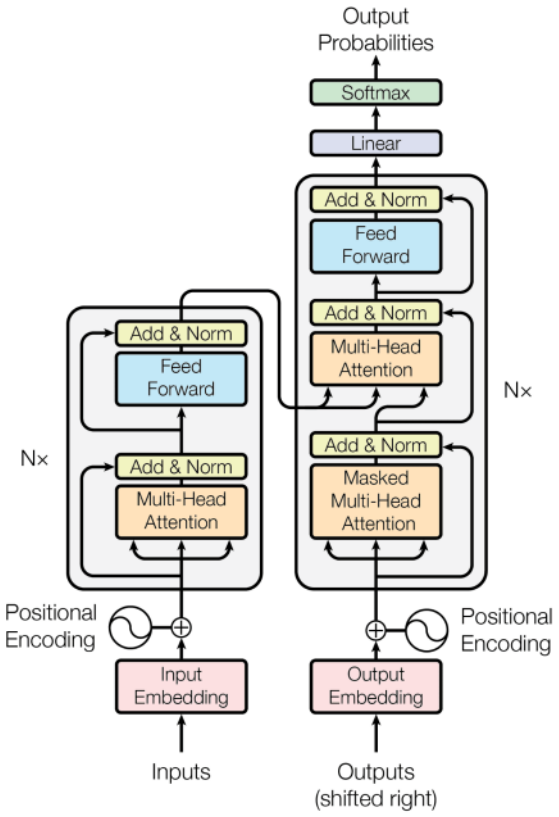
Transformer

A transformer is a type of deep learning model that has revolutionized the field of natural language processing (NLP). Introduced in the paper "Attention is All You Need" . in 2017, transformers rely on a mechanism called **self-attention** to process input data in **parallel**, rather than sequentially, which allows them to handle longer dependencies in the data more effectively than previous models like recurrent neural networks (RNNs).



ANN - Tabular data
CNN - Image data
RNN - Sequence to Sequence (text data)

Whereas Transformer is used for

Seq2Seq : Input is also Sequence and output is also Sequence task such as:

Machine translation
Question answer
Text summarization

Impact of Transformer Models

1. **Natural Language Processing (NLP)**
2. **Computer Vision**
3. **Scientific Research & Medicine**
4. **Code Generation & Software Development**
5. **Search Engines & Information Retrieval**
6. **Finance & Business Analytics**
7. **Education & Personalized Learning**
8. **Creative AI & Content Generation**
9. **Robotics & Autonomous Systems**
10. **Ethical Considerations & Challenges**

Part-2

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

Impact on the field of natural language processing (NLP) and deep learning

Big problem of NLP is solved using this.

Democratizing of the AI

1- Anyone can use their purpose as foundational model
2- Fine tune it for making domain specific
3- EX- BERT GPT …user can fine tune them on its external dataset for domain specific work

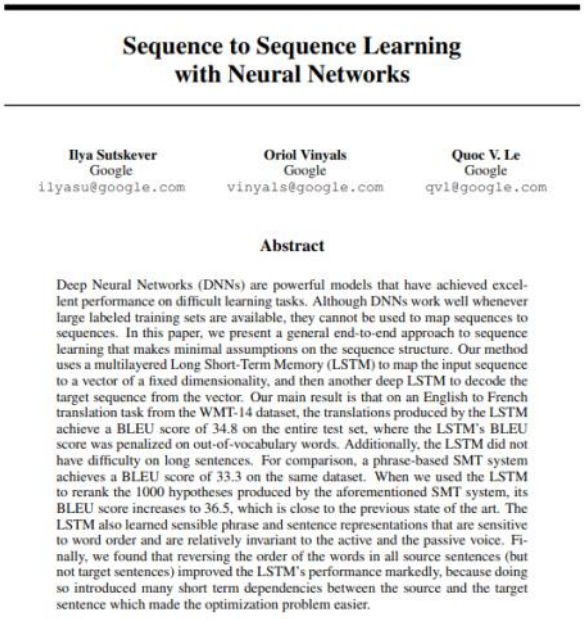
Cross-domain Applications: Beyond NLP, transformers have found applications in computer vision, audio processing, drug discovery, and more.

