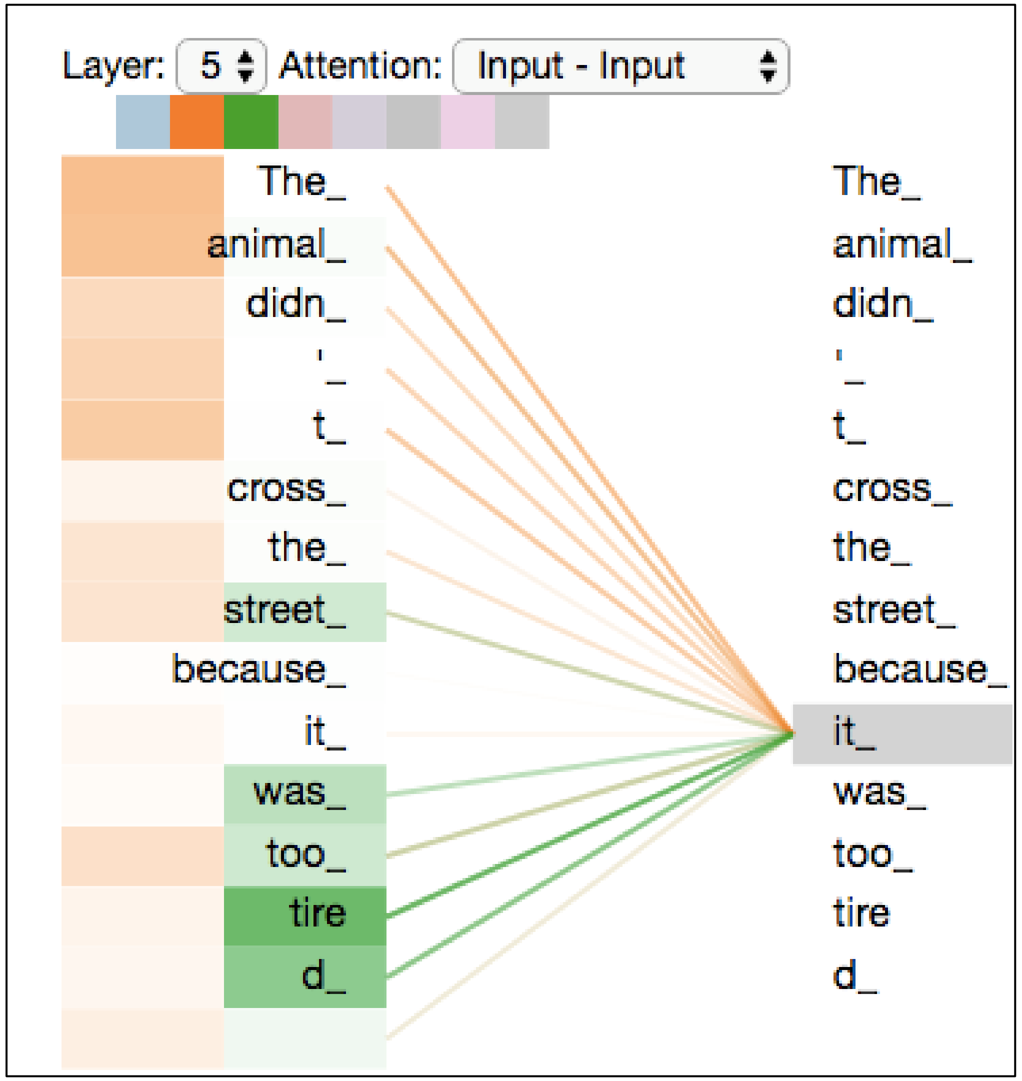
https://www.nature.com/articles/323533a0

DARTMOUTH ENGINEERING https://home.dartmouth.edu/about/artificial-intelligence-ai-coined-dartmouth NVIDIA. https://developer.nvidia.com/gen-ai-teaching-kit-syllabus

RP

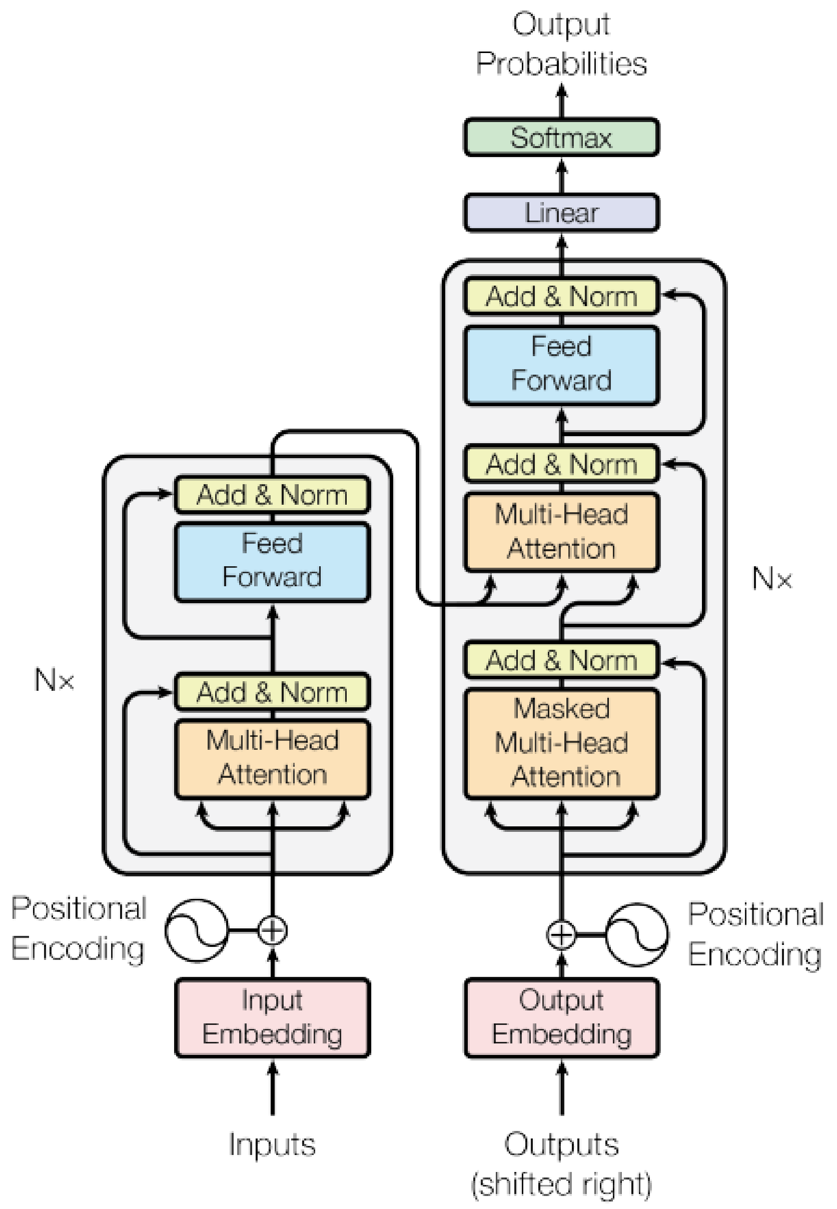
AI History

- “Neural Machine Translation by Jointly Learning to Align and Translate“ by Dzmitry Bahdanau et al. (2015) provided a significant milestone in the development of neural machine translation.
- Introduced the concept of attention mechanism which greatly improved the performance of machine translation systems.
- The development of NLP research activities didn’t really pick up until the Transformer architecture was proposed in the paper titled “Attention is All You Need” by Vaswani et al. (2017)
- Both “attention” designs were not quite the same.
- The transformer architecture marked a significant leap in the field of NLP and form the basis for many state-of-the-art (SOTA) models.



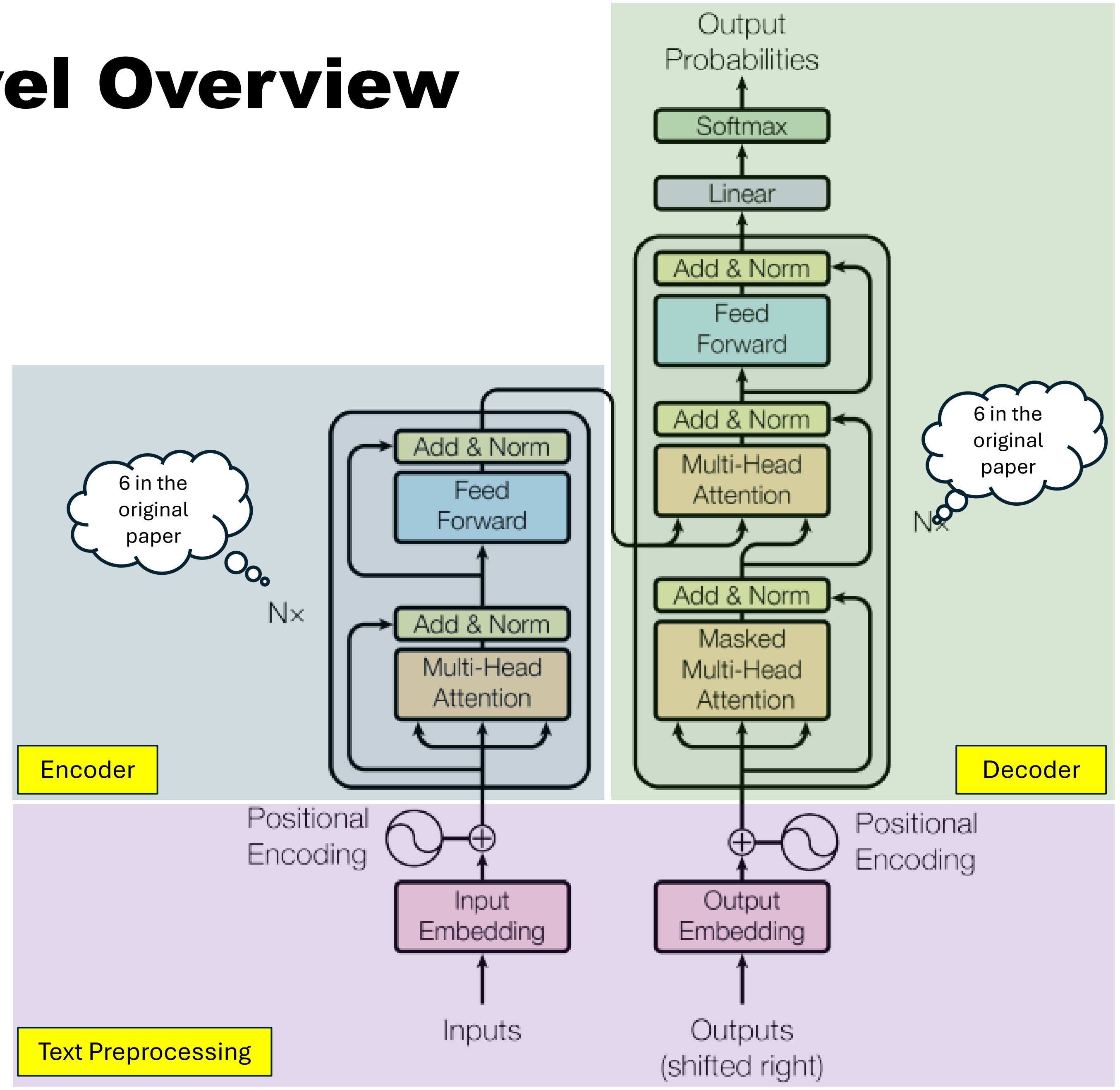
https://diverger.medium.com/attention-mechanisms-and-beyond-c6fd48112d09

