

# DHAKA UNIVERSITY OF ENGINEERING & TECHNOLOGY, GAZIPUR



## Department of Computer Science and Engineering

**Course No:** CSE-4620

**Course Title:** Machine Learning Sessional

**Date of Submission:** 04-10-2023

### **Presented To :**

**Dr. Fazlul Hasan Siddiqui**

Professor, Department of CSE, DUET.

**Dr. Amran Hossain**

Associate Professor, Department of CSE, DUET.

### **Presented By:**

Taposh Kumar Biswas

Student ID: 194024

Israt zahan

Student ID:194049

Prantha Debonath

Student ID:194056

## Abstract

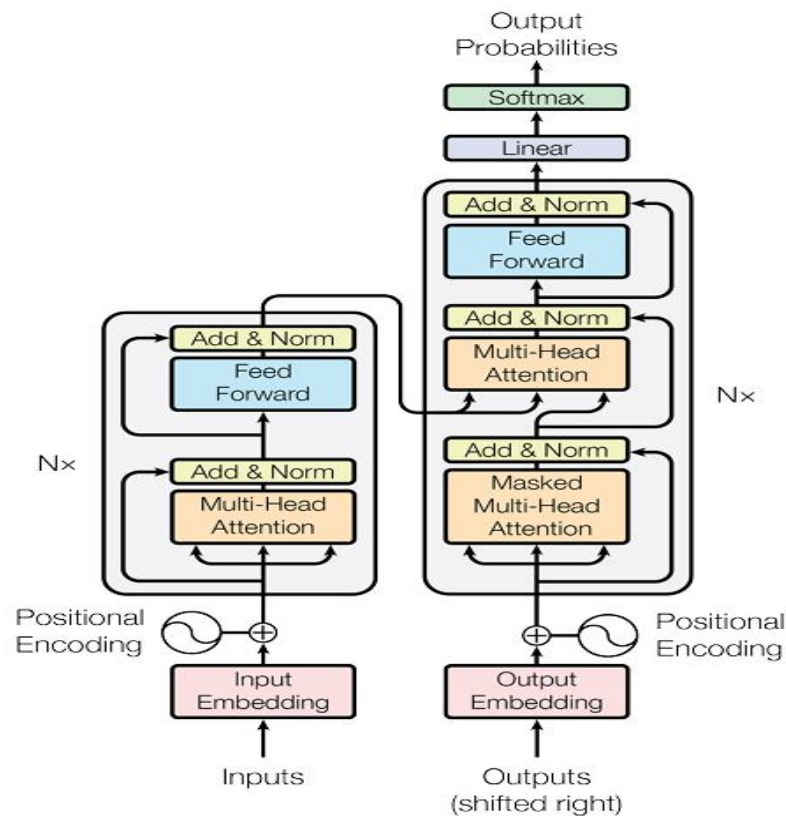
The transformer model is one of the most recently developed models for translating texts into another language. The model uses the principle of attention mechanism, surpassing previous models, such as sequence-to-sequence, in terms of performance. It can convert a sequence of text from one language to another. It refers to translating texts or documents from the source language into the target language without human intervention. This study presents a deep learning-based MT system concerning both-way translation for the Bangla-English language. The attention-based multi-headed transformer model has been considered in this study due to its significant features of parallelism in input processing. A transformer model consisting of encoders and decoders is adapted by tuning different parameters (especially, number of heads) to identify the best performing model for Bangla to English.

Keywords: Deep Learning, Machine Translation, Neural Machine Translation, Transformer Model

## Introduction

A transformer network is a machine learning technology introduced in 2017 that revolutionized tasks like language translation and natural language understanding. It excels at handling sequential data by using a mechanism called "attention" to capture relationships between input elements. Transformers have become fundamental in various applications, making them a cornerstone of modern AI.

**Encoder:** The encoder is composed of a stack of  $N = 6$  identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [10] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is  $\text{LayerNorm}(x + \text{Sublayer}(x))$ , where  $\text{Sublayer}(x)$  is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension  $d_{\text{model}} = 512$ .



