Statistical Machine Learning

TRANSFORMER ARCHITECTURE

Students

20127449 – Trần Quốc Bảo

20127452 – Hồ Đăng Cao

20127476 – Đỗ Đức Duy

3/2/2024

Table of Contents

I. Transformer architecture model 3

1) What is Transformer? 3
2) Comparison to the old models 3
3) Key components of the transformer architecture 5
4) Encoder 6

Scaling the Scores 11

5) Decoder 13

Cross-Attention: Bridging the Encoder and Decoder 13

So, why is this important? 14

6) Linear & softmax 16
7) Pros and Cons 16

II. Machine Translation 17

1) Introduce 17
2) Data collection 18
3) Data analysis 18
4) Create training set 20
5) Training 21
6) Special Techniques for Transformer Training 21
7) Beam search 22
8) Evaluation (use BLEU score) 23

References 24

I. Transformer architecture model

1) What is Transformer?

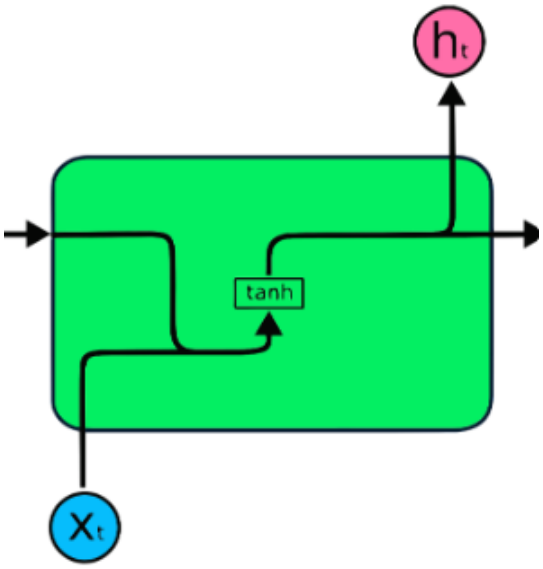
Transformer is a specific type of deep learning model architecture particularly in natural language processing (NLP) tasks. The transformer architecture was introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017, and it revolutionized the way researchers approach sequence-to-sequence tasks, such as language translation, text generation, and more.

The transformer architecture is characterized by its innovative use of self-attention mechanisms, which allow the model to weigh the importance of different words in a sequence when processing them. This self-attention mechanism enables the transformer to capture contextual relationships between words, leading to improved performance in handling tasks involving sequences of data, like sentences or paragraphs.

2) Comparison to the old models

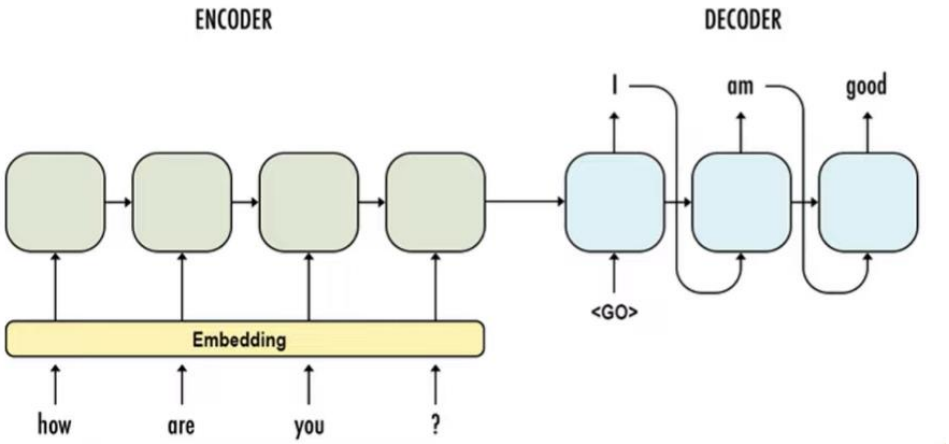
Some famous models before:

Recurrent Neural Networks (RNNs) are neural networks used to model sequential data such as word sequences in a sentence. They perform the same task for each element of a sequence, with the output at each element depending on the previous element. This helps capture the sequential dependency relationships within the sequence. RNNs have been widely applied in natural language processing, for example, in machine translation using the sequence-to-sequence model with RNNs for encoding and decoding. RNNs are also used for stock market prediction.

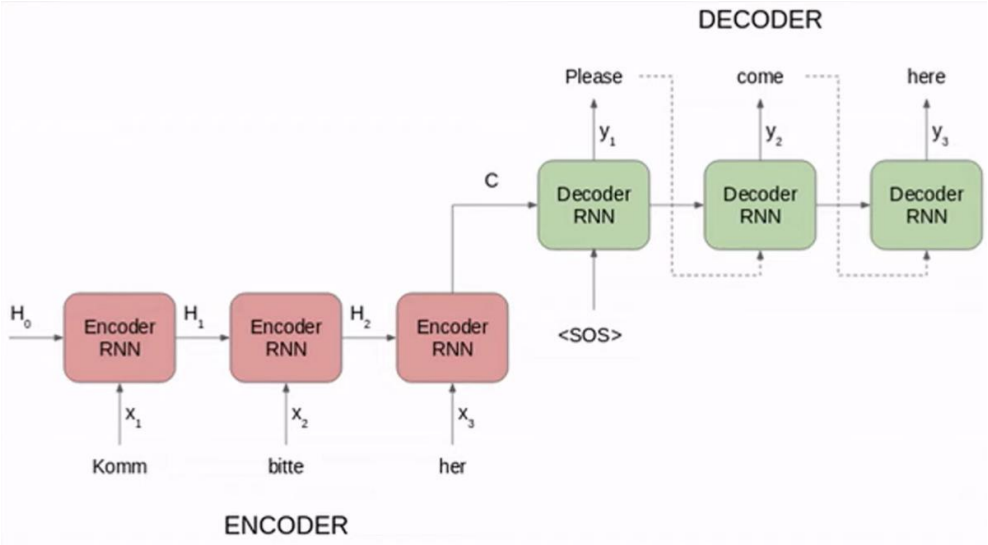


However, RNNs have some weaknesses:

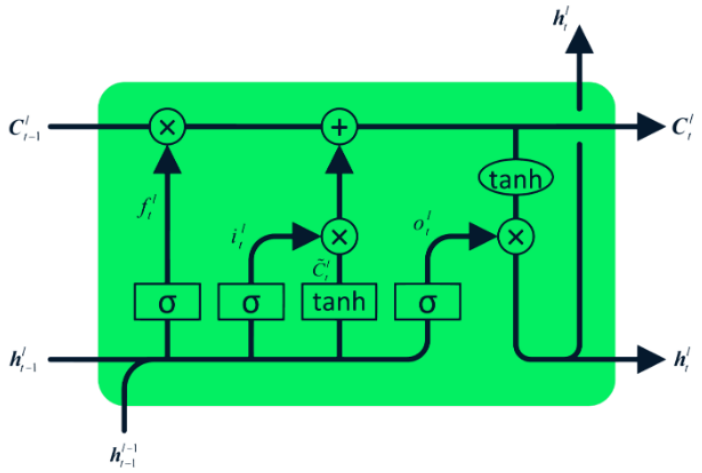
- Slow training speed, even when using Truncated Backpropagation.



- Inefficient processing with long sequences due to the Gradient Vanishing problem. As the number of words increases, the number of units also increases, leading to a gradual decrease in gradient at the later units due to the chain rule, causing the loss of information about long-range dependencies between units.



To address the Gradient Vanishing issue, the Long Short-Term Memory (LSTM) model was developed. However, LSTM is more complex and has slower training speed compared to RNNs, affecting the development of natural language processing.



The emergence and mission of the **Transformer** model in the NLP field:

The **Transformer** is a deep learning model used for various language and speech processing tasks such as automatic translation, language generation, classification, entity recognition, speech recognition, and text-to-speech conversion.