Scalability: Transformers are highly scalable, capable of handling large datasets and computations efficiently

Research Advancements: The development of transformers has spurred research in improving model architectures, training techniques, and understanding of deep learning principles

Part-3

Sequence to Sequence Learning with Neural Networks :



Sequence to Sequence Learning with Neural Networks

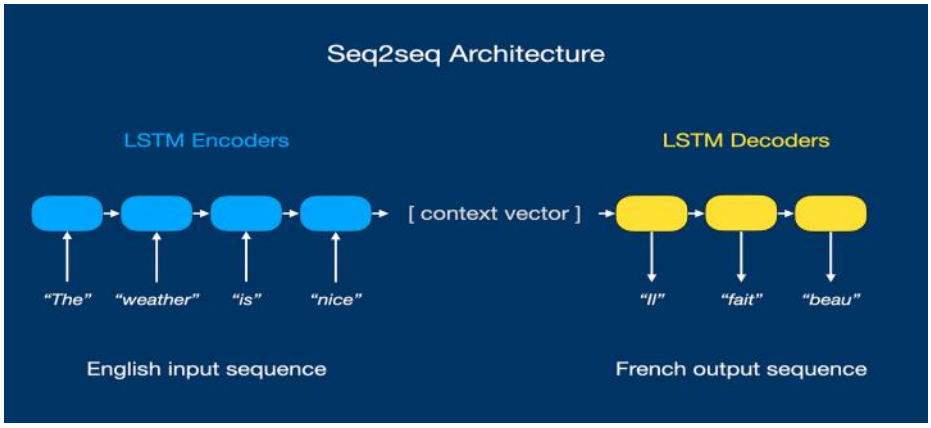
Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT-14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous state of the art. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.



To solve the Seq2Seq problem it came up with the architecture called Seq2Seq Architecture and It consist of two parts Encoder and Decoder with LSTM network in both the network.

Fails because it can't store all the values in the context vector for long sentences. It was working good for the small sentence in translation process.

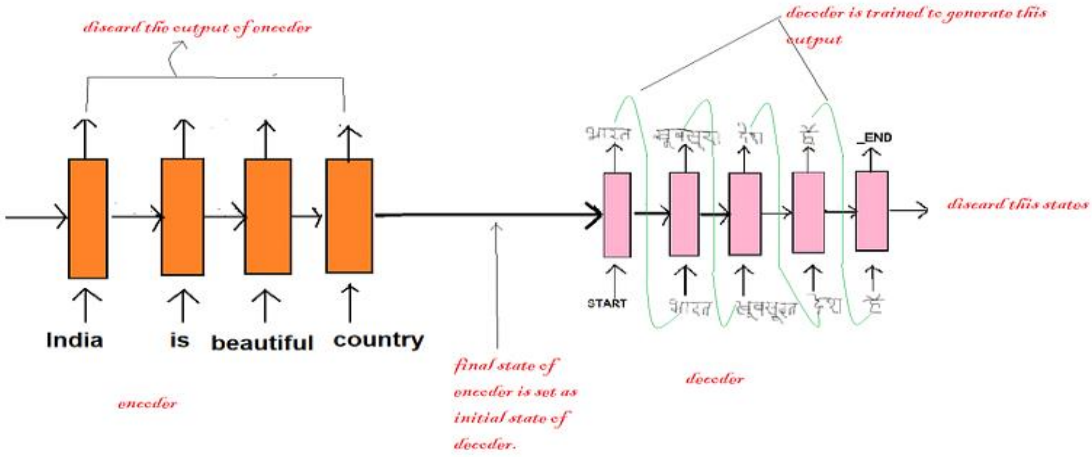
NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.



Concept of **attention was proposed to solve above problem.**

Encoder was working in the same way but in decoder it was proposed that it is not necessary to send the whole sentence in the one go….you can send it in the fraction (context vector)….

This time decoder tells the encoder which word will be influence most by the word that will be printed in decoder part…This is called attention mechanism.

Dynamic Focus:

Attention mechanisms allow the decoder to dynamically focus on different parts of the input sequence at each step of decoding. This improves the model's ability to handle longer sequences and capture dependencies more effectively.

Don’t Need to send full sentence at a time.
Send the input in the sequential manner by using attention mechanism.

Drawback:

Slow as it is sequential i.e., One - One at a time So can't train model in less time.
Transfer learning was not possible.

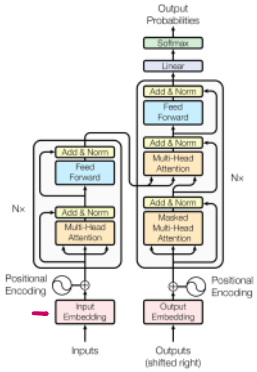
Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Lukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* † illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

RNN | LSTM = X
SELF -ATTENTION + PARALLEL PROCESSING



Disadvantages

High Computational Costs:
Energy Consumption:
Data Requirements:
Model Size and Complexity:
Bias and Fairness:
Bias due to lack of diversity in data
Security and Privacy:
Scalability Issues:
Ethical and Social Implications: Misinformation: Deep fake , fake news and spam | Job displacement

Future of Transformer

Efficiency Improvements: Model Compression: pruning, quantization, and knowledge distillation
Scaling Up:
Cross-Modal Applications: Vision Transformers (ViT) |Multimodal Transformers:
Domain-Specific Transformers: healthcare, finance, legal, and scientific research to improve accuracy and relevance in these areas.
Ethics and Fairness: Bias Mitigation: Developing methods to identify and reduce biases
Research and Development:
Interactive AI and Conversational Agents:
Integration with Other Technologies:

NLP --> WORDS -->NUMBERS -->VECTORIZATION

1. **One Hot Encoding** : Unique categories: ["Red", "Blue", "Green"]
   Vectorization:

   Red: [1, 0, 0]
   Blue: [0, 1, 0]
   Green: [0, 0, 1]

2. **Bags of words** : check word is present and it also check no of times it present in the sentence.
   Document 1: "i love machine learning"
   Document 2: "machine learning is fun"

   Unique words: ["i", "love", "machine", "learning", "is", "fun"]
   Vectorization:
   Document 1: [1, 1, 1, 1, 0, 0]
   Document 2: [0, 0, 1, 1, 1, 1]

3. **Term Frequency-Inverse Document Frequency (TF-IDF)**

4. Word Embeddings (new concept)

Word Embedding is a technique in natural language processing (NLP) that represents words in continuous vector space,
capturing their meanings, semantic relationships, and syntactic roles.
Unlike traditional methods like
Bag of Words (BoW) and TF-IDF, word embeddings create dense,
low-dimensional, and **fixed-length vectors** for each word,
where similar words are closer in the vector space.

Common methods:
Word2Vec:
GloVe (Global Vectors for Word Representation):
FastText:
BERT (Bidirectional Encoder Representations from Transformers):

This powerful property of **word embeddings** enhances the capability of NLP models to understand and process human language.

Conversion of word to vector

- An apple a day keeps the doctor away
- Apple is healthy
- Apple is better than orange
- Apple makes great phones

X=TASTE
Y=TECHNOLOGY

[0.8 , 0.2] - Static Embedding
[X, Y]
Word Embedding aye made only once so they are static.

Now taking a fresh sentence :

Apple launched a new phone while I was eating an apple.

Here its fails due to static embedding . To avoid this we should have **contextual embedding** (values changes dynamically according to situation)

Apple launched a new phone while I was eating an apple. ---->

□□ □□□ □□□ □□ □□□ □□, □□ □□□ □□ □□ □□□ □□□ □□□□□ □□ □□□□

To avoid this we use **self -attention mechanism**

□□ □□□ □□□ □□ □□□ □□, □□ □□□□□ □□ □□□ □□□ □□□□□ □□ □□□□

Self-Attention Mechanism

Self-attention, also known as scaled dot-product attention is for calculating the similarity scores , allows the model to weigh the importance of different tokens in the input sequence. This is achieved using query, key, and value matrices.

Self-attention layer helps us to understand between words in the sentence

Apple launched a new phone while I was eating an apple

Definitions

- **Query (Q)**: is like a word looking for other words to pay attention.
- **Key (K)**: is like a word being looked by other words.
- Key can be thought of as the pieces of information that help answer the questions posed by the Query.
- **Value (V)**: Value represents the actual information you end up using after figuring out the importance (attention) of each word.

Why "Self-Attention"?

- **Self:** Each word uses its own information (query, key, value) and interacts with all words in the same sequence, including itself.
- **Attention:** The mechanism computes attention weights that determine how much focus each word should give to every other word in the sequence to generate context-aware representations.

Apple launched a new phone while I was eating an apple.

Apple launched a new phone while I was eating an apple.

Apple launched a new phone while I was eating an apple.