**Decoder:** The decoder is also composed of a stack of  $N = 6$  identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position  $i$  can depend only on the known outputs at positions less than  $i$ .

**Attention:** An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

## Related Work

Transformer networks have been the subject of extensive research since their introduction in the paper "Attention Is All You Need" by Vaswani et al. in 2017. Since my knowledge cutoff date is in September 2021, I can provide you with an overview of some key areas of related work and advancements in transformer network research up to that point:

### Architectural Variants:

**BERT** (Bidirectional Encoder Representations from Transformers): Introduced by Google AI in 2018, BERT pre-trained models revolutionized natural language understanding tasks.

**GPT** (Generative Pre-trained Transformer): Developed by OpenAI, GPT models like GPT-2 and GPT-3 achieved state-of-the-art results in a wide range of NLP tasks.

**XLNet**: Introduced a permutation-based training approach to pre-training transformers, addressing limitations of BERT's bidirectional context.

### Efficiency and Scaling:

**Sparse Transformers**: Research into making transformers more computationally efficient by using sparse attention mechanisms.

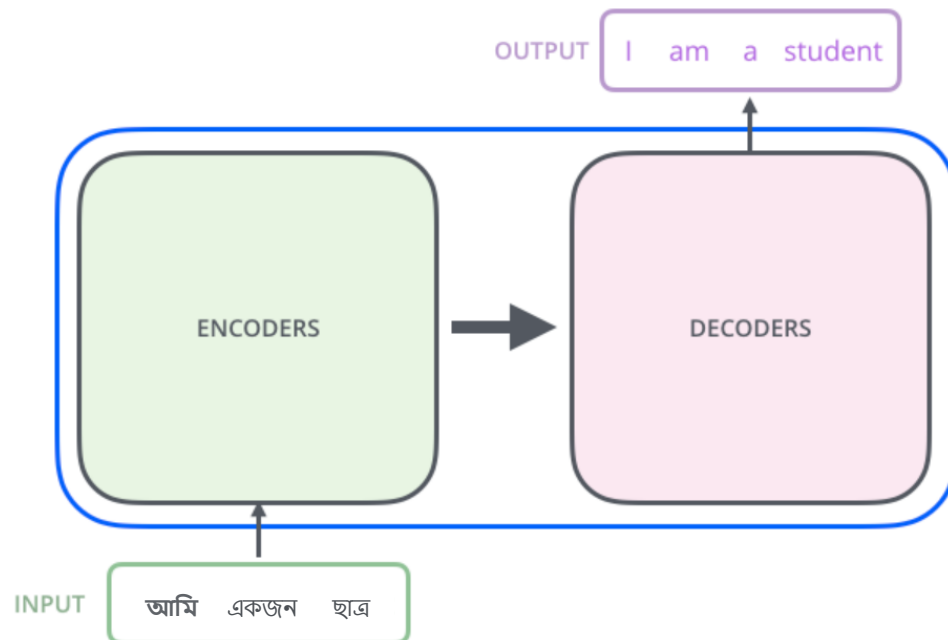
**Evolving Model Sizes**: Researchers have continually increased the size of transformer models, pushing the limits of what's possible in terms of model size and performance.