Unlike RNNs, the Transformer doesn't process elements in a sequence. With its **self-attention** capability, it efficiently understands relationships between words in a sentence, utilizing the parallel computation power of GPUs and speeding up processing.

The Transformer consists of 6 encoder layers and 6 decoder layers. Each encoder layer contains self-attention and a feedforward neural network (FNN), while the decoder includes an attention layer to focus on relevant parts of the input.

3) Key components of the transformer architecture

1. Encoder: A phrase that converts input into learning features capable of learning tasks (digitizing). The result is embedded vectors. The output of the encoder is the input of the decoder.

2. Decoder: This phrase aims to find out the probability distribution from the learning features in the Encoder and then determine what is the label of the output. The result can be a label for categorical models or a chronological sequence of labels for model seq2seq.

3. Positional Encoding: Since transformers do not have an inherent sense of word order, positional encodings are added to the input embeddings to provide information about the position of each word in the sequence.

4. Self-Attention Mechanism: allows the model to consider the relationships between all words in a sequence simultaneously, rather than sequentially. It computes weighted representations of each word based on its relationships with other words in the input sequence.

5. Multi-Head Attention: The self-attention mechanism is used multiple times, with different linear projections of the input. This enables the model to focus on different aspects of the data and learn various types of relationships.

6. Feed-Forward Neural Networks: Each encoder and decoder layer include feed-forward neural networks that apply pointwise fully connected layers to the representations.

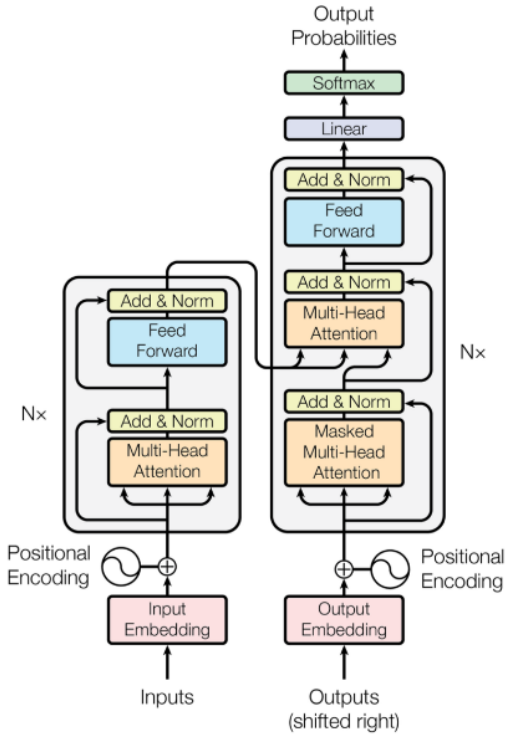


Figure 1: The Transformer - model architecture.

This architecture consists of 2 parts: encoder on the left and decoder on the right.

Note: In Machine Translation (MT), Encoder is used to learn the expression vector of a sentence with the expectation that this vector carries the perfect information of that sentence. Decoder performs the function of converting the other representation vector into the target language.

4) Encoder

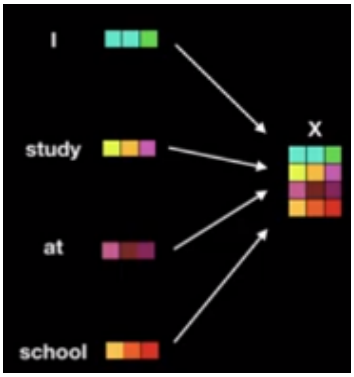
Encoder consists of 6 encoding layers, each layer consists of 2 sublayers. The first sublayer is the multi-head self attention, which allows words to interact to create improved representations. The second sub-layer is the fully-connected feed-forward layer, which adds complexity to the words. Each sub-layer has residual connections and normalization for information transfer and network stability.

Figure 1:Encoder process with 6 layers

a. Input Embedding Matrix

Each word is represented by a vector of equal dimension. The words in the sentence are then joined together into rows of a 2-dimensional matrix $D \in R^{m \times n}$ containing the semantics of each individual word, where m is the sentence length and n is the dimensionality of a word embedding vector (n is $d_{model} = 512$ in the paper).

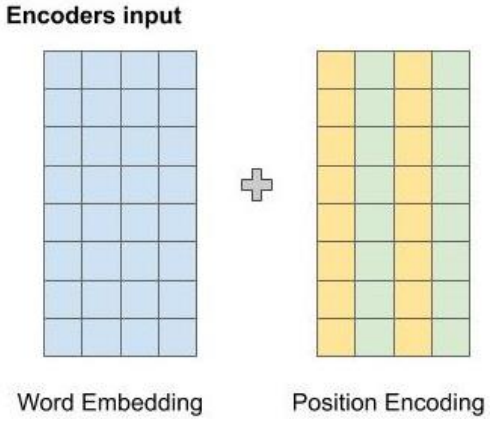
e.g.,



b. Positional Encoding

Positional Encoding (PE) is added to the X input matrix of the encoder and decoder (2 matrices of the same size) to encode the position of the word in the sentence.

e.g.,



We use sin and cos functions for PE:

$$PE(pos,k) = \{sin\ sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right)\ cos\ cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad if\ k = 2i \quad if\ k$$

$$= 2i + 1 \quad i \in N$$

where pos is the position of the word and k is each dimension of the position vector.

e.g., given 2 words and word embedding has 6 dimensions.

	0	1	2	3	4	5
0	$\sin(\frac{1}{1000^{0/512}} \times 0)$	$\cos(\frac{1}{1000^{0/512}} \times 0)$	$\sin(\frac{1}{1000^{2/512}} \times 0)$	$\cos(\frac{1}{1000^{2/512}} \times 0)$	$\sin(\frac{1}{1000^{4/512}} \times 0)$	$\cos(\frac{1}{1000^{4/512}} \times 0)$
1	$\sin(\frac{1}{1000^{0/512}} \times 1)$	$\cos(\frac{1}{1000^{0/512}} \times 1)$	$\sin(\frac{1}{1000^{2/512}} \times 1)$	$\cos(\frac{1}{1000^{2/512}} \times 1)$	$\sin(\frac{1}{1000^{4/512}} \times 1)$	$\cos(\frac{1}{1000^{4/512}} \times 1)$
2	$\sin(\frac{1}{1000^{0/512}} \times 2)$	$\cos(\frac{1}{1000^{0/512}} \times 2)$	$\sin(\frac{1}{1000^{2/512}} \times 2)$	$\cos(\frac{1}{1000^{2/512}} \times 2)$	$\sin(\frac{1}{1000^{4/512}} \times 2)$	$\cos(\frac{1}{1000^{4/512}} \times 2)$

Features of using PE:

- Embedding values are normalized because the sine and cosine functions have values in the range [-1,1]

- Words are embedded as binary number representation, i.e. first dimensions (first bits) will change faster than final dimensions (last bits).

Explain: $\frac{1}{10000^{\frac{2i}{d_{model}}}}$ is smaller as i gets bigger, so $\frac{pos}{10000^{\frac{2i}{d_{model}}}}$ grows more slowly as pos increases in large dimensions.