□□ □□□ □□□ □□ □□□ □□, □□ □□□□□ □□ □□□ □□□ □□□□□ □□ □□□□

Mathematical Formulation

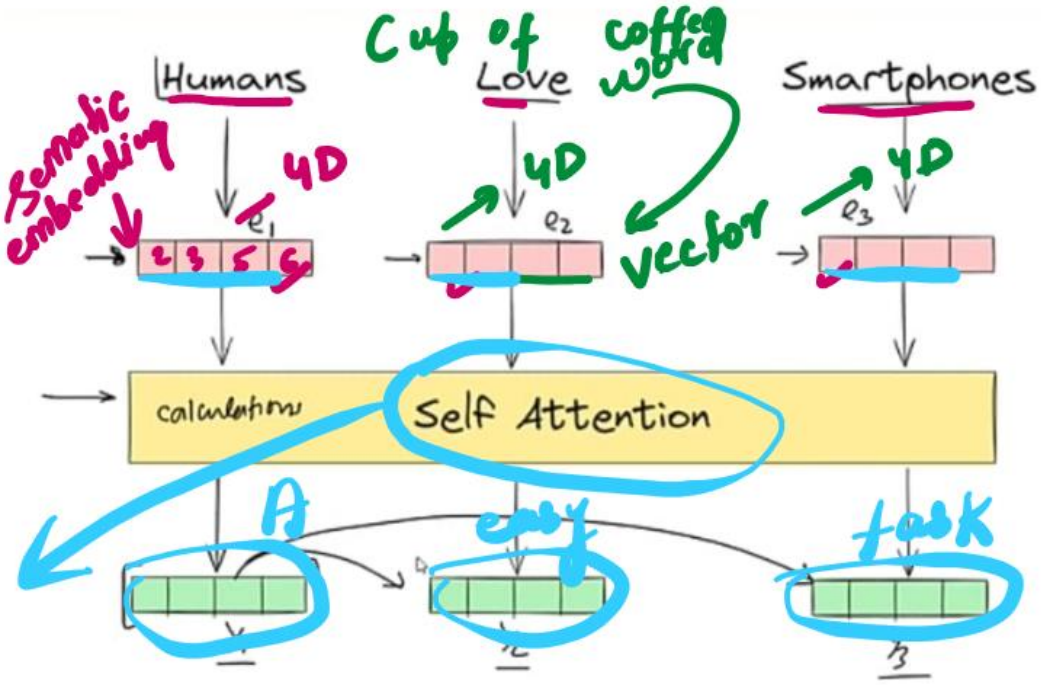
1. **Computing Attention Scores:**

   - The attention scores are calculated using the dot product of the query and key vectors.
   - These scores are then scaled by the square root of the dimensionality of the key vectors (to stabilize gradients) and passed through a softmax function to obtain the attention weights.

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

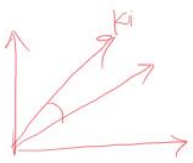
   Where:

   - Q is the matrix of queries.
   - K is the matrix of keys.
   - V is the matrix of values.
   - d_k is the dimension of the key vectors.

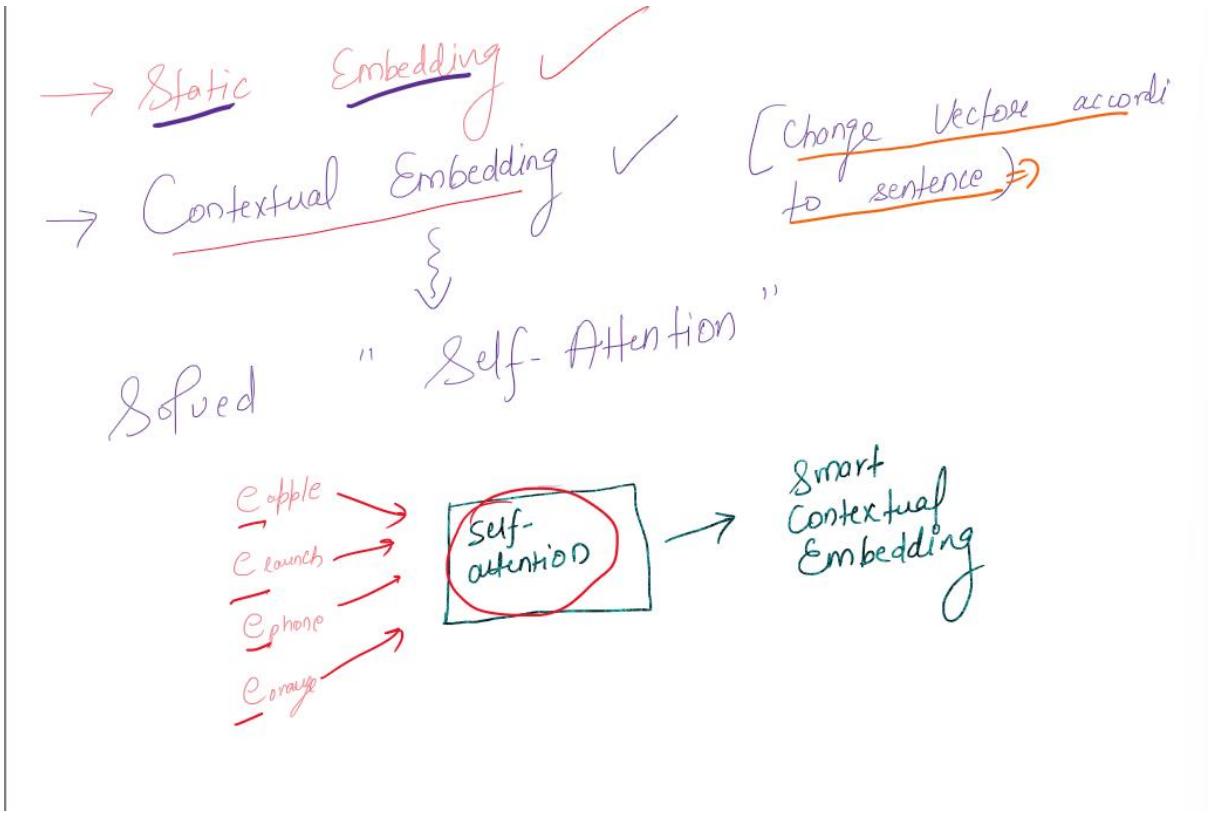


(1) OHE
(2) Bag of Words
(3) Idf ⇒ frequency
(4) Word - Embedding – Semantic meaning (capture).