Transformer Architecture (2017)



Source: https://arxiv.org/pdf/1706.03762.pdf

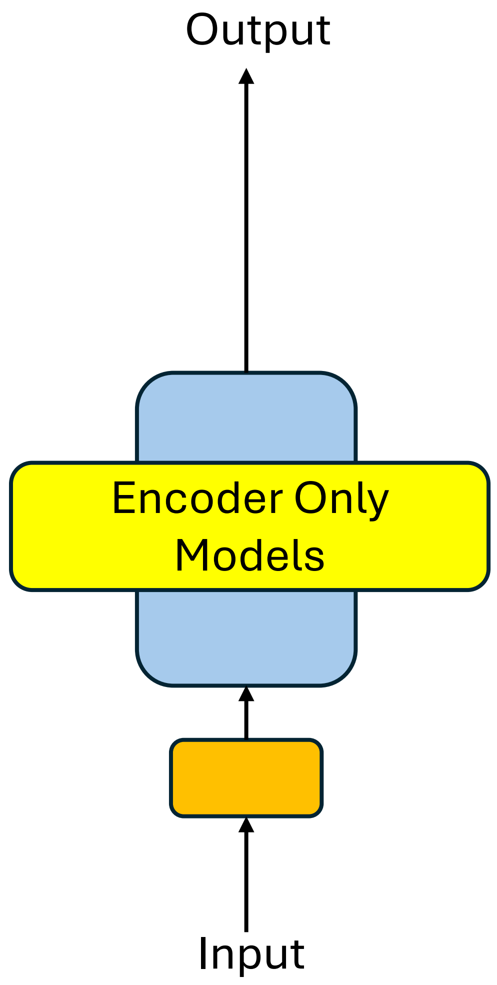
High Level Overview

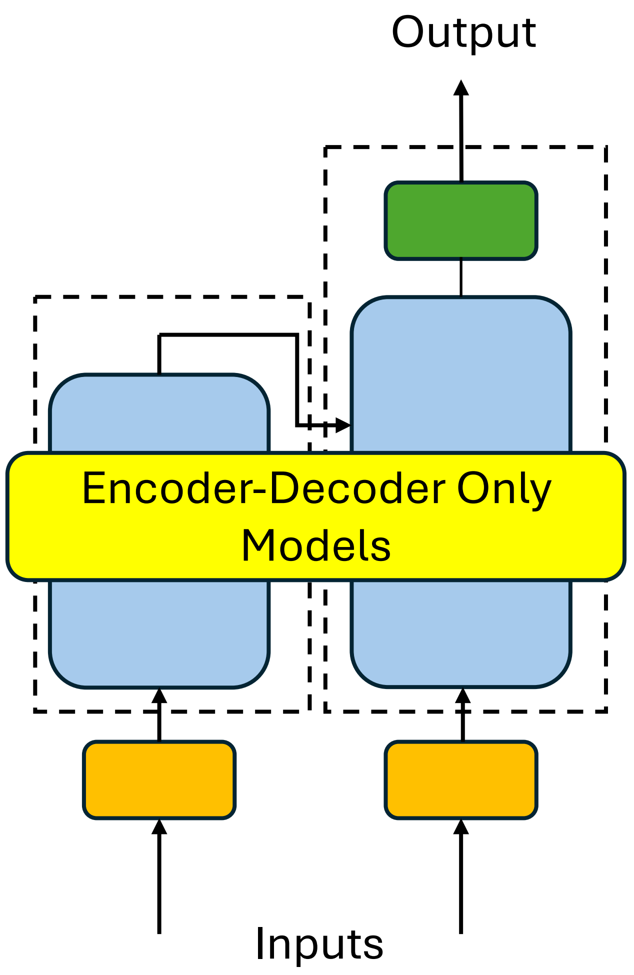


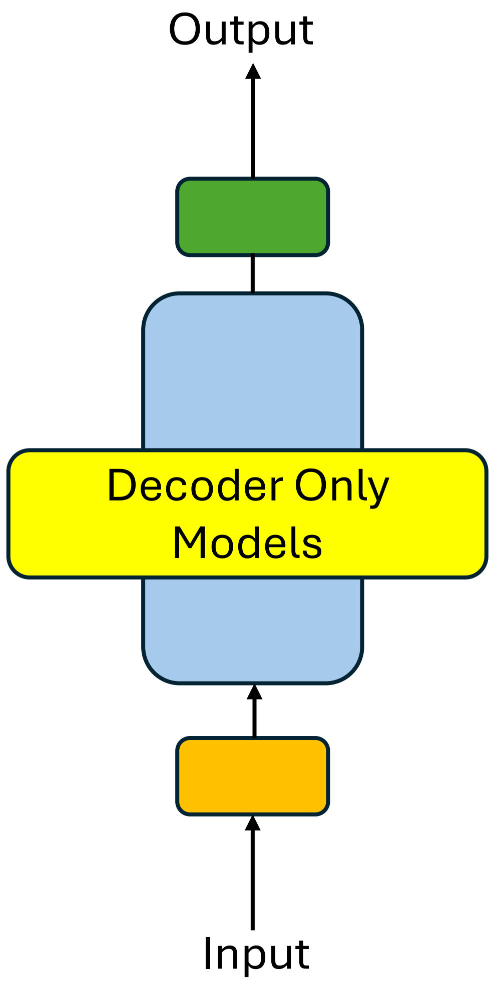
Transformer Architecture

- All language models use some variants of the transformer architecture.
- The original proposed architecture was for solving sequence-to-sequence tasks i.e., translation.
- It was extended to solve a variety of different problems.
  - Semantic similarity of text
  - Classifying images
- Encoder processes the input sequence and produces an output sequence.
- Decoder generates its own output sequence, given the encoder's output sequence as input.
- Transformer takes a sequence of input and produces a new sequence as output.