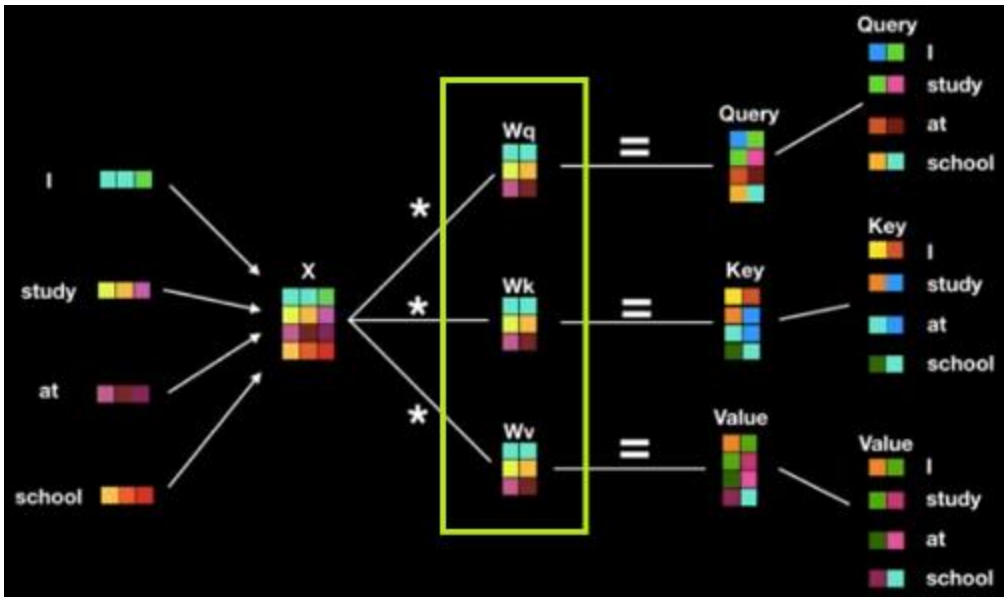
After adding PE to X, we come to the self-attention mechanism.

c. Self-Attention - The first sub-layer

Self-attention meaning: each word can adjust its weight for other words in the sentence so that the closer the word is closest to its meaning, the greater the weight, and the further away the meaning, the smaller the weight.

e.g., In the sentence: "I have a flower and it is very beautiful", the word 'it' refers to the word 'flower', so the output of the embedding vector of the word 'it' after self attention must be close to the embedding vector of the word 'flower'.

Self-attention mechanism:



Step 1: Generate randomly 3 weight matrices Wq, Wk and Wv, then multiply X with these weight matrices to generate 3 matrices Query (Q), Key (K) and Value (V). Where:

Wq, Wk and Wv are the weights that the model needs to train.

Query: contains vectors, each vector used to contain information of the searched and compared word. Like a google search query.

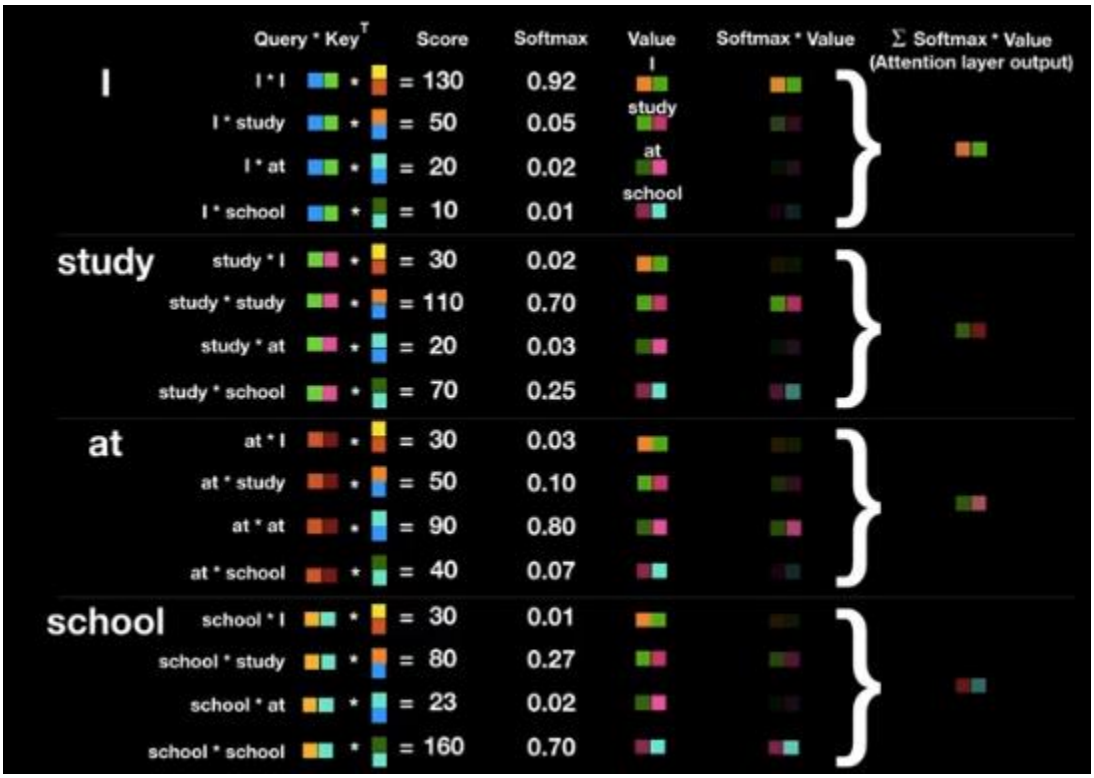
Key: contains vectors, each vector used to represent information about the words compared with the word to be searched above. Like the web pages that google will compare with the keywords you search for.

Value: contains vectors, each vector represents the content and meaning of words. Like the web page content that is displayed to the user after a search.

- **Queries** (Q): This matrix is the word we're **currently focusing** on.
- **Keys** (K): This matrix represents **all the words** in the document we're comparing our query to.
- **Values** (V): This matrix holds the **actual context** for each word in the input document.

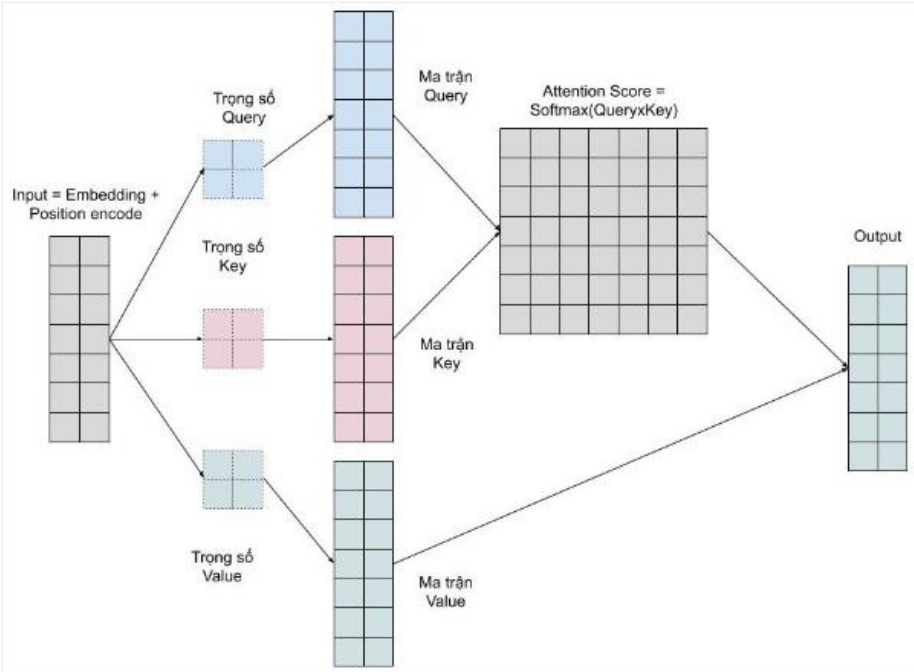
Step 2: Multiply the Query matrix by the Key matrix to get the attention weights, which contains the correlation coefficient between each word wQ and the words wK. Then the coefficients are normalized to the interval [0-1] using the softmax function. The larger the coefficient, the higher the probability that there is a relationship between the word pair (wQ, wK).

Step 3: The output attention matrix will be calculated based on the weighted average of the value vectors in the Value matrix by multiplying the attention weights matrix.



Comment: the output attention vector of each word is closest to its own Value vector.

Summarize the self-attention process:



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

Divide by $d_k = d_v = \frac{d_{model}}{h}$ (the number of dimensions of the Key vector) to avoid pushing the softmax function into regions where it has extremely small gradients if the QK^T value is too large. h is the number of heads.