→ If two words are same then their vectors will b also be same



In Vectors every word will have some meani in their Vector Representation.



In a Transformer model, the **weight matrices** are learnable parameters that transform input vectors into **Query (Q), Key (K), and Value (V)** representations. These matrices define how the model attends to different parts of the input sequence.

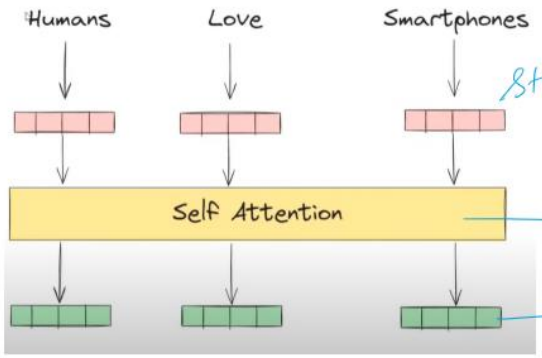
Weight Matrices in Attention Mechanism

For a **model dimension** d and **number of attention heads** h, the weight matrices are:

1. **Query Weight Matrix** W_q
   - Shape: (d, d)
   - Transforms input into the **query** vector $Q = XW_q$
2. **Key Weight Matrix** W_k
   - Shape: (d, d)
   - Transforms input into the **key** vector $K = XW_k$
3. **Value Weight Matrix** W_v
   - Shape: (d, d)
   - Transforms input into the **value** vector $V = XW_v$

Each of these weight matrices is learned during training.

*. Embedding should dynamic change.

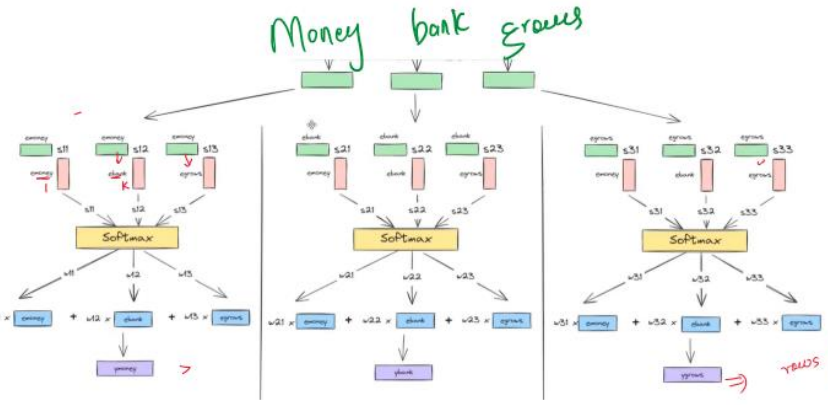


Static Embedding

self-attention

contextual

Keeping money in banks grows by 8%.



$K = I \cdot W_{matrix}$

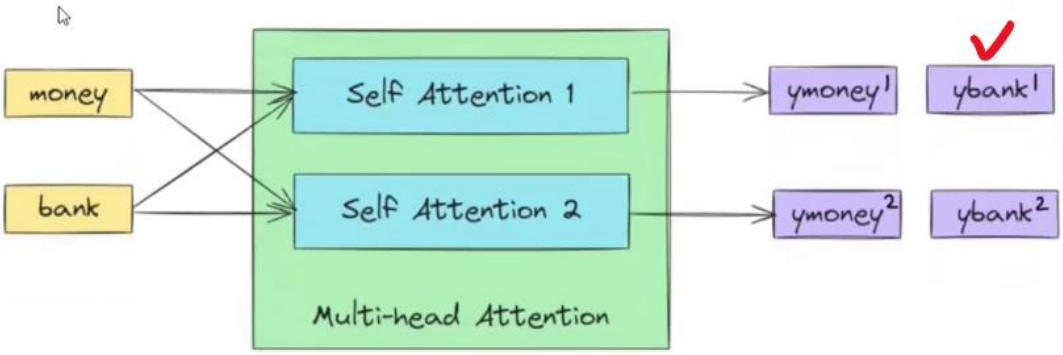
$Q = m \cdot W_{matrix}$

$V = n \cdot W_{matrix}$

$\begin{pmatrix} 2 & 3 \\ 4 & 1 \end{pmatrix}$

The man saw the astronomer with a telescope

	The	man	saw	the	astronomer	with	a	telescope
The								
man								
saw								
the								
astronomer								
with								
a								
telescope								



① We cannot capture the word order because we are sending them in one go.

→ So here comes Positional Embedding.