## System details

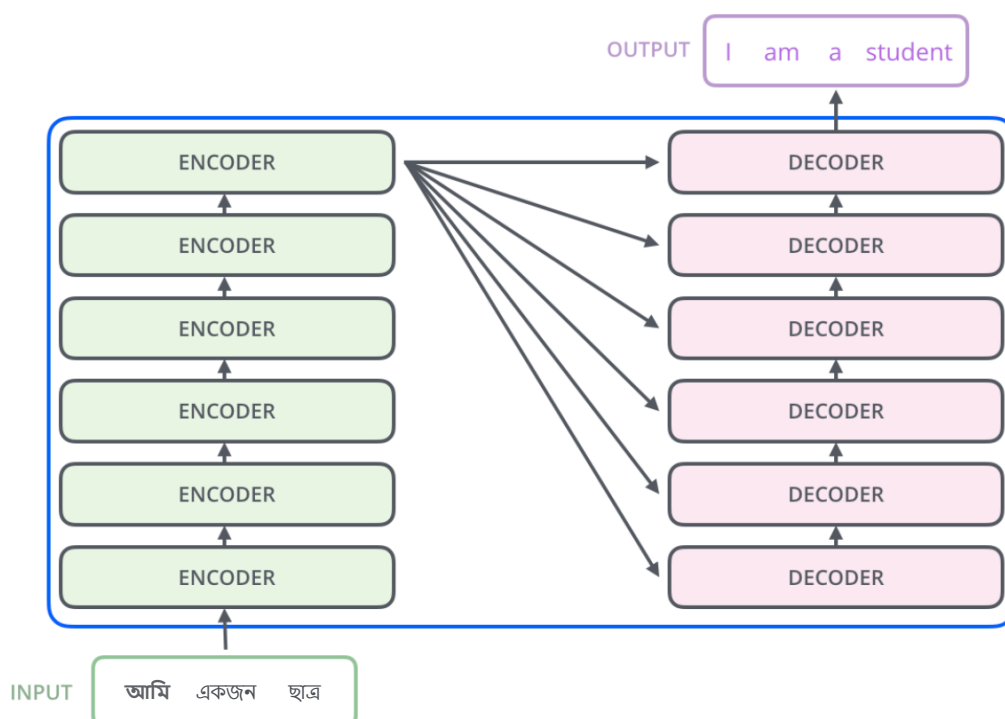
A deep learning-based transformer model is investigated in this study to develop an MT, taking advantage of the transformer's parallelism features in the input data processing. A transformer model consists of encoders and decoders, where learnable parameters are tuned to identify the best performing MT model for Bangla to English.



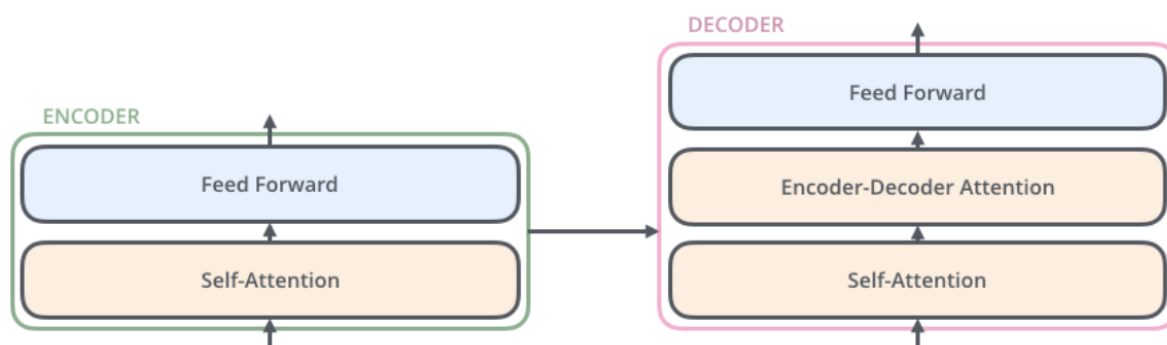
If we see this in more details then there have an encoder and decoder.



The encoding component is a stack of encoders (the paper stacks six of them on top of each other – there's nothing magical about the number six, one can definitely experiment with other arrangements). The decoding component is a stack of decoders of the same number.



The encoders are all identical in structure. Each one is broken down into two sub-layers:



The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word. We'll look closer at self-attention later in the post. The outputs of the self-attention layer are fed to a feed-forward neural network. The exact same feed-forward network is independently applied to each position. The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence.

## Literature Review

Now that we've seen the major components of the model, let's start to look at the various vectors/tensors and how they flow between these components to turn the input of a trained model into an output.