Scaling the Scores

(**Important**) While dealing with high-dimensionality matrices, the dot product leads to very ***large variance***. This can cause instability in the model.

Here's why:

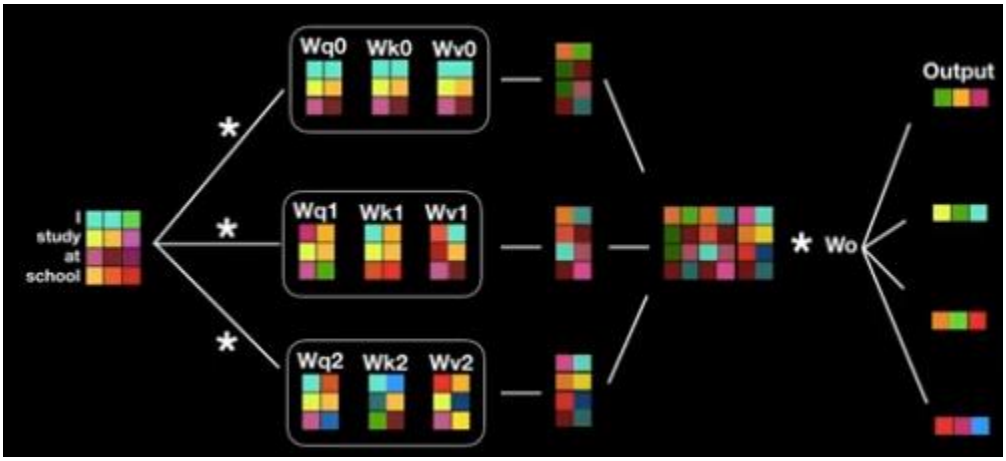
During backpropagation, the softmax function would produce extremely large gradients, for large dot product values. This results in ***negligible updates of smaller dot product values during the training process,*** *due to small gradients, leading the model towards the* ***vanishing gradient problem****.*

Therefore, to mitigate this problem, the **dot products** are scaled by a square root of the ***dimensionality of Key vectors*** i.e.,$\sqrt{d_k}$.

This is why it is known as **Scaled-Dot Product Attention**.

d. Multi-head attention

Multi-head attention is a collection of many self-attention heads to learn many other word relationships.



After getting the attention matrices in the output, we will concatenate these matrices horizontally to obtain a multi-head matrix. Then project the multi-head matrix onto the same space as the input matrix by multiplying by the matrix W_0.

$$MultiHead(Q, K, V) = concatenate(head_1, head_2, \dots, head_h)W_0$$

Where $head_i = Attention(Q_i, K_i, V_i)$

e. Feed Forward - The second sub-layer

In this sub-layer, like many other models, we will go through the fully connected (FC) network and return the output with the same shape as the input. Thanks to that, we can go through the encoder layer 6 times.

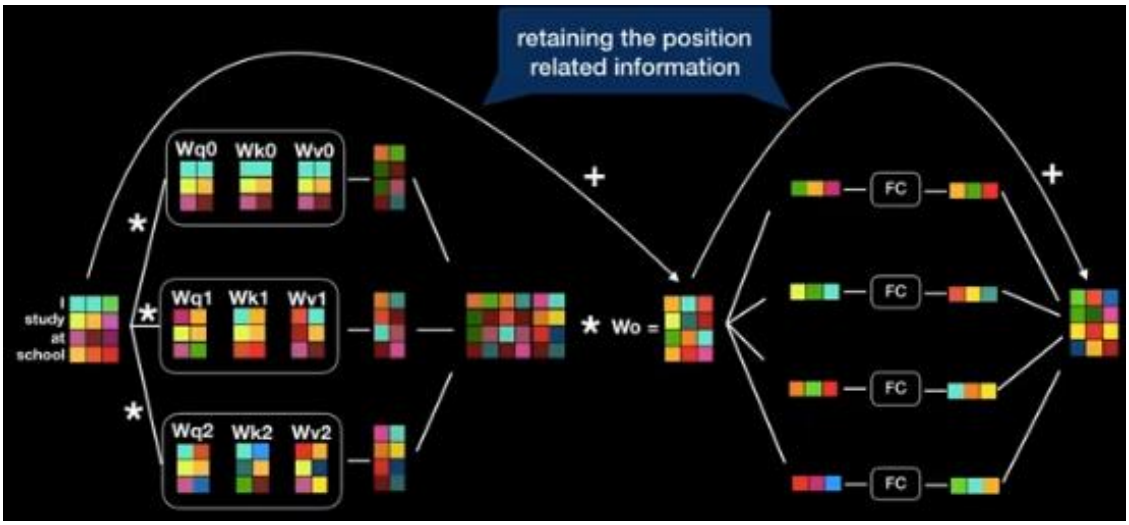


Figure 2:Diagram of a layer encoder

$$FFN(x) = (0, xW_1 + b_1)\, W_2 + b_2$$

At output matrices of each sub-layer, to retain the position and related information, we implement **Add & Norm**.

f. Add & Norm

Add (Residual Connection): This operation helps in mitigating the vanishing gradient problem and enables the network to learn both the original input representation (e.g., keep position information) and the transformation applied to it simultaneously.

Norm (Layer Normalization): The Normalization layer is applied immediately after the 'Add' so that the training process is stable and converges faster. e.g., using z-score

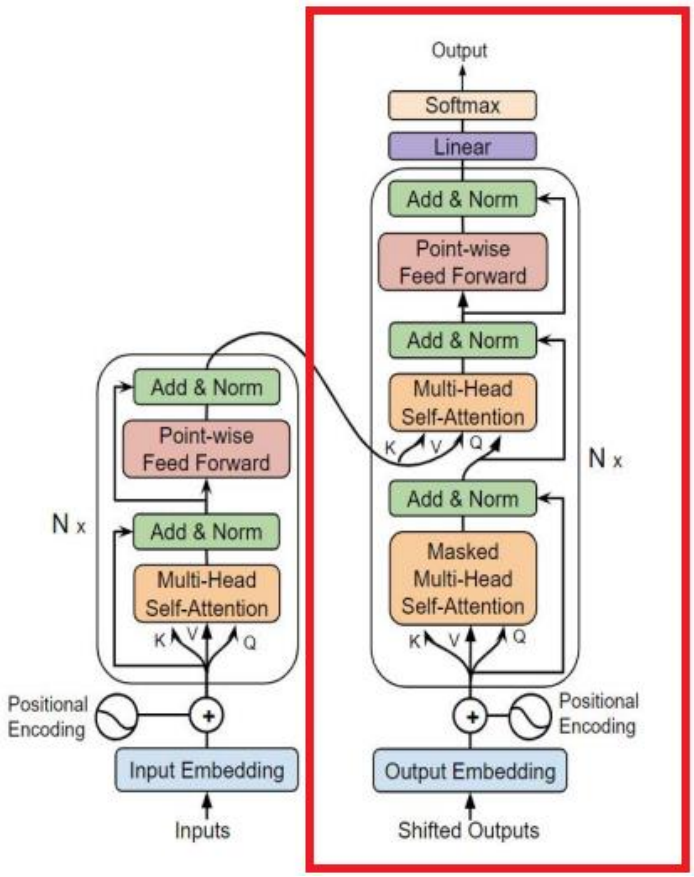
$$LayerNorm(X) = \gamma \frac{X - \mu}{\sigma} + \beta$$

Where:

- γ and β are learnable scaling and shifting parameters.
- μ is the mean of the feature vector across the mini-batch.
- σ is the standard deviation of the feature vector across the mini-batch.

5) Decoder

Decoder is also a stacked composite of 6 layers. The architecture is similar to the sub-layers in the Encoder except that one more sub-layer Masked Multi-Head Attention. As for the second Mullti-Head Attention layer, normally each Self-Attention layer that receives Q, K, V will receive K, V as the output of the last layer of the Encoder (Above I have an image. description), of course Q is still input and output.