Architecture Variants







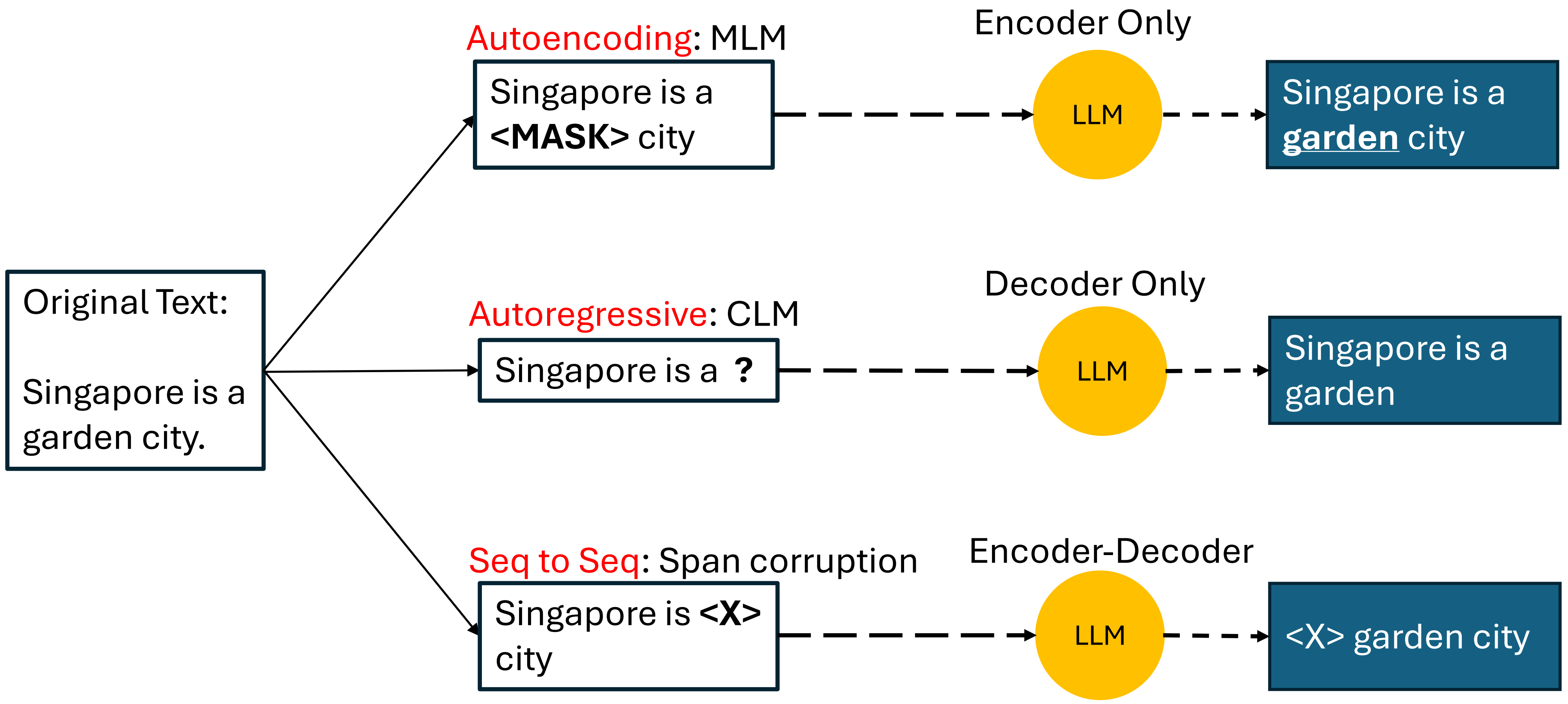
Model Characteristics

- Encoder (aka Autoencoding models)
  - Pretrained using Masked Language Modelling (MLM).
  - Objective is to reconstruct text (‘denoising’).
  - Bidirectional context.
  - Good use cases for: Sentiment Analysis, Named Entity Recognition (NER) and Word Classification.
  - Models: BERT, ROBERTA, distilBERT.
- Decoder (aka Autoregressive models)
  - Pretrained using Causal Language Modelling (CLM).
  - Objective is to predict the next token.
  - Unidirectional context.
  - Good use cases for: Text generation.
  - Models: GPT, BLOOM.

Model Characteristics

- Encoder-Decoder (aka Sequence-to-Sequence models)
  - Pretrained techniques varies, for example, span corruption.
  - Objective is to reconstruct the span of words.
  - Good uses cases for Translation, Text Summarization and Question Answering.
  - Models: T5, BART, Marian.

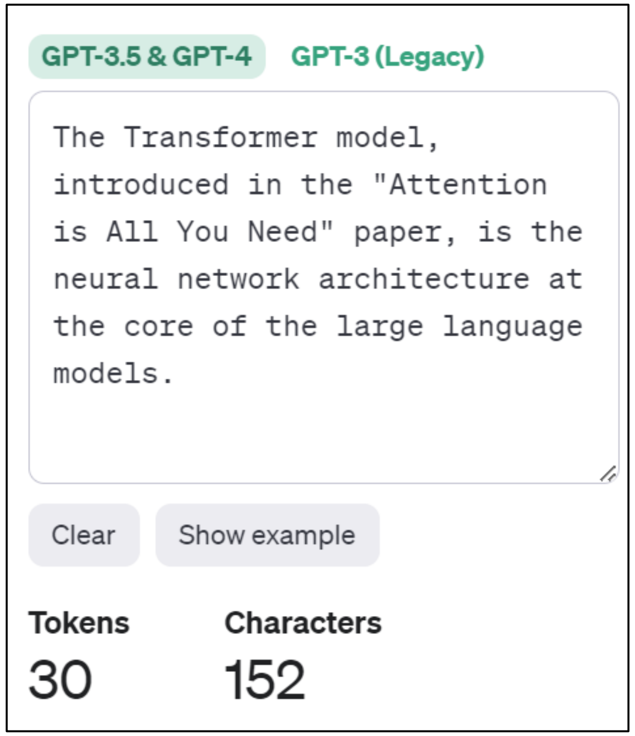
Pre-Training



MLM: Masked Language Modelling
CLM: Causal Language Modelling

Text Processing

- Tokenization
  - Convert raw text (corpus) into a format that the model can understand.
  - It involves splitting the text into tokens (usually words or sub-words).
  - There are many methods to tokenize text:
    - NLTK (Natural Language Toolkit)
    - spaCy
    - Tokenizer from Hugging Face Transformers
    - Stanford NLP
    - Gensim
    - etc.

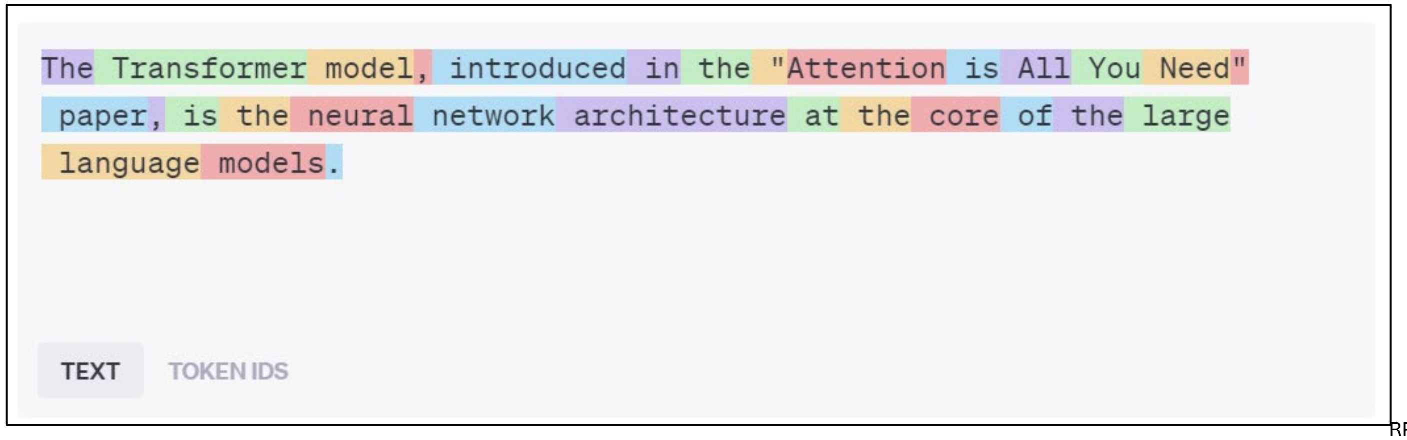


```
The Transformer model, introduced in the "Attention is All You Need" paper, is the neural network architecture at the core of the large language models.
```

Tokens 30

Characters 152

An example: OpenAI Tokenizer



```
The Transformer model, introduced in the "Attention is All You Need" paper, is the neural network architecture at the core of the large language models.
```

Source: https://platform.openai.com/tokenizer

Tokenizer

- The computers aren’t very good with words but are great with numbers.
- NLP models have limitation on the maximum number of token it can process.
- Tokenization breaks down a continuous stream of text into smaller, discrete units called tokens.
- Token helps to provides a structured way to break down text into manageable pieces for the model to process.
- Tokens are typically words, sub-words, characters (including punctuations) depending on the specific tokenization approach used.
- Example: I’m hungry vs I am hungry.
  - “I’m” ➔ 1 word
  - “I am” ➔ 2 words