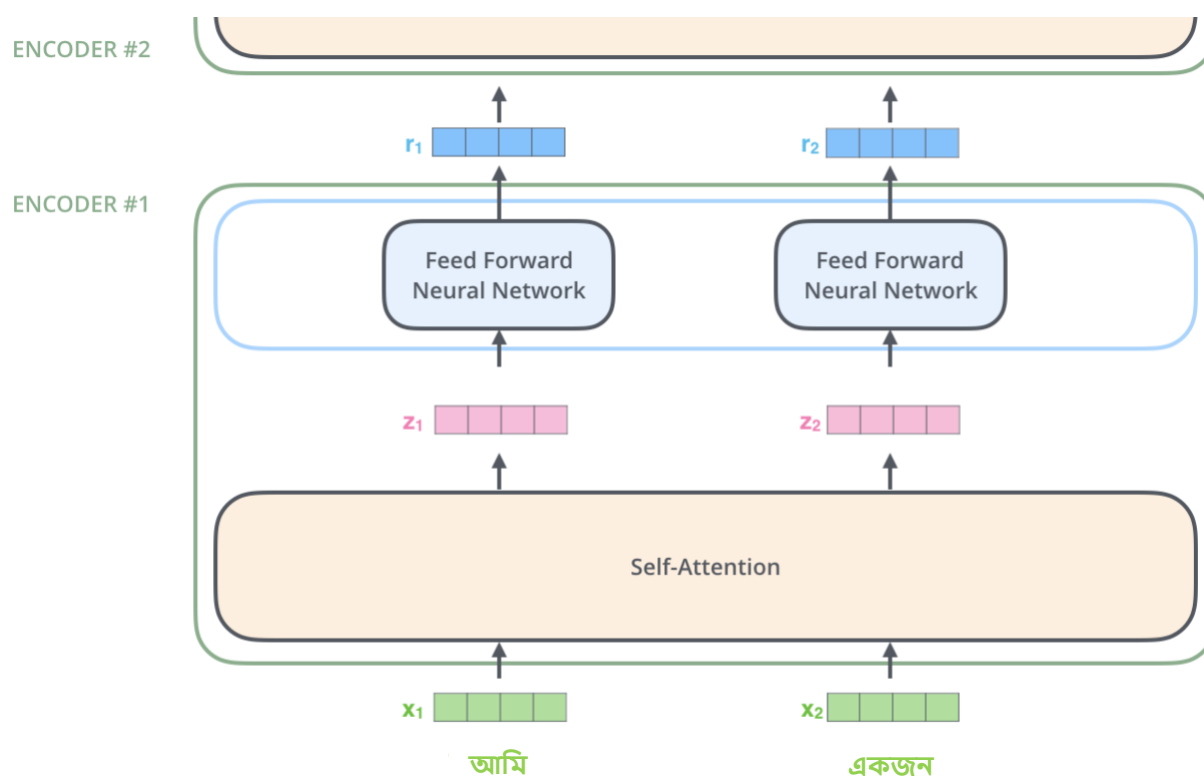
Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

The embedding only happens in the bottom-most encoder. The abstraction that is common to all the encoders is that they receive a list of vectors each of the size 512 – In the bottom encoder that would be the word embeddings, but in other encoders, it would be the output of the encoder that's directly below. The size of this list is hyperparameter

we can set – basically it would be the length of the longest sentence in our training dataset. After embedding the words in our input sequence, each of them flows through each of the two layers of the encoder.

### Encoder Works:

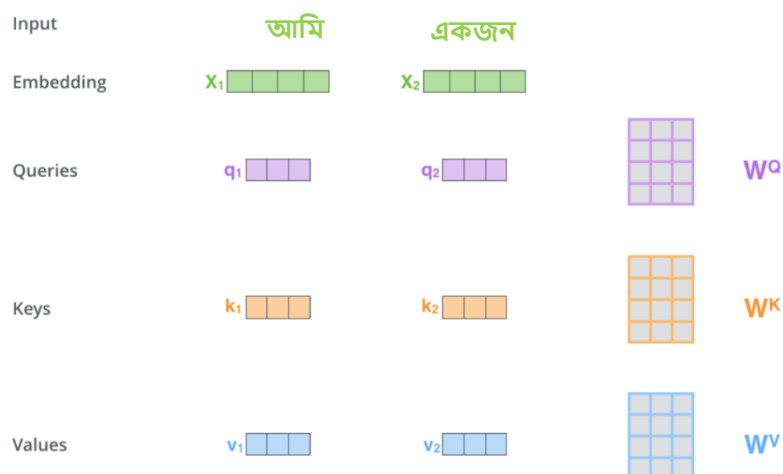
As we've mentioned already, an encoder receives a list of vectors as input. It processes this list by passing these vectors into a 'self-attention' layer, then into a feed-forward neural network, then sends out the output upwards to the next encoder. Now we just work with two vector from embedded the sentence.