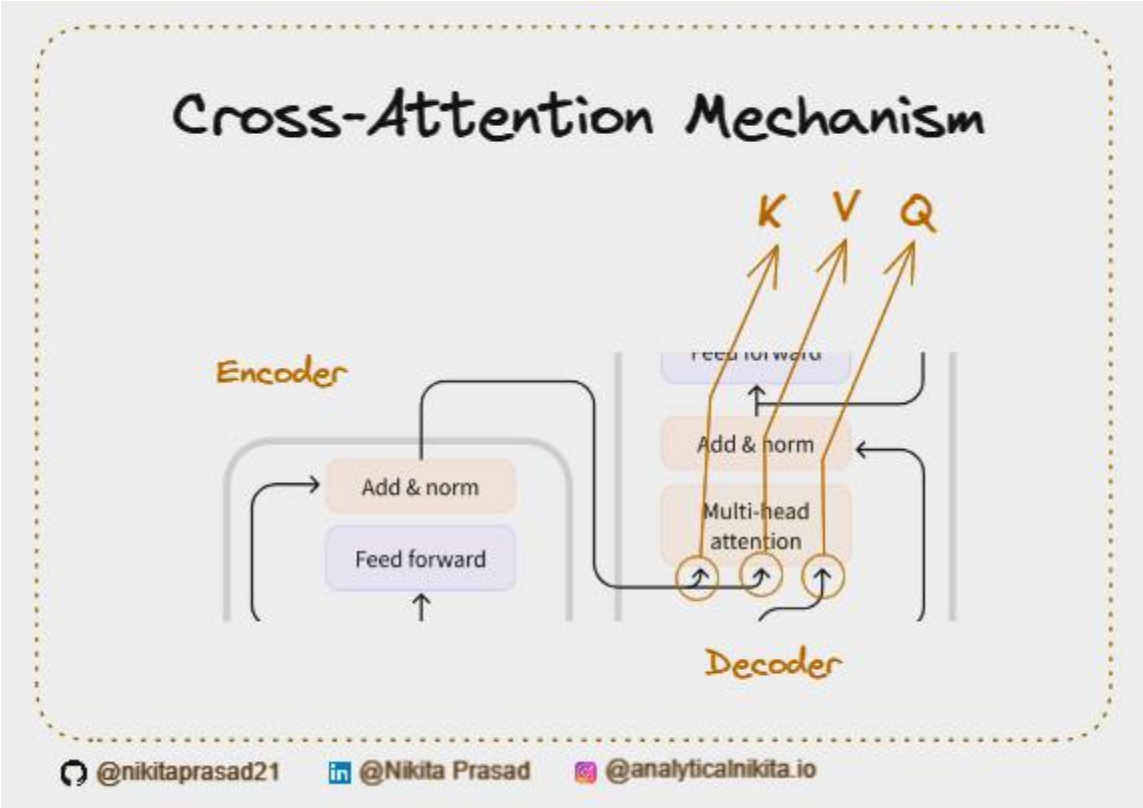
Cross-Attention: Bridging the Encoder and Decoder

Now, let's talk about cross-attention. While self-attention deals with a single input sequence, ***cross-attention involves two different input sequences***.

Simply putting, that's the only difference.

This is crucial in tasks like ***machine translation***, where you're dealing with a source language (like *English*) and a target language (like *French*).

Cross-attention acts as the bridge between the encoder and decoder in a transformer model. Here's how it works:



Multi-head Attention layer that utilizes Cross-Attention Mechanism — (Image by Author)

1. **Encoder Output:** The encoder processes the source language (English) and produces a set of encoded representations.
2. **Decoder Input:** The decoder takes the target language (French) input sequence.
3. **Cross-Attention:** The output of the encoder (encoded representations) is used as the **Key** and **Value** in the cross-attention mechanism, while the **Query** comes from the previous decoder's layer, for Multi-Head Attention.

So, why is this important?

By using cross-attention mechanism, each part of the decoder can focus on any part of the input sequence. This allows the model to learn different language conversion patterns between the two sequences, leading to more accurate and fluent translations.

a. Masked Multi_head Attention

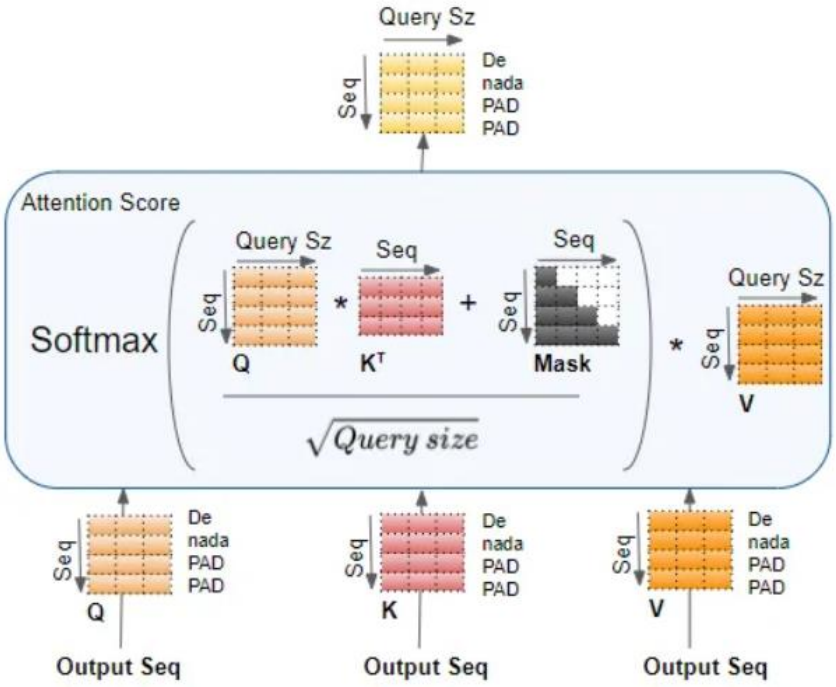
This represents the attention distribution in the first place. This layer is no different from the multi-head self-attention layer except that it is adjusted to not bring future words to attention. At the ith step of the decoder, we only know the words at the position less than i, so the attention adjustment only applies to the words smaller than the i position. Residual mechanism is also applied in the same way as in Encoder.

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

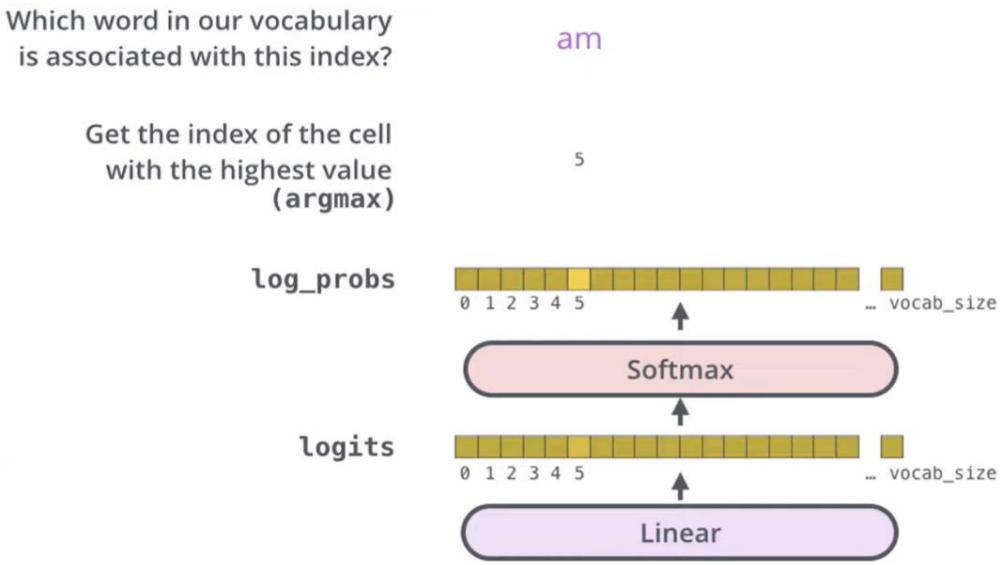
Where M is mask matrix of 0 and $-\infty$