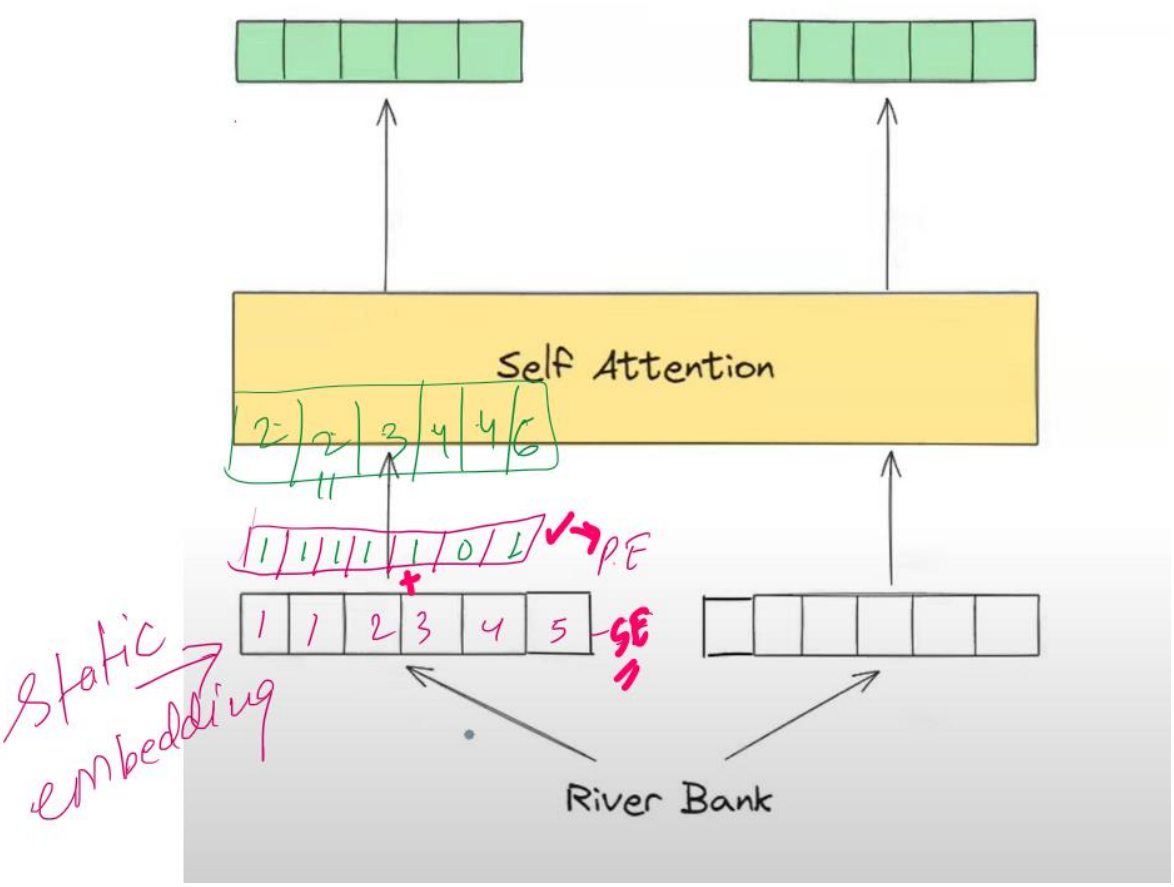
Introduction

In transformer models (e.g., BERT, GPT), positional embeddings are used to encode the order of words in a sentence since these models do not have inherent sequential processing like RNNs. One popular method for positional encoding is using sine and cosine functions.

The formula for positional encoding at position pos and dimension i is:

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

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


Self Attention
PE
Static embedding
River Bank

Normalization in DL refers to the process of transforming data or model outputs to have specific properties, typically a mean of zero or variance of one.

⚙️ **What is Layer Normalization?**

Layer Normalization (often written as LayerNorm) is used to **stabilize and speed up** training in deep neural networks like Transformers.

🔍 **What it does:**

It **normalizes the input** across the features **for each data point individually**, unlike BatchNorm which normalizes across the batch.

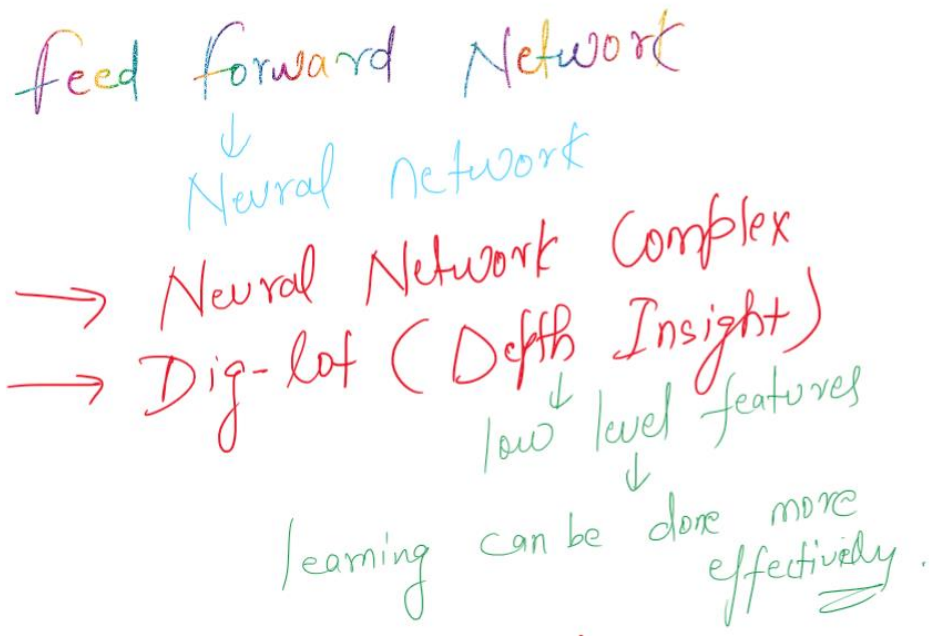
Why is it important in Transformers?