### Self-Attention

Don't be fooled by me throwing around the word "self-attention" like it's a concept everyone should be familiar with. I had personally never come across the concept until reading the Attention is All You Need paper. Let us distill how it works. Let's first look at how to calculate self-attention using vectors, then proceed to look at how it's actually implemented – using matrices.

The **first step** in calculating self-attention is to create three vectors from each of the encoder's input vectors (in this case, the embedding of each word). So for each word, we create a Query vector, a Key vector, and a Value vector. These vectors are created by multiplying the embedding by three matrices that we trained during the training process.



In The **Second steps** we have calculate a score. The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring. So if we're processing the self-attention for the word in position #1, the first score would be the dot product of  $q_1$  and  $k_1$ . The second score would be the dot product of  $q_1$  and  $k_2$ .

The **third and fourth steps** are to divide the scores by 8 (the square root of the dimension of the key vectors used in the paper – 64. This leads to having more stable gradients. There could be other possible values here, but this is the default), then pass the result through a softmax operation. Softmax normalizes the scores so they're all positive and add up to 1.

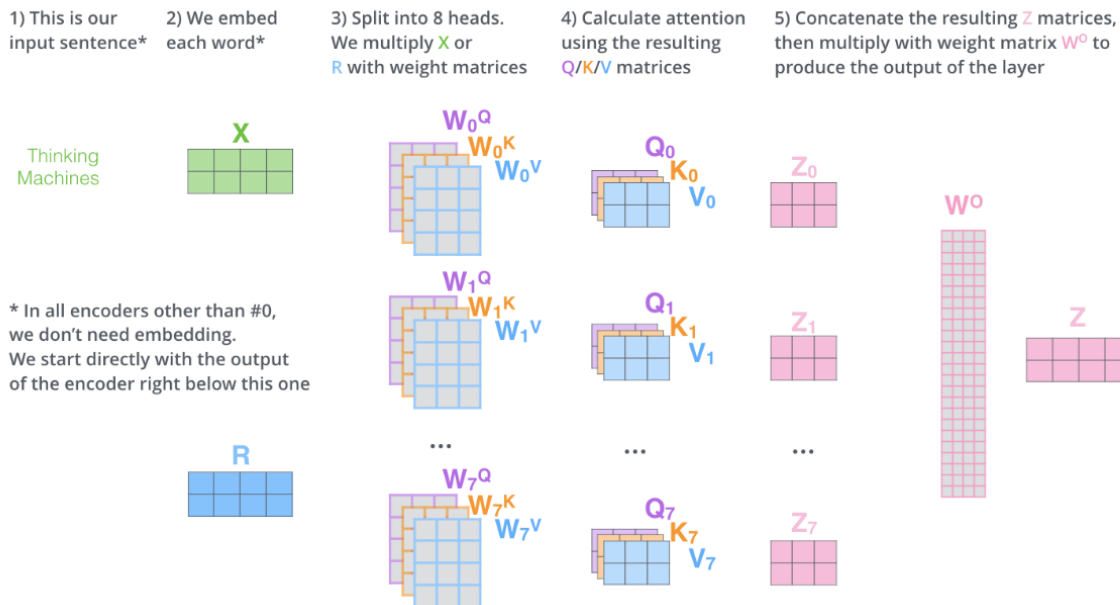
This softmax score determines how much each word will be expressed at this position. Clearly the word at this position will have the highest softmax score, but sometimes it's useful to attend to another word that is relevant to the current word.