$M = [0 \quad -\infty\ 0 \quad 0 \quad \ldots \quad -\infty \ -\infty \ -\infty \ -\infty \ \ldots \ \ldots \ 0 \quad 0 \quad \ldots \ \ldots \quad 0 \quad 0 \quad]$

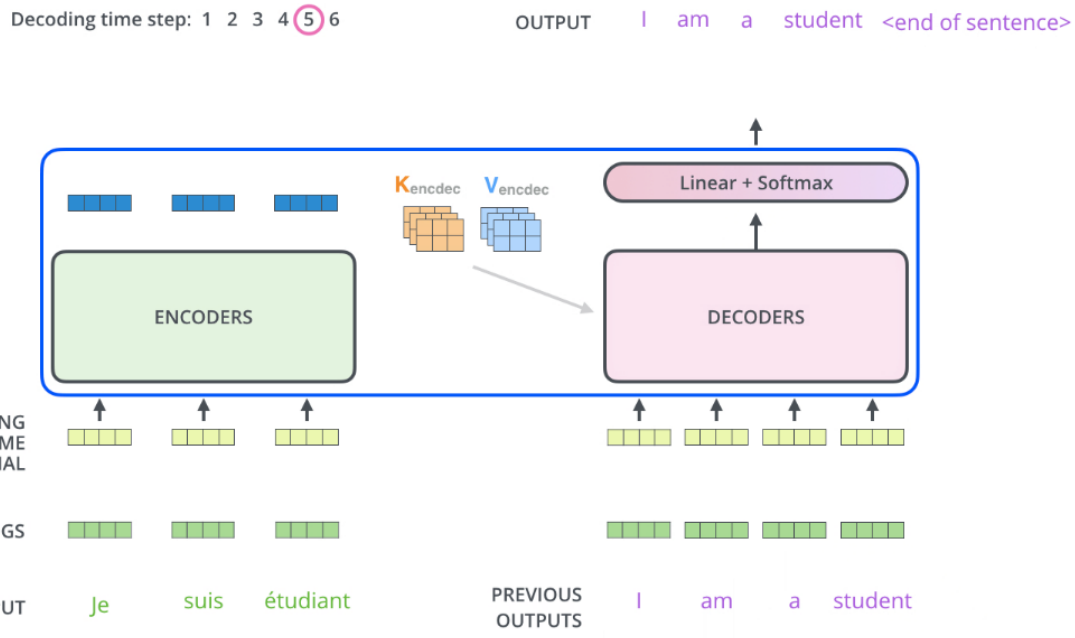
When using the softmax function, the values of $-\infty$ will be converted to the value 0.



6) Linear & softmax



The output of the Decoder will be passed through a linear layer to convert to a dimension equal to the dimension of the Decoder's embedded layer (Output Embedding) and through a Softmax layer to divide their probabilities. The process goes on like that until the end of the sentence is met (for the machine translation problem).



That's why we always have to pad each sentence with the beginning and the end of the string to be able to start and end.

7) Pros and Cons

- **Pros:**

Parallelization: Transformers process input data much faster than sequential models like RNNs due to their high parallelizability, resulting in faster training and inference.

Scalability: Transformers handle sequences of varying lengths effectively, making them suitable for diverse tasks, from short text classification to long document summarization.

State-of-the-Art Performance: Transformers have achieved state-of-the-art results in various NLP tasks, including machine translation, text classification, question answering, and language modeling. They've also excelled in computer vision tasks.

Multimodal Capabilities: Transformers can handle multiple data modalities (text, images, audio), making them versatile for applications beyond text-based tasks.

Long-term Dependencies: Transformers effectively capture long-range dependencies in sequences, making them suitable for tasks requiring extensive context.

- **Cons:**

Computational Intensity: Training large-scale Transformer models demands substantial computational resources, which may not be accessible to all researchers and organizations.

Large Memory Footprint: Deep and wide Transformer models can have a significant memory footprint, posing challenges for deployment on resource-constrained devices or real-time applications.

Limited Sequential Understanding: Transformers may struggle with tasks requiring genuine sequential understanding, such as programmatic reasoning or precise temporal modeling.

Data Hungry: Pre-training large-scale Transformers typically requires massive text data, limiting their usability for low-resource languages and domains.

Fine-tuning Challenges: Fine-tuning pre-trained Transformer models can be challenging, often necessitating careful hyperparameter tuning and substantial task-specific data.

II. Machine Translation

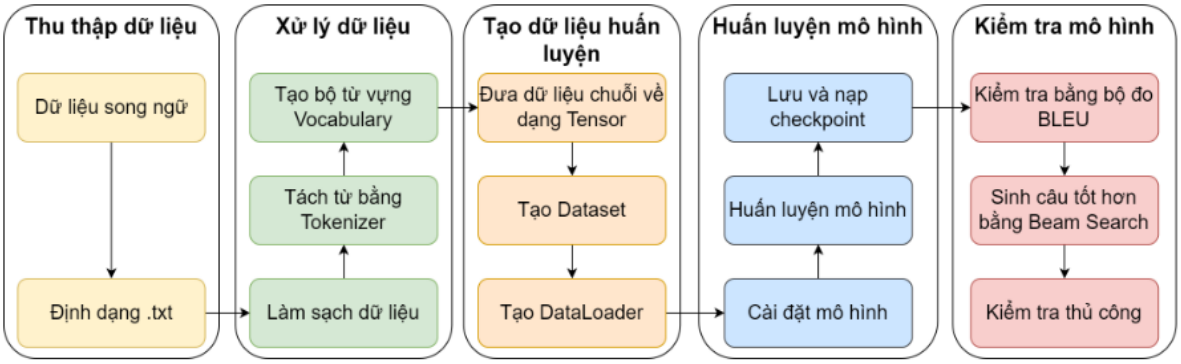
1) Introduce

Overview about Neural Machine Translation (NMT):

Neural Machine Translation (NMT) is a cutting-edge approach to automatic language translation that utilizes artificial neural networks, particularly deep learning models, to translate text from one language to another. It has gained significant popularity and surpassed traditional statistical machine translation methods in recent years due to its ability to produce more fluent and contextually accurate translations.

The main phases in the NMT system are:

Data collection > Data analysis > Create and prepare data for training > Training model > Evaluate model.



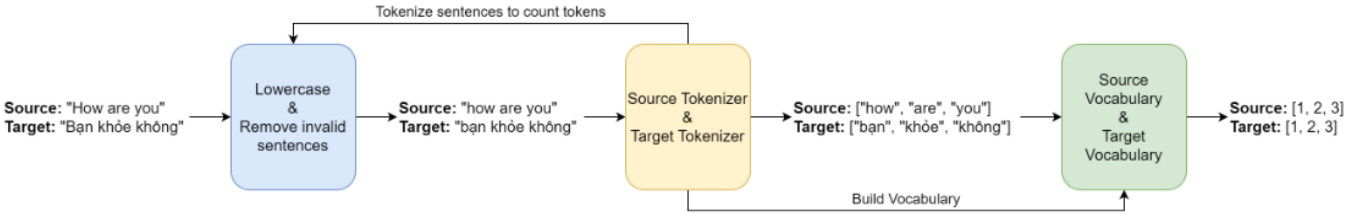
2) Data collection

Concept of project: Use a transformer model for translating sentences or words from English to VietNamese.

Data source: IWSLT'15 Anh-Viet dataset including 600k sentences collected and processed on TED. And we will use 133000 pairs of sentences for the training model.

Type of data: train.vi, train.en for training dataset and the same with validation dataset. The reason for choosing a text data set is because its size is quite light for storage.

3) Data analysis



To get good results, the first thing we need to do is to process the data.