Transformers rely heavily on:

- **Multi-head self-attention**
- **Feed-forward networks**

To maintain stable training and avoid exploding or vanishing gradients in these deep stacks, **LayerNorm** is used:

- After **self-attention** layers
- After **feed-forward** layers
- Usually combined with **residual connections**



sentence: “I love ice cream.”

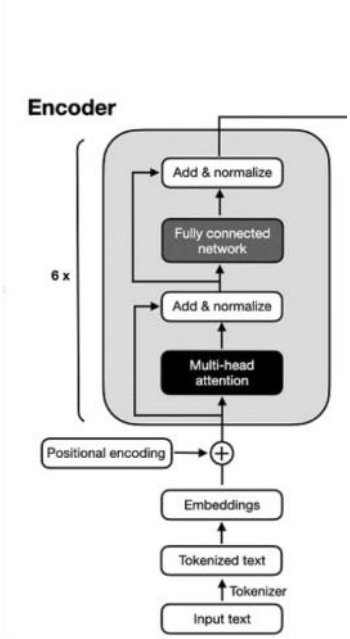
1. Input

The input to the encoder is the sentence “**I love ice cream**.”

2. Tokenization

The sentence is split into smaller parts called tokens. These can be words or subwords.

```
Tokens: [I, love, ice, cream]
```



3. Embedding

Each token is converted into a numerical representation, typically a vector of fixed size. This vector captures the semantic meaning of the word.

```
Embeddings:

[I -> [0.1, 0.3, 0.5, ...],
love -> [0.7, 0.2, 0.8, ...],
 ice -> [0.4, 0.9, 0.6, ...],
 cream -> [0.3, 0.1, 0.4, ...]
]
```

4. Positional Encoding

Since the order of words is important, we add positional encoding to each embedding. Positional encoding provides information about the position of the word in the sentence using sine and cosine functions.

```
Positional Embeddings:

[0, 1, 2, 3]


Combined Embeddings:

[I -> [0.1, 0.3, 0.5, ...] + [PE0],
 love -> [0.7, 0.2, 0.8, ...] + [PE1],
 ice -> [0.4, 0.9, 0.6, ...] + [PE2],
cream -> [0.3, 0.1, 0.4, ...] + [PE3]]
```

5. Self-Attention Mechanism

The self-attention mechanism allows each word to focus on other words in the sentence to understand context better. Here's how it works:

5.1. Creating Query, Key, and Value Matrices

For each token, we create three vectors: Query (Q), Key (K), and Value (V) by multiplying the embedding with learned weight matrices.

```
Q = Embedding * W_Q
K = Embedding * W_K
V = Embedding * W_V
```

5.2. Calculating Attention Scores

We calculate attention scores for each word by taking the dot product of the Query vector of one word with the Key vectors of all words, followed by scaling and applying a softmax function to get probabilities.

```
Attention Score for word_i and word_j = softmax((Q_i * K_j)
/ √d_k)
```

5.3. Weighted Sum of Values

The attention scores are used to create a weighted sum of the Value vectors, giving more importance to certain words.

```
Weighted Sum for word_i = Σ(Attention Score_ij * V_j)
```

6. Multi-Head Attention

Instead of having a single set of Q, K, V matrices, we have multiple sets (heads). Each head captures different aspects of the relationships between words.

```
Multi-Head Output for word_i = [Head1_output, Head2
_output, ..., HeadN_output]
```

These outputs are concatenated and projected using another weight matrix.

7. Add & Normalize

The output from the multi-head attention is added to the original embeddings (**residual connection**) and **normalized**.

```
Norm1(word_i) = LayerNorm(Multi-Head Output + Original
Embedding)
```

8. Feed-Forward Neural Network

Each normalized vector is passed through a feed-forward neural network which typically consists of two linear transformations with a ReLU activation in between.

```
FFN Output(word_i) = FFN(Norm1(word_i))
```

9. Add & Normalize

Again, we add the feed-forward output back to the input of the feed-forward network (residual connection) and normalize.