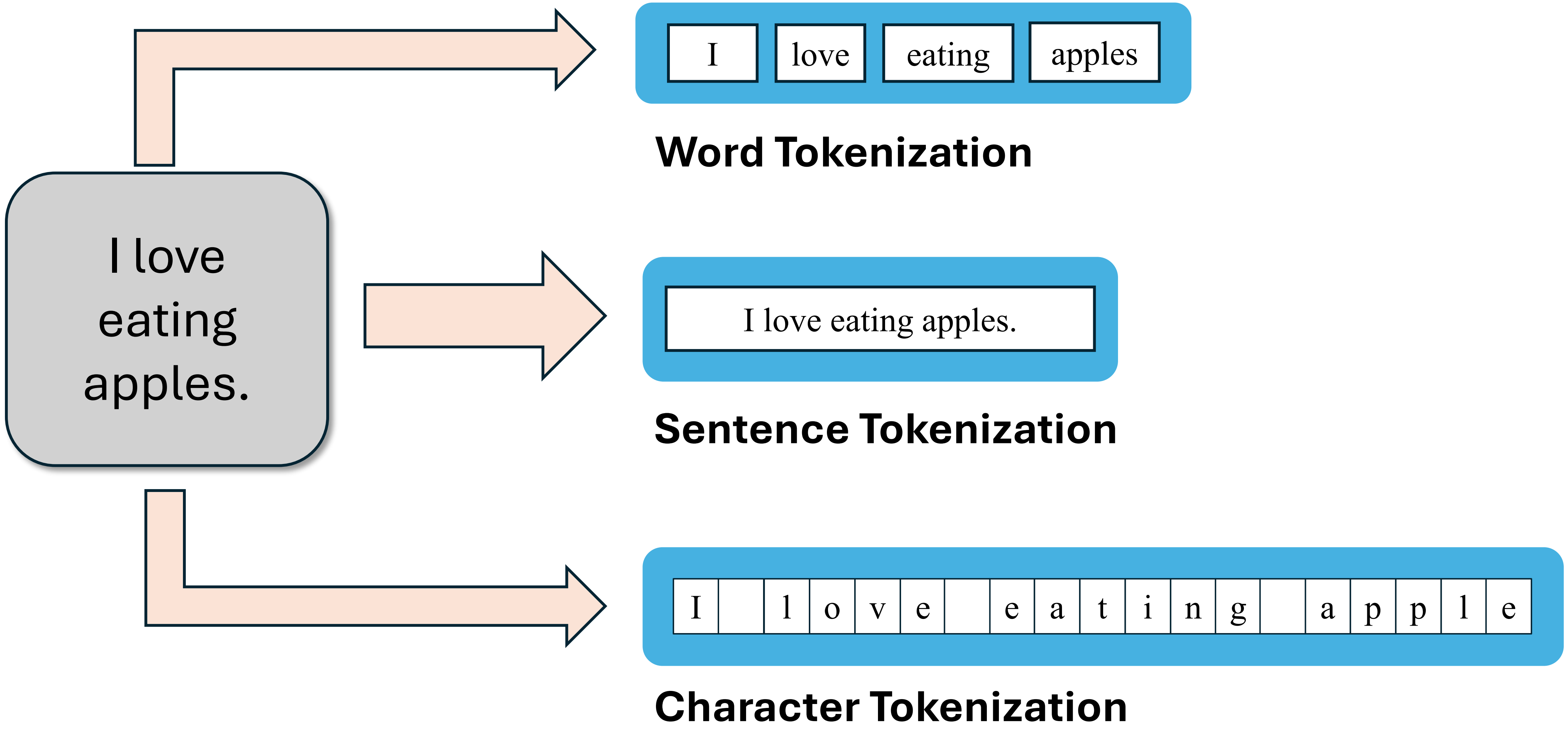
Tokenizer

- Tokenizer is an integral part of a language model used during training.
- When a model is trained, it learns not just the relationship between words but also specific tokenization patterns introduced by the tokenizer.
- Tokens are basic units that the model processes and different tokenizers can produce different token sequences for the same text.
- Using a different tokenizer during inference than the one used during training is not recommended.
  - Can lead to token misalignment
  - Out-of-vocabulary problems
  - Degraded performance

Tokenization Approaches

- Character-level tokenization
  - Represent the entire English language with limited characters i.e. upper/lower case character plus punctuation.
  - Not very useful since the meaning of “Cat” or “Cradle” is different although the “C” in both words means the same.
- Word-level tokenization
  - More meaningful than character-level tokenization but requires a much larger dictionary.
- Sentence-level tokenization
  - Capture meaningful phrases but resulted in an absurdly large dictionary.
- Each method has its pros and cons.
- Best solution provides decent compromises.

Tokenization



Embeddings

- The next step after tokenizing the text is to convert the words to numbers.
- Transform text into something that machines can interpret is called “word embedding”.
- Word embeddings are numerical illustrations of a text.
- Sentence and texts include organized sequences of information.
- Goal is to develop a representation of words that capture their meanings, meaningful connections plus other sorts of situations in which the words are employed.

Embeddings

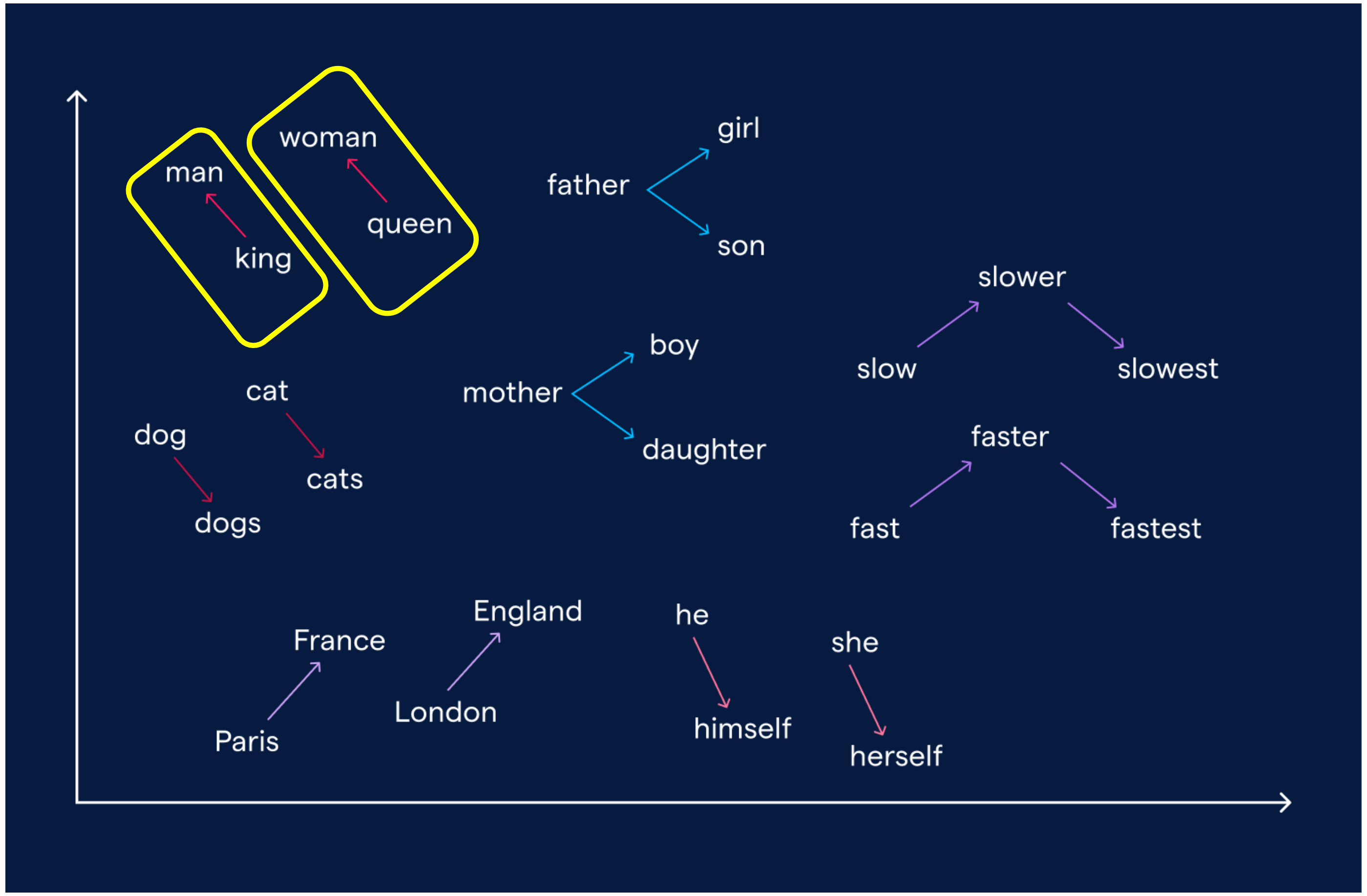
- The aim is to generate numeric which forms the semantic representation from those tokens.
- A list of numbers which holds the semantic representation of a word.

  [-2, 4, -3.7, 41 …., -0.98]
- One of the important characteristics is that if we plot it on a 2D graph, similar terms will be closer than dissimilar terms.
- Words that are found in similar contexts will have similar embeddings.
- An embedding contains a dense vector of floating-point values.
- Embeddings utilizes an efficient representation in which related words are encoded in the same manner.

* A dense vector is one where most of its elements are non-zero. A sparse vector is one where most of its elements are zero.

Embeddings

- The vector from king➔man is very similar to queen➔woman.



Source: Feature Form

Pre-trained Word Embedding

- Modern Natural Language Processing (NLP) uses word embeddings that has been previously trained on a large corpus of text.
- It is hence called “Pre-trained Word Embeddings”.
- Captures both the connotative and syntactic meaning of a word.
- Examples of pre-trained word embeddings are:
  - Word2Vec
  - Continuous Bag-of-Words (CBOW)
  - Skip-Gram Model
  - GloVe
  - fastText
  - ELMo
  - BERT

Positional Encodings

- Now that we have a way to represent (and learn) words (numbers), can we make it better?

> The hunter chased the boar.
> The boar chased the hunter.

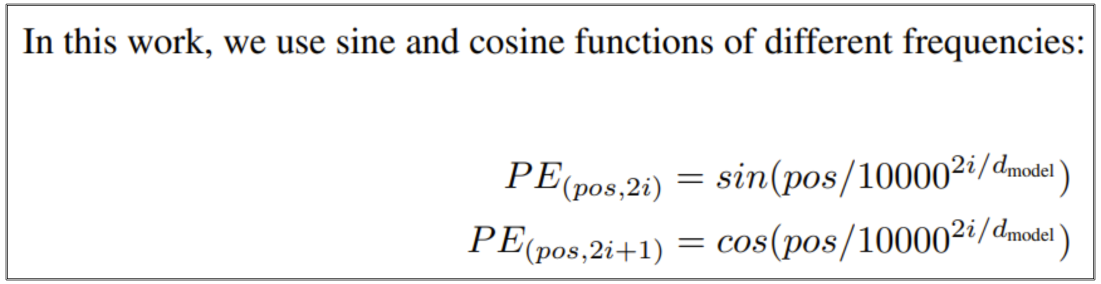
- The two sentences above can be represented using the same embeddings.
- But the two sentences have different meaning,
  - Problem:
    - The embeddings contain semantic meanings, but no exact order meaning
    - And ORDER usually matters
- We can compute the positional encoding with sine and cosine functions of varying frequencies.

Positional Encodings

- Using embeddings to represent words without order is not good enough.
- Transformer add positional encodings to the embeddings.
- Calculate a position vector (a list of numbers again!) for every word and summing the two vectors to form another vector.
- The positional encodings are typically added to the input embeddings before sending them into the transformer model.
- Positional encodings are necessary to introduce the notion of position or order.

Positional Encodings

- The common approach is to use sine and cosine functions to generate these positional encodings.



In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Source: arxiv.org/pdf/1706.03762.pdf

- $PE_{(pos,2i)}$ and $PE_{(pos,2i+1)}$ are the components of the positional encoding for position *pos* and dimension *2i* and *2i* + 1 respectively.
- d_{model} : dimension of model embedding i.e., 512 in original paper.
- pos : position of the token in the sequence.
- *i* used for making alternate even and odd sequences.