Here are some *commonly used terms* in natural language processing (NLP):

Source Sentence (or Original Sentence): This term refers to the sentences in the source language, which are the sentences intended to be translated into the target language.

Target Sentence (or Reference Sentence): A target sentence is the sentence in the target language that serves as a reference. It is used to compare with the model's translated sentence for evaluation and accuracy assessment.

Predicted Sentence (or Model Candidate Sentence, Translated Sentence): The predicted sentence is the result of the model's translation from the original source sentence. This sentence is also in the target language.

Tokenizer (or Word Segmenter, Word Separator): A tokenizer is a tool or function used to split a sentence into smaller units, known as tokens. For example, it can break "Hello, somebody here?" into tokens like ["Hello", ",", "some", "body", "here", "?"].

Vocabulary (or Lexicon): Vocabulary is a mechanism used to convert tokens into numerical representations (usually indices) and vice versa. Tokens that have been split by the tokenizer are stored in a dictionary format (str to int) within the vocabulary. It can also be used in reverse, converting indices back to tokens using a dictionary (int to str):

- **BOS** (or SOS)**, EOS, UNK, PAD**: These are special tokens used in NLP tasks.
- **BOS** (or SOS) stands for "Begin/Start Of String" and is a token that signifies the beginning of a sentence.
- **EOS** stands for "End Of String" and is a token that marks the end of a sentence.
- **UNK** represents "Unknown" tokens, indicating words or tokens that the model doesn't recognize or understand.
- **PAD** signifies "Padding" tokens, used to fill sequences to a fixed length. These are often added to ensure uniform input sizes.

  Step by step for handle data:

Data Cleaning: For the IWSLT'15 English-Vietnamese project, this step has been completed.

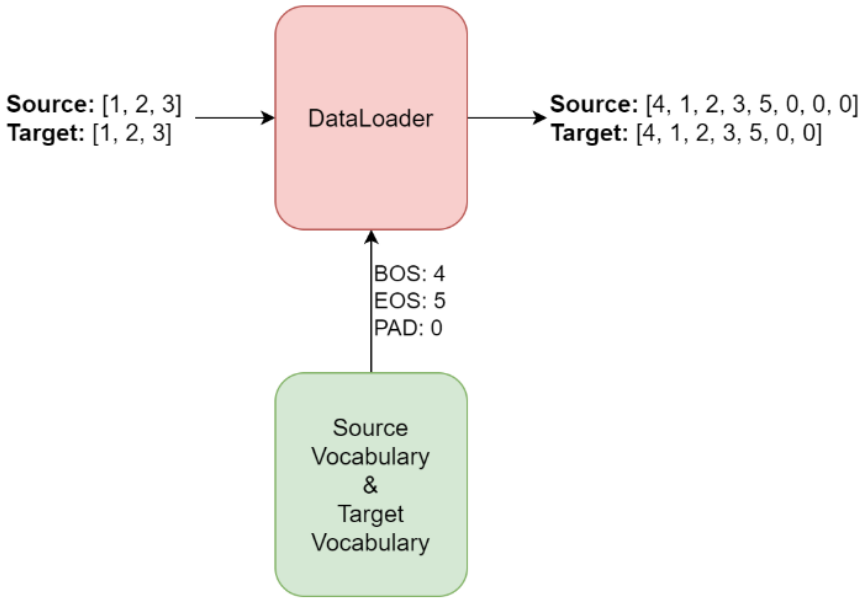
Tokenize Data: Each language requires a specific tokenizer. Vietnamese includes compound words, but we will use a word tokenizer. Word tokenization is usually more effective than multi-word tokenization for machine translation tasks.

Filtering with Tokenizer: It's necessary to remove sentences that are too long or too short. For bilingual data, source and target sentences will be removed if they are excessively long or short.

Adding Tokens to Vocabulary: After tokenization and filtering, tokens are added to the Vocabulary. Choices about the number of tokens and filtering low-frequency tokens are important.

Save Tokenizer and Vocabulary: These objects are saved to save time in the future and ensure consistent tokenization for model training.

4) Create training set



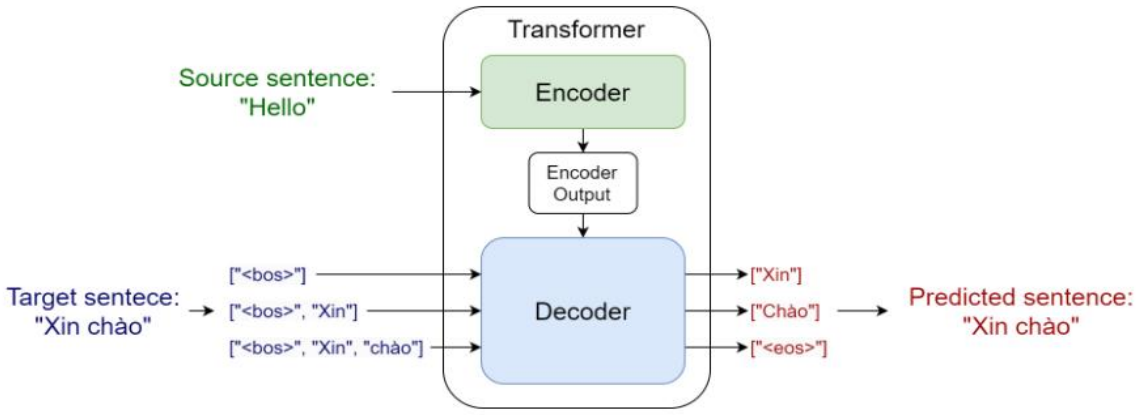
Step 1: **Convert Tokens to Numbers (Token to Index)**: In this step, we use a Vocabulary to map tokens in text to corresponding indices. Any tokens not found in the Vocabulary are mapped to the index of the UNK token (unknown).

Step 2: **Create a Dataset of Sentence Pairs**: In this step, we create a PyTorch `Dataset` by transforming the input data into a list of dictionaries, where each dictionary contains a pair of source and target sentences converted to indices.

$$dict("src": list(int), "trg": list(int))$$

Step 3: **Create a DataLoader from the Dataset**: We create a `DataLoader` from the `Dataset` to manage dividing the data into small batches. It's important to ensure that sentences in each batch have the same length by adding the index of PAD tokens to shorter sentences. This helps the model work efficiently and prevents GPU overflow. Sorting sentences by length, either ascending or descending, can also improve model performance because the number of PAD tokens is reduced.

5) Training



In the case of Neural Machine Translation (NMT), the Decoder is a type of autoregressive model that takes input as the output of the encoder and a sequence of token indices, starting with the BOS token, to predict the next token. It iteratively repeats this process until it predicts EOS.

In the initial stages, the model is often incomplete, so it takes time to wait for the prediction of EOS or a sentence longer than the predefined limit. This leads to poor decoder outputs, affecting training performance. Therefore, we use the target sentence to address this issue, adding one word at a time, stopping when all tokens from the target sentence have been added; this is known as **Teacher Forcing**.

During testing, **Teacher Forcing** should not be used. The Decoder autonomously uses its own output as input to evaluate the model's capabilities, especially when translating user sentences where a target sentence is missing.