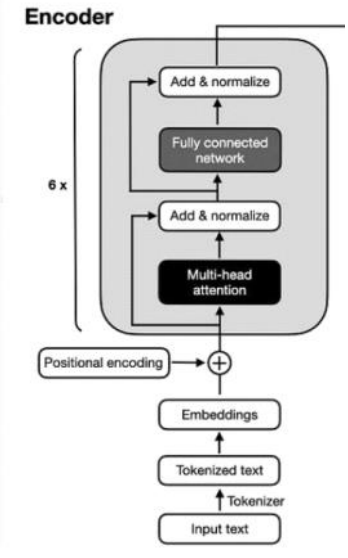
```
Norm2(word_i) = LayerNorm(FFN Output + Norm1(word_i))
```

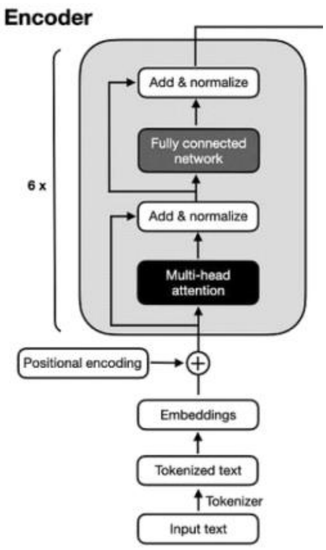
10. Output

The encoder's output is a set of vectors (one for each word), which captures the meaning of each word in the context of the entire sentence. These vectors are passed to the decoder for further processing.

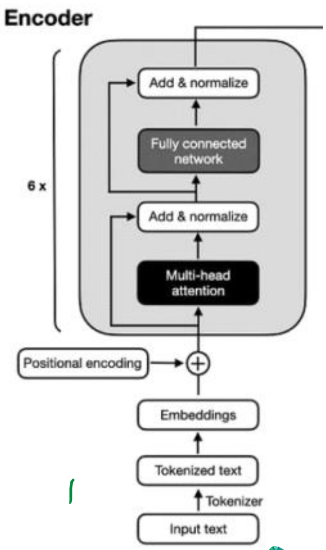
```
Encoder Output:
[word1_vector, word2_vector, word3_vector, word4_vector]
```

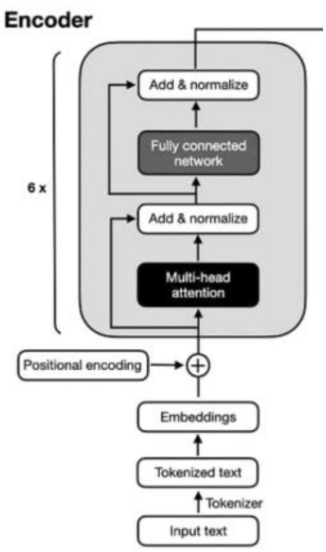
The encoder transforms the sentence “I love ice cream” into a series of context-aware vectors that represent the input in a way that a computer can understand and use for various tasks such as translation, summarization, or answering questions.

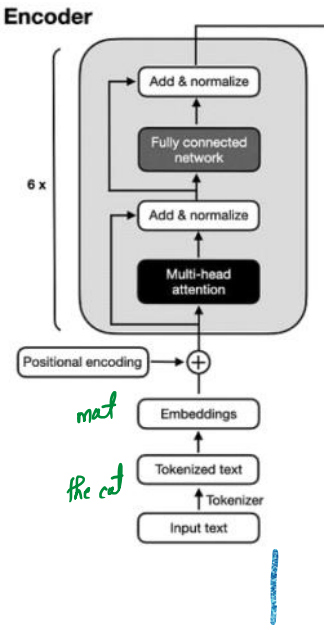


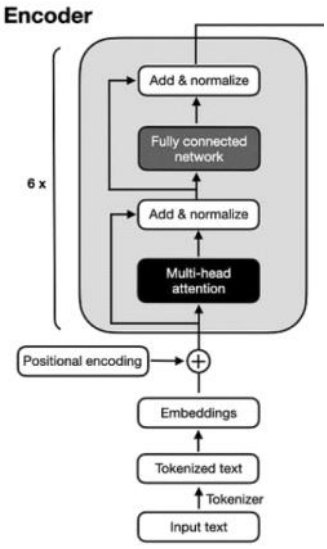

Encoder
6 x
Add & normalize
Fully connected network
Add & normalize
Multi-head attention
Positional encoding
Embeddings
Tokenized text
Tokenizer
Input text

6x


Encoder
6 x
Add & normalize
Fully connected network
Add & normalize
Multi-head attention
Positional encoding
Embeddings
Tokenized text
Tokenizer
Input text


Encoder
6 x
Add & normalize
Fully connected network
Add & normalize
Multi-head attention
Positional encoding
Embeddings
Tokenized text
Tokenizer
Input text





Only one Encoder work


Only one word
Embedding
Contextual