Simple Example

Positional Encoding Matrix with d=4,, n=100

Sequence	Index of Token, K	i=0	i=1	i=0	i=1
I	0	P_{00}=sin(0) = 0	P_{01}=cos(0) = 1	P_{02}=sin(0) = 0	P_{03}=cos(0) = 1
am	1	P_{10}=sin(1/1) = 0.84	P_{11}=cos(1/1) = 0.54	P_{12}=sin(1/10) = 0.10	P_{13}=cos(1/10) = 1.0
a	2	P_{20}=sin(2/1) = 0.91	P_{21}=cos(2/1) = -0.42	P_{22}=sin(2/10) = 0.20	P_{23}=cos(2/10) = 0.98
Robot	3	P_{30}=sin(3/1) = 0.14	P_{31}=cos(3/1) = -0.99	P_{32}=sin(3/10) = 0.30	P_{33}=cos(3/10) $\boxed{= 0.96}$

$$100^{\frac{2i}{d_{models}}} = 100^{\frac{2\times1}{4}} = 100^{0.5} = 10$$

In Radian

$$\cos\left(\boxed{\frac{3}{10}}\right) = 0.9553364 \approx 0.96$$

https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Position Encoding (Sinusoids)

- Original transformer architecture used Sinusoids.

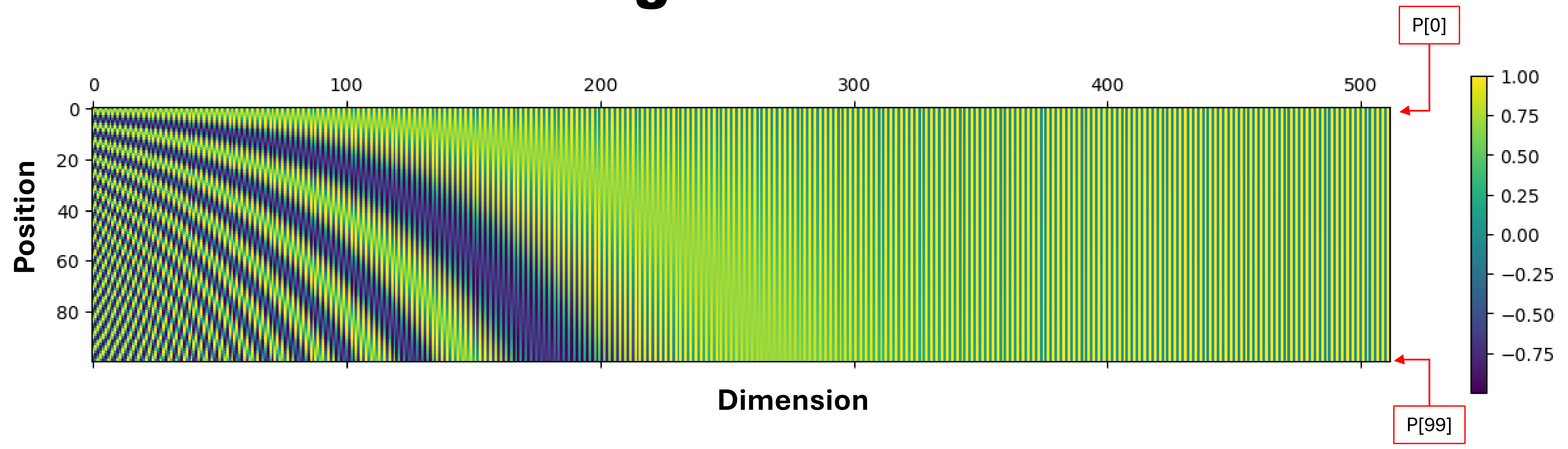
```
import numpy as np
import matplotlib.pyplot as plt

def getPositionEncoding(seq_len, d, n=10000):
    P = np.zeros((seq_len, d))
    for k in range(seq_len):
        for i in np.arange(int(d/2)):
            denominator = np.power(n, 2*i/d)
            P[k, 2*i] = np.sin(k/denominator)
            P[k, 2*i+1] = np.cos(k/denominator)
    return P
```

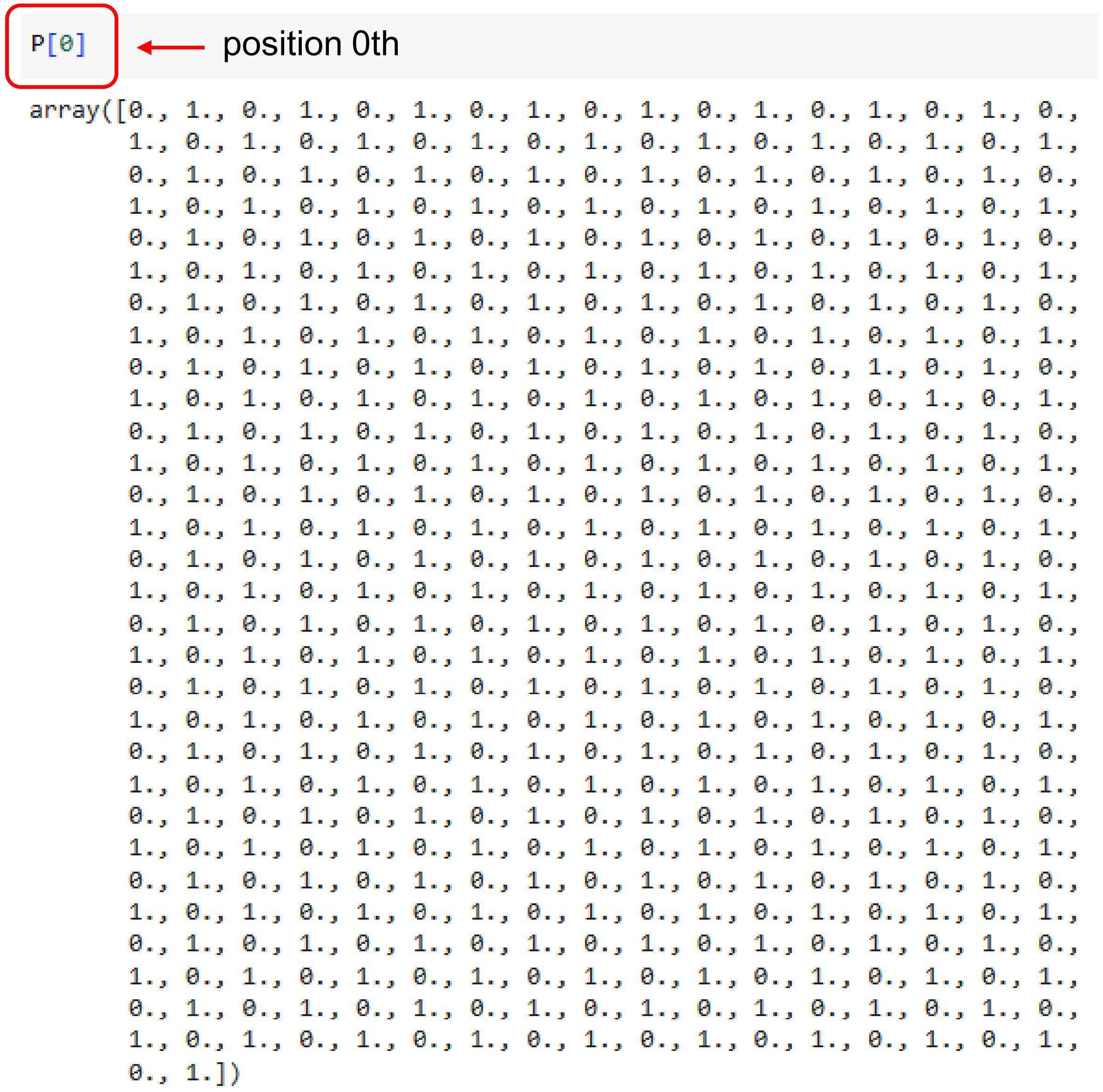
```
def plotSinusoid(k, d=512, n=10000):
    x = np.arange(0, 100, 1)
    denominator = np.power(n, 2*x/d)
    y = np.sin(k/denominator)
    plt.plot(x, y)
    plt.title('k = ' + str(k))

P = getPositionEncoding(
   seq_len=100,
   d=512,
   n=10000)
cax = plt.matshow(P)
plt.gcf().colorbar(cax)
```