6) Special Techniques for Transformer Training

a. Optimizer

We used the Adam optimizer. We varied the learning rate over the course of training:

$$lrate = d_{model}^{-0.5} \cdot min\left(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-0.5}\right)$$

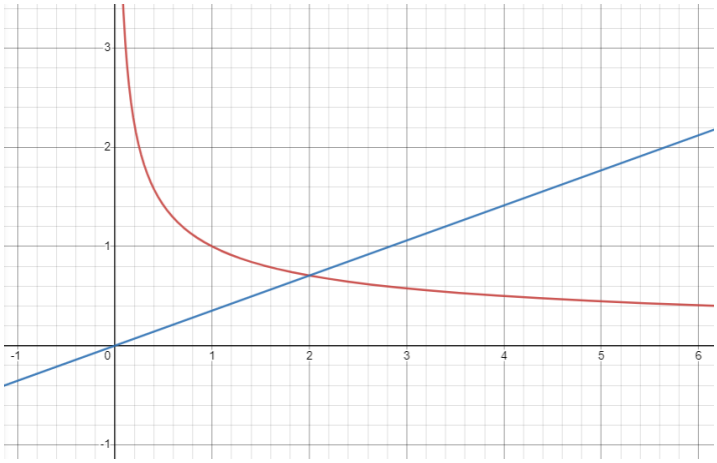
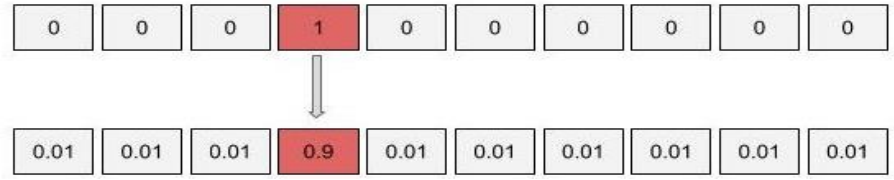


Figure 3:The red curve is the exponential function $step\_num^{-0.5}$ *, the blue line is linear function* $step\_num \cdot warmup\_steps^{-0.5}$ *(with* $warmup\_steps = 2$*)*

This corresponds to increasing the learning rate linearly for the first warmup_steps training steps (help the model to converge faster), and decreasing it thereafter proportionally to the inverse square root of the step number (to avoid the model only oscillates around the minimum value that cannot converge due to the large learning rate).

b. Label Smoothing

With a large number of parameters, the transformer is prone to overfitting. To solve this problem, instead of encoding the output labels of the training set with one-hot vectors, we distribute some probabilities to the remaining labels.



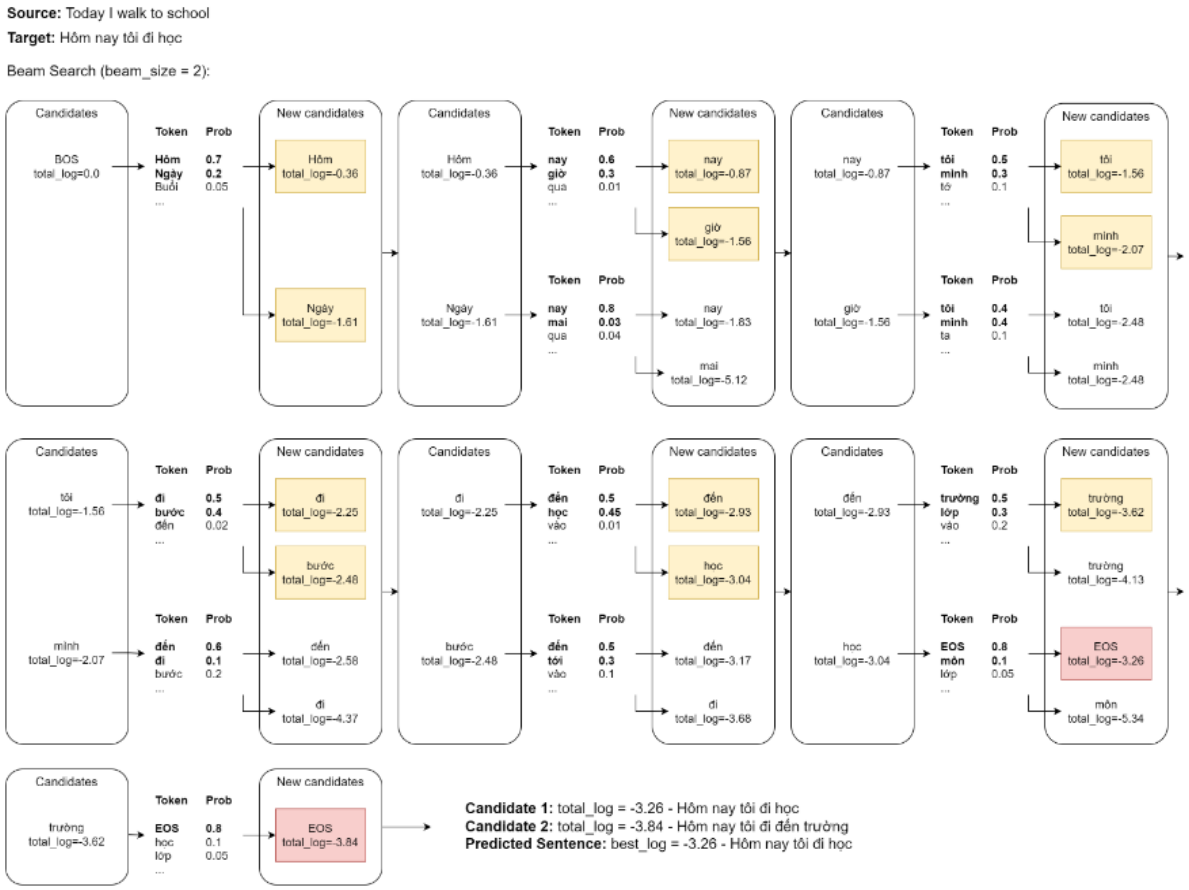
The model learns to be more unsure, but improves accuracy and BLEU score.

7) Beam search

As we said in the model training section, the decoder will receive the index string to predict the output token. This output token will by default be selected as the highest probability Token in Vocabulary. The ultimate goal is to find the sequence of tokens that produce the highest probabilities.

Beam search is a way to strike a balance between Greedy Search and Exhaustive Search. The idea is that the model will keep the best n translations at each step. With n (beam size) is a hyper-parameter chosen by me (in project, n = 5).

e.g.,



8) Evaluation (use BLEU score)

BLEU (Bilingual Evaluation Understudy) is a metric used to evaluate the quality of machine - generated text, typically in the context of machine translation tasks.

The value range of The BLEU is [0;1], where a higher score indicates better similarity between the machine-generated output and the human references.

$$Bleu(N) = Brevity\ Penalty \cdot Geometric\ Average\ Precision(N)$$

$$Geometric\ Average\ Precision\ (N) = exp\ exp\left(\sum_{n=1}^{N} w_n\ log\ log\ p_n\right) = \prod_{n=1}^{N} p_n^{w_n}$$

$$= (p_1)^{\frac{1}{4}} \cdot (p_2)^{\frac{1}{4}} \cdot (p_3)^{\frac{1}{4}} \cdot (p_4)^{\frac{1}{4}}$$

p_n: is the precision n-gram calculated by $\frac{Number\ of\ correct\ predicted\ n-grams}{Number\ of\ total\ predicted\ n-grams}, n = 1,2,3,4.$

It works by comparing n-grams (contiguous sequences of n words) in the machine-generated text with those in the reference translations.

e.g., calculate P2

Target: The guard arrived late because it was raining.

Predicted: The guard arrived late because of the rain.

Precision 2-gram $(p_2) = \frac{4}{7}$

$$Brevity\ Penalty = \{1\ e^{1-\frac{r}{c}} \quad if\ c > r\ if\ c \leq r$$

Where:

- c là predicted length
- r là target length

Brevity Penalty is less than or equal to 1. And the fewer words in the prediction sentence than in the target sentence, the smaller this value and BLEU score will be.

References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin, 2023, Attention Is All You Need

Quoc Pham, 2020, Tìm hiểu mô hình Transformer

Pham Dinh Khanh, 2019, Attention is all you need

Minh Bùi Blog, 2020, Giải Thích Mô Hình Transformer

Lê Minh Tú, 2023, Cùng tìm hiểu hệ thống dịch máy mạng nơ ron từ đầu. Từ BLEU score đến Beam Search Decoding

https://ai.plainenglish.io/how-large-language-models-work-ae40b277ff5c

Fareed Khan, Solving Transformer by Hand: A Step-by-Step Math Example