Positional Encoding Chart

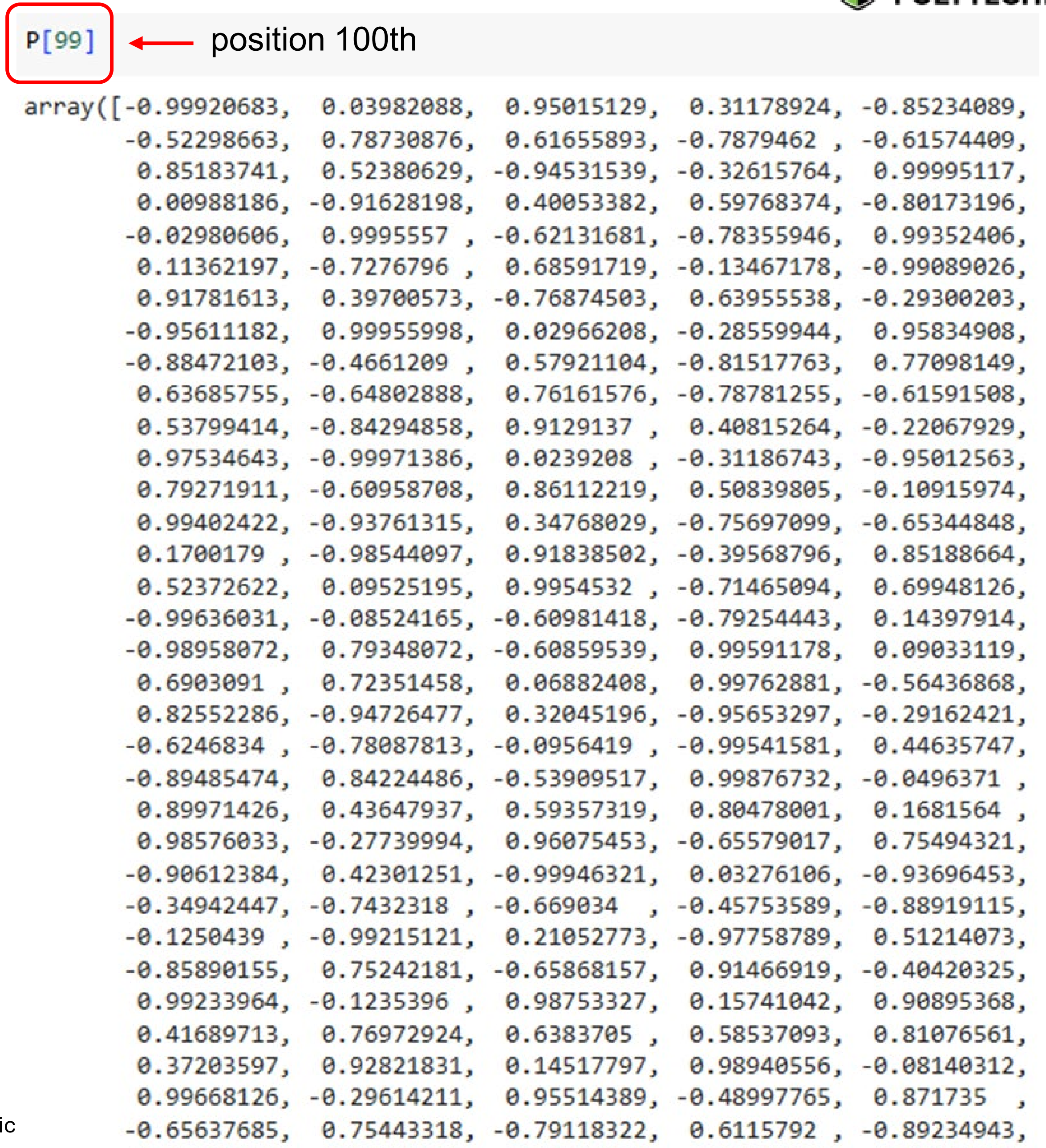


- Each (token) position ➔ Vector of 512 dimensions.
- In the chart above, as an example, there are 100 positions (100 tokens).



P[0] ← position 0th

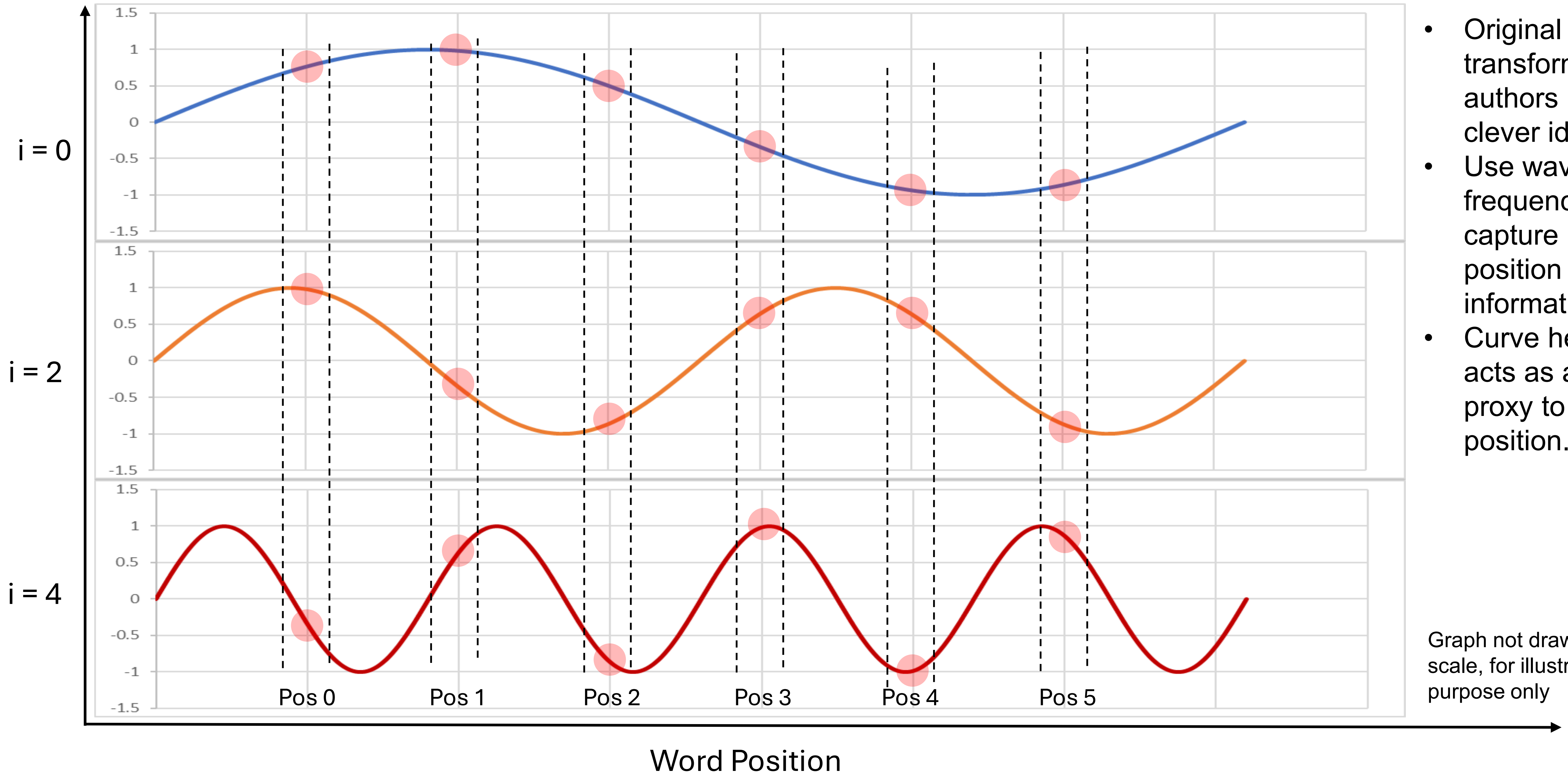
```
array([0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0.,
       1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1.,
       0., 1.])
```



P[99] ← position 100th

```
array([-0.99920683,  0.03982088,  0.95015129,  0.31178924, -0.85234089,
       -0.52298663,  0.78730876,  0.61655893, -0.7879462 , -0.61574409,
        0.85183741,  0.52380629, -0.94531539, -0.32615764,  0.99995117,
        0.00988186, -0.91628198,  0.40053382,  0.59768374, -0.80173196,
       -0.02980606,  0.9995557 , -0.62131681, -0.78355946,  0.99352406,
        0.11362197, -0.7276796 ,  0.68591719, -0.13467178, -0.99089026,
        0.91781613,  0.39700573, -0.76874503,  0.63955538, -0.29300203,
       -0.95611182,  0.99955998,  0.02966208, -0.28559944,  0.95834908,
       -0.88472103, -0.4661209 ,  0.57921104, -0.81517763,  0.77098149,
        0.63685755, -0.64802888,  0.76161576, -0.78781255, -0.61591508,
        0.53799414, -0.84294858,  0.9129137 ,  0.40815264, -0.22067929,
        0.97534643, -0.99971386,  0.0239208 , -0.31186743, -0.95012563,
        0.79271911, -0.60958708,  0.86112219,  0.50839805, -0.10915974,
        0.99402422, -0.93761315,  0.34768029, -0.75697099, -0.65344848,
        0.1700179 , -0.98544097,  0.91838502, -0.39568796,  0.85188664,
        0.52372622,  0.09525195,  0.9954532 , -0.71465094,  0.69948126,
       -0.99636031, -0.08524165, -0.60981418, -0.79254443,  0.14397914,
       -0.98958072,  0.79348072, -0.60859539,  0.99591178,  0.09033119,
        0.6903091 ,  0.72351458,  0.06882408,  0.99762881, -0.56436868,
        0.82552286, -0.94726477,  0.32045196, -0.95653297, -0.29162421,
       -0.6246834 , -0.78087813, -0.0956419 , -0.99541581,  0.44635747,
       -0.89485474,  0.84224486, -0.53909517,  0.99876732, -0.0496371 ,
        0.89971426,  0.43647937,  0.59357319,  0.80478001,  0.1681564 ,
        0.98576033, -0.27739994,  0.96075453, -0.65579017,  0.75494321,
       -0.90612384,  0.42301251, -0.99946321,  0.03276106, -0.93696453,
       -0.34942447, -0.7432318 , -0.669034  , -0.45753589, -0.88919115,
       -0.1250439 , -0.99215121,  0.21052773, -0.97758789,  0.51214073,
       -0.85890155,  0.75242181, -0.65868157,  0.91466919, -0.40420325,
        0.99233964, -0.1235396 ,  0.98753327,  0.15741042,  0.90895368,
        0.41689713,  0.76972924,  0.6383705 ,  0.58537093,  0.81076561,
        0.37203597,  0.92821831,  0.14517797,  0.98940556, -0.08140312,
        0.99668126, -0.29614211,  0.95514389, -0.48997765,  0.871735  ,
       -0.65637685,  0.75443318, -0.79118322,  0.6115792 , -0.89234943,
```

Positional Encoding

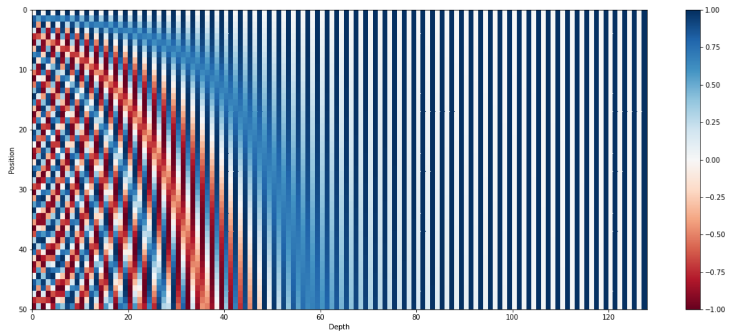


- Original transformer authors had a clever idea.
- Use wave frequencies to capture position information.
- Curve height acts as a proxy to word position.

Graph not drawn to scale, for illustration purpose only

Types of Positional Encoding

- Sinusoidal Positional Encoding
  - Gained popularity through its use in the paper “Attention Is All You Need” by Vaswani et. al.
  - This encoding is deterministic and does not require training, as each position has a unique representation. It allows the model to generalize to longer sequences.
- RoPE
  - Rotary Positional Encoding. Represents positional information by rotating embeddings in a complex space, applying rotational transformations based on token positions.
  - This encoding is particularly useful in applications where the model might need to process very long contexts, such as in language modeling for long documents.
  - RoPE encodes the position by adding a rotation to the query and key vectors in self-attention mechanisms, maintaining efficiency with long sequences.



Source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

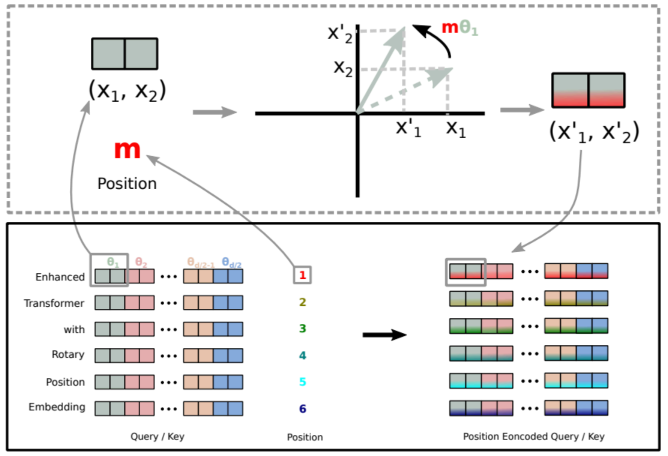


Figure 1: Implementation of Rotary Position Embedding(RoPE).

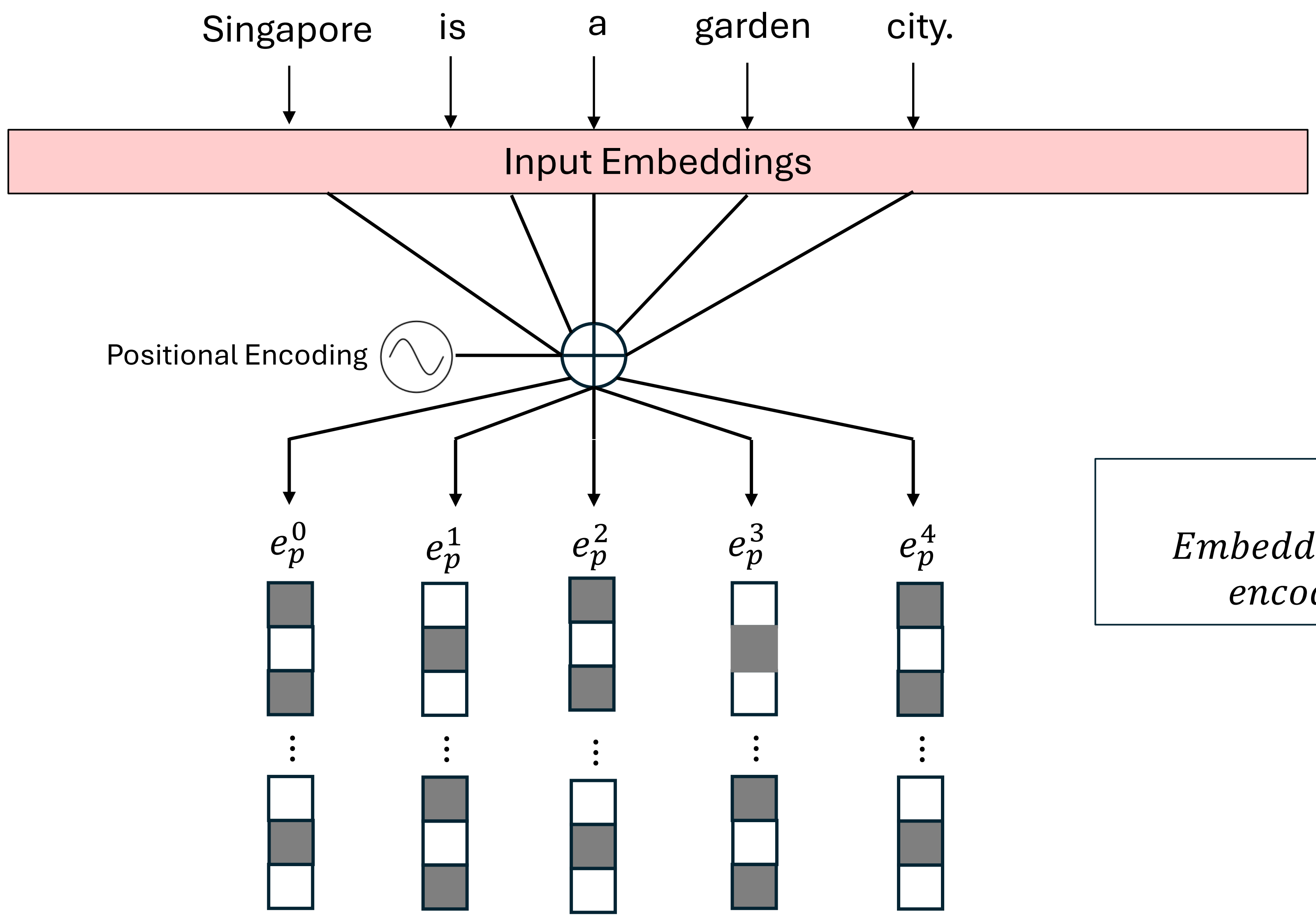
Source: https://paperswithcode.com/method/rope

RP

Types of Positional Encoding

- Relative Positional Encoding
  - Encodes the relative positions between tokens, rather than absolute positions.
- Learnable Positional Embeddings
  - In this approach, the positional encodings are learned parameters rather than fixed sinusoidal values.
  - Each position has an associated embedding that is trained alongside the other model parameters.
  - This approach provides flexibility but requires additional training data for each sequence length, and it might not generalize to unseen sequence lengths as well as sinusoidal encoding.
- Mixture of Positional Embeddings
  - Some models use a mixture of several types of positional embeddings, combining sinusoidal, learned, and/or relative encodings to capture positional information in various ways.
  - This approach can leverage the strengths of each encoding method and adapt to different tasks or input structures.

Let’s Recap

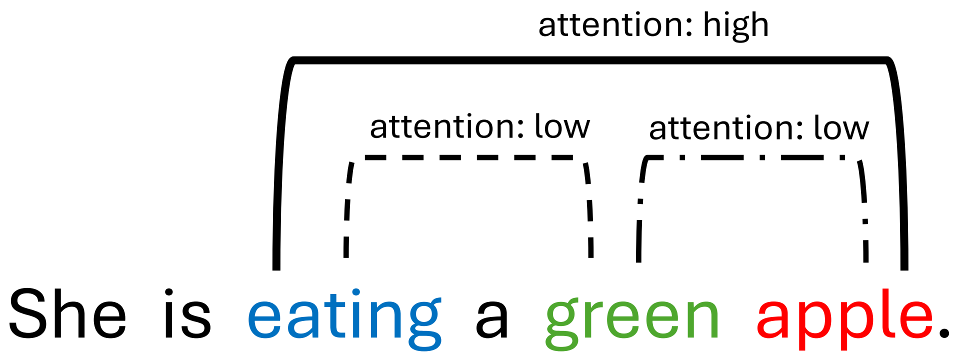


$\boldsymbol{Note}$: $\boldsymbol{e_p^0}$

$Embeddings\ plus\ position\ encoding\ \ at\ index\ 0$

Self Attention

- A mechanism for the model to focus on the relevant parts of the input during training.
- As the model processes each position in the input sequence, self attention allows it to look at other position in the input sequence for clues that can help to a better encoding for this word.



Self Attention Mechanism

Attention Output
Input Sequence
TOKEN
TOKEN
TOKEN
TOKEN
TOKEN
TOKEN
TOKEN
TOKEN

Calculate score from **pairwise comparisons between tokens**
Attention is used, with Q, K, V vectors generated for the token at every position.

Self Attention

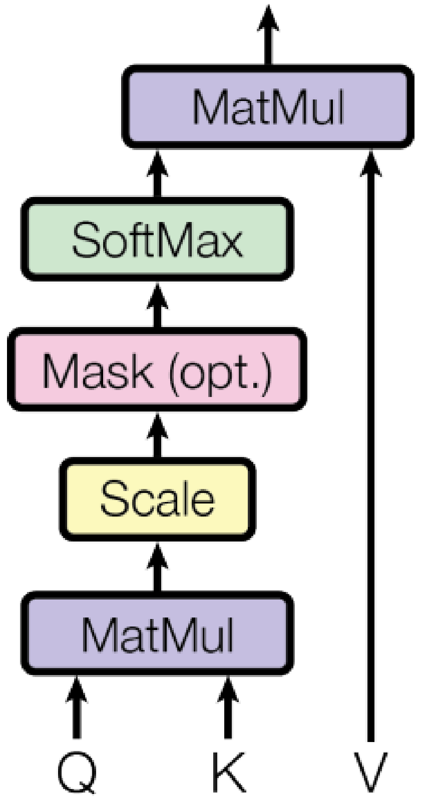
- Model attends to different parts of the input sequence to better capture the contextual dependencies between words.
- Self-attention weights that are learned during training and stored in these layers reflects the importance of each word in that input sequence to all other words in the sequence.
- In practice, this does not happen just once, the transformer architecture has multi-headed self-attention.
- One of the advantages of transformer is: Sentences are processed as a whole rather than word by word.

Scaled Dot-Product Attention

- The outputs (word-embeddings plus positional embeddings) forms the inputs to attention layer by splitting the input into (Q)uery, (K)ey, (V)alue.

Analogy:

- You want to search something from YouTube or Google. The text which you type in the search box is Query.
- The results: videos or article titles are the Key.
- The content inside them is the Value.



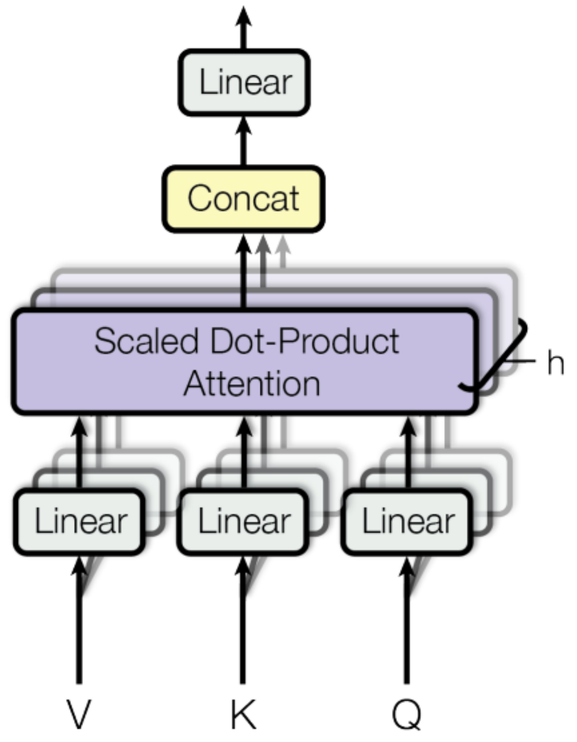
Source: https://arxiv.org/pdf/1706.03762.pdf

Scaled Dot Product Attention

- The output of first matrix multiplication, where we take the similarity of each query to each of the keys is known as the attention matrix.
- The attention matrix depicts how much each token in the sequence is paying attention to each of the keys.
- It then goes through a Softmax function to scale the values to a probability distribution that adds up to 1.
- Another matrix multiplication is performed to “index” into the information for each value in the value matrix in proportion with the amount attention found in the attention matrix.

Multi-Head Attention

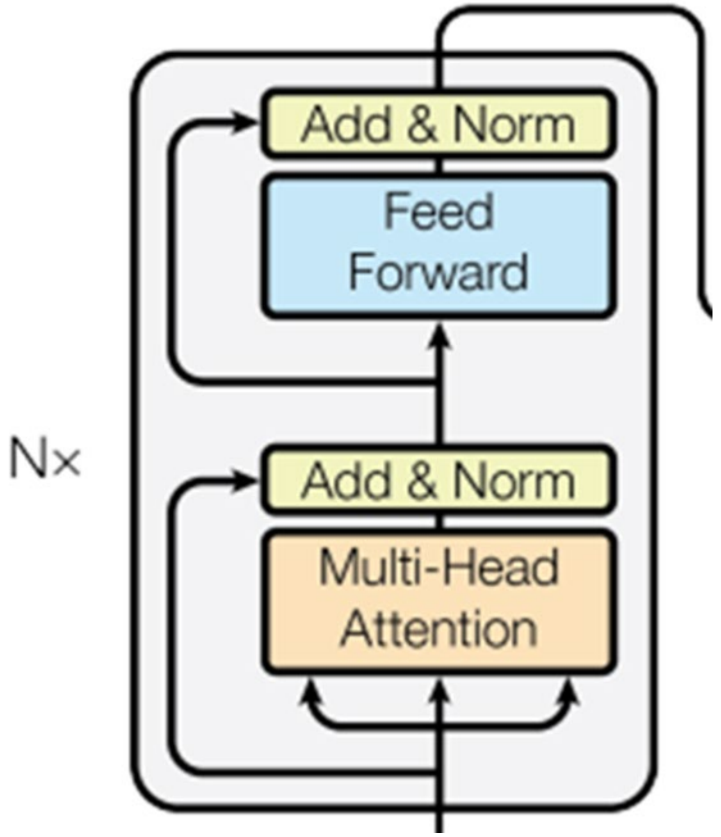
- Multiple sets of self-attention weights (or heads) are learned in parallel independently of each other.
- Each self-attention head learn a different aspect of the language.
- For example, one head may focus on the activity in the sentence whilst another head may focus on other properties such as if the words rhyme.
- The number of attention heads included in the attention layer varies from model to model, but the numbers in the range of 12-100 are common.



Source: https://arxiv.org/pdf/1706.03762.pdf

Multi-Head Attention

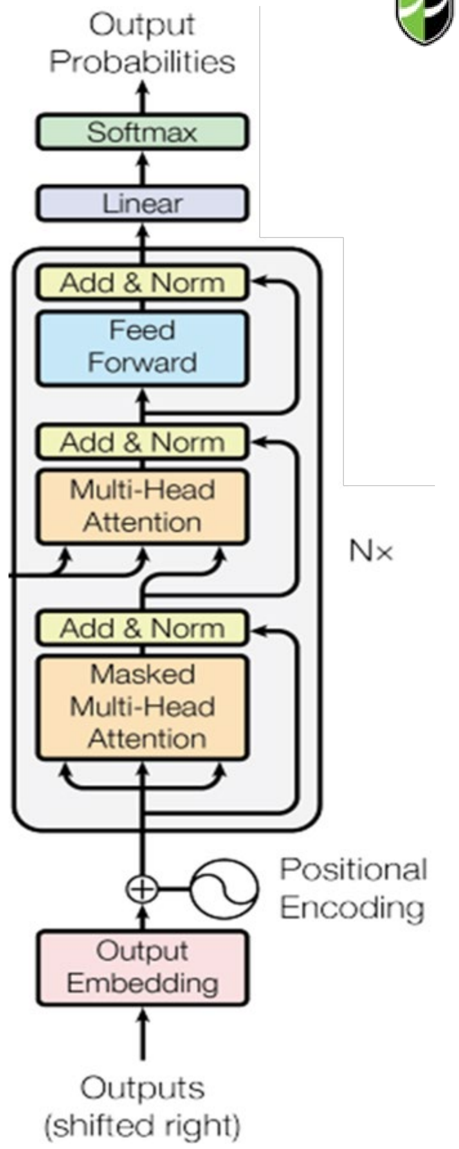
- Cannot dictate ahead of time what aspects of the language the attention head will learn.
- Weights of each heads are randomly initialized.
- With sufficient training data and time, each will learn different aspect of the language.
- Once the attention weights have applied to the input data, the output is processed through a fully-connected feed-forward network.



Decoder

- The output (multi-head attention) of this layer is a vector of logits proportional to the probability score of each and every token in the tokenizer dictionary.
- The logits are passed to a final Softmax layer where they are normalized into a probability score for each word.
- One single token will have a higher score than the rest. This is the most likely predicted token.

Logit: An overloaded term. A raw, unnormalized predictions or scores produced by model before applying a Softmax activation function



REPUBLIC POLYTECHNIC

Thank you!

References

“Illustrated” Series of Blog Posts:

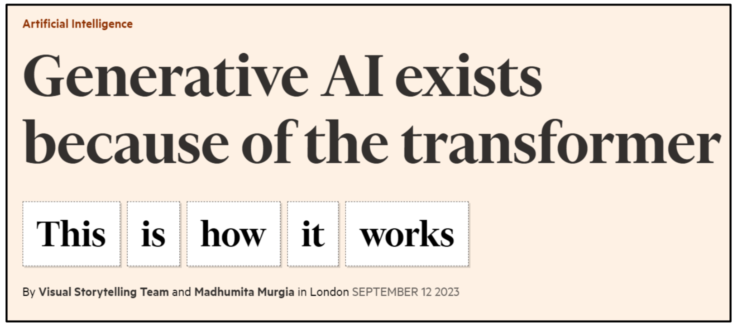
- https://jalammar.github.io/illustrated-transformer/
- https://jalammar.github.io/illustrated-gpt2/
- https://jalammar.github.io/illustrated-bert/
- https://jalammar.github.io/how-gpt3-works-visualizations-animations/

- CS224n
  - Transformers, Self Attention https://www.youtube.com/watch?v=5vcj8kSwBCY

References

- Generative AI exists because of transformer

  Financial Times



Artificial Intelligence

Generative AI exists because of the transformer

This is how it works

By Visual Storytelling Team and Madhumita Murgia in London SEPTEMBER 12 2023

- YouTube
  - Visual Guide to Transformer Neural Networks - (Episode 1) Position Embeddings
  - How large language models work, a visual intro to transformers | Chapter 5, Deep Learning (27 mins)
  - Attention in transformers, visually explained | Chapter 6, Deep Learning (26 mins)
  - The matrix math behind transformer neural networks, one step at a time!!! (23 mins)
  - Transformer Neural Networks, ChatGPT's foundation, Clearly Explained!!! (36 mins)

- Transformer Explainer

  https://poloclub.github.io/transformer-explainer/

References

- Large language models, explained with a minimum of math and jargon

  https://www.understandingai.org/p/large-language-models-explained-with?utm_source=multiple-personal-recommendations-email&utm_medium=email&triedRedirect=true

Thank you!