The **fifth step** is to multiply each value vector by the softmax score (in preparation to sum them up). The intuition here is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words (by multiplying them by tiny numbers like 0.001, for example).

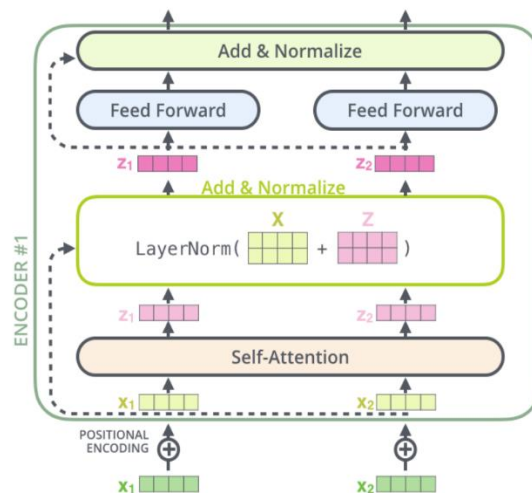




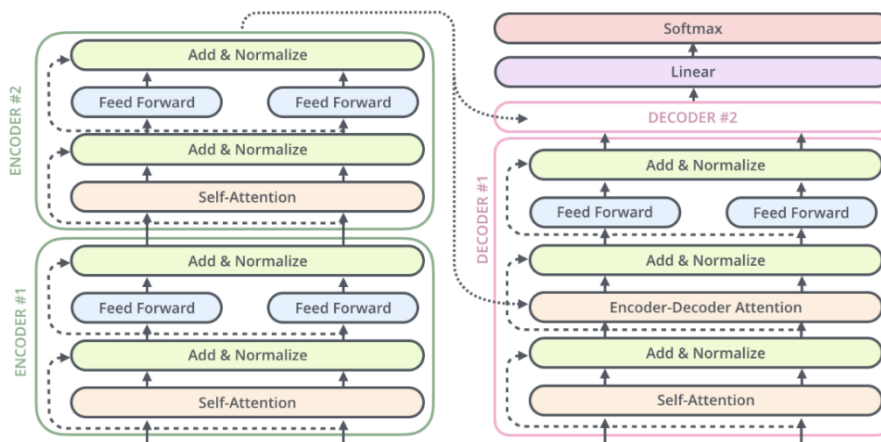
The looks when multiple head of word vector pass self-attention.



If we're to visualize the vectors and the layer-norm operation associated with self attention, it would look like this:

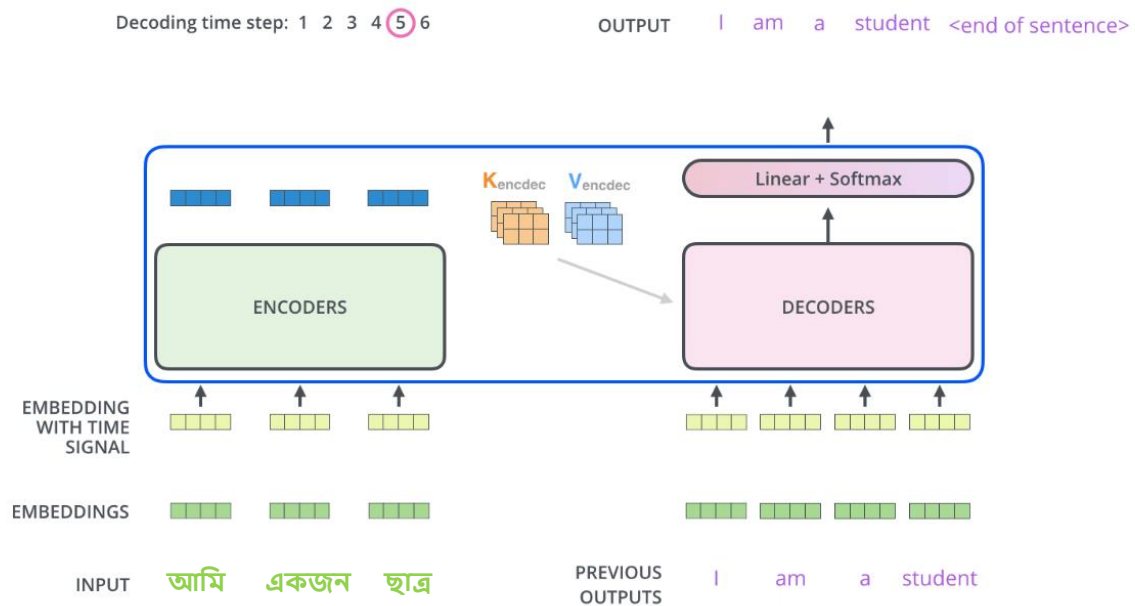


This goes for the sub-layers of the decoder as well. If we're to think of a Transformer of 2 stacked encoders and decoders, it would look something like this:



## The Decoder Side

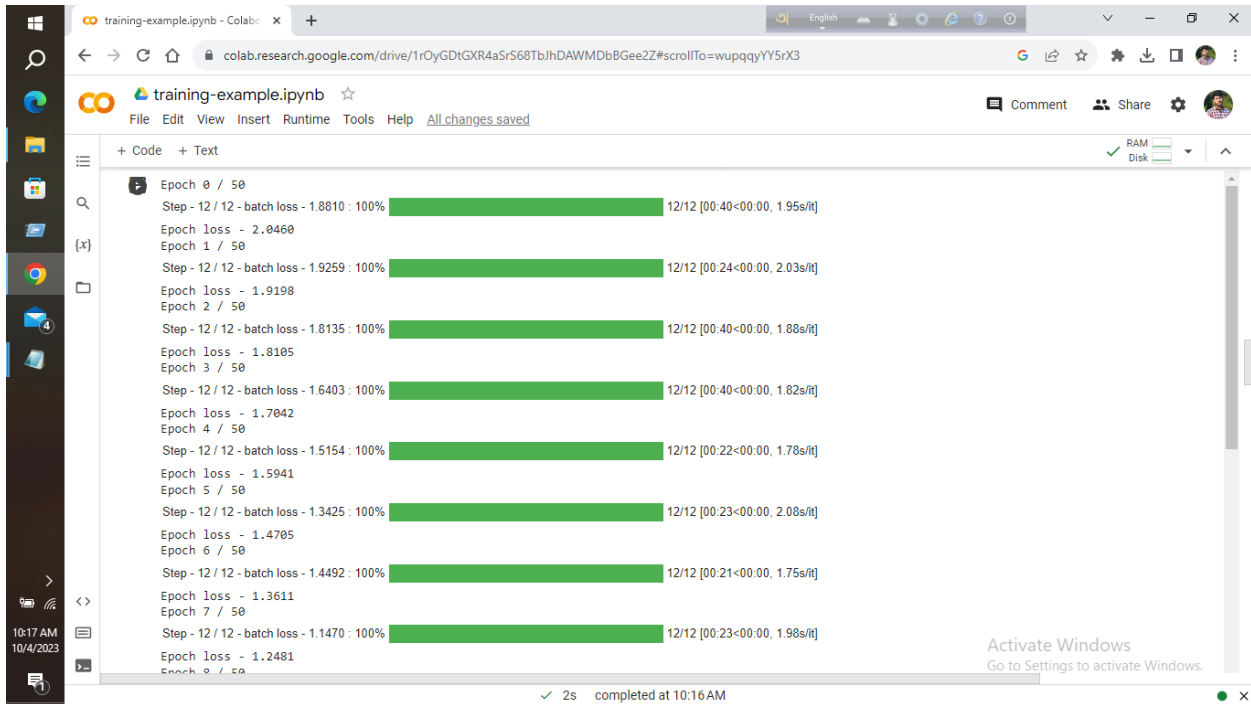
Now that we've covered most of the concepts on the encoder side, we basically know how the components of decoders work as well. But let's take a look at how they work together. The encoder starts by processing the input sequence. The output of the top encoder is then transformed into a set of attention vectors  $K$  and  $V$ . These are to be used by each decoder in its "encoder-decoder attention" layer which helps the decoder focus on appropriate places in the input sequence:



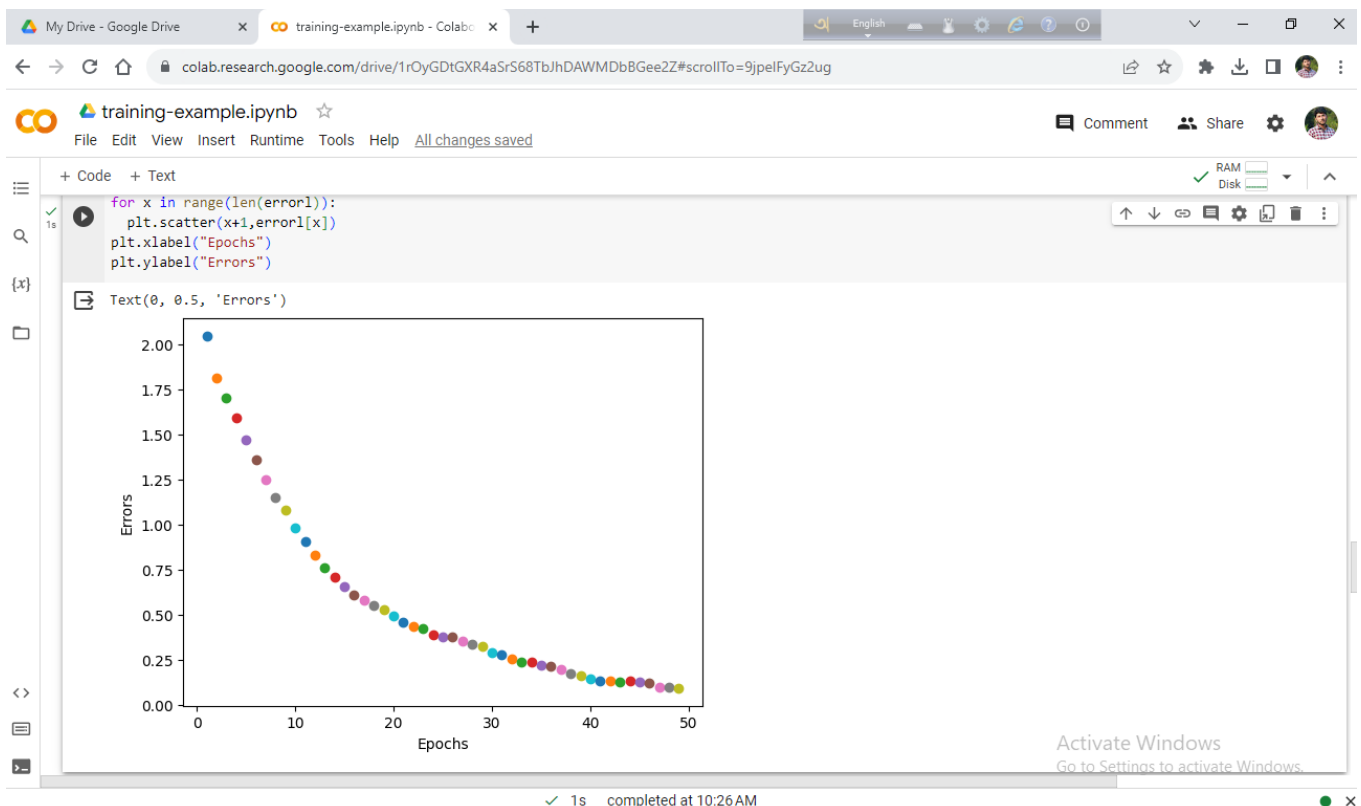
## Experiments

Here we have large number of dataset to train the model, but for the time and machine complexity we just train the model with 2000 data and 50 epochs with batch size 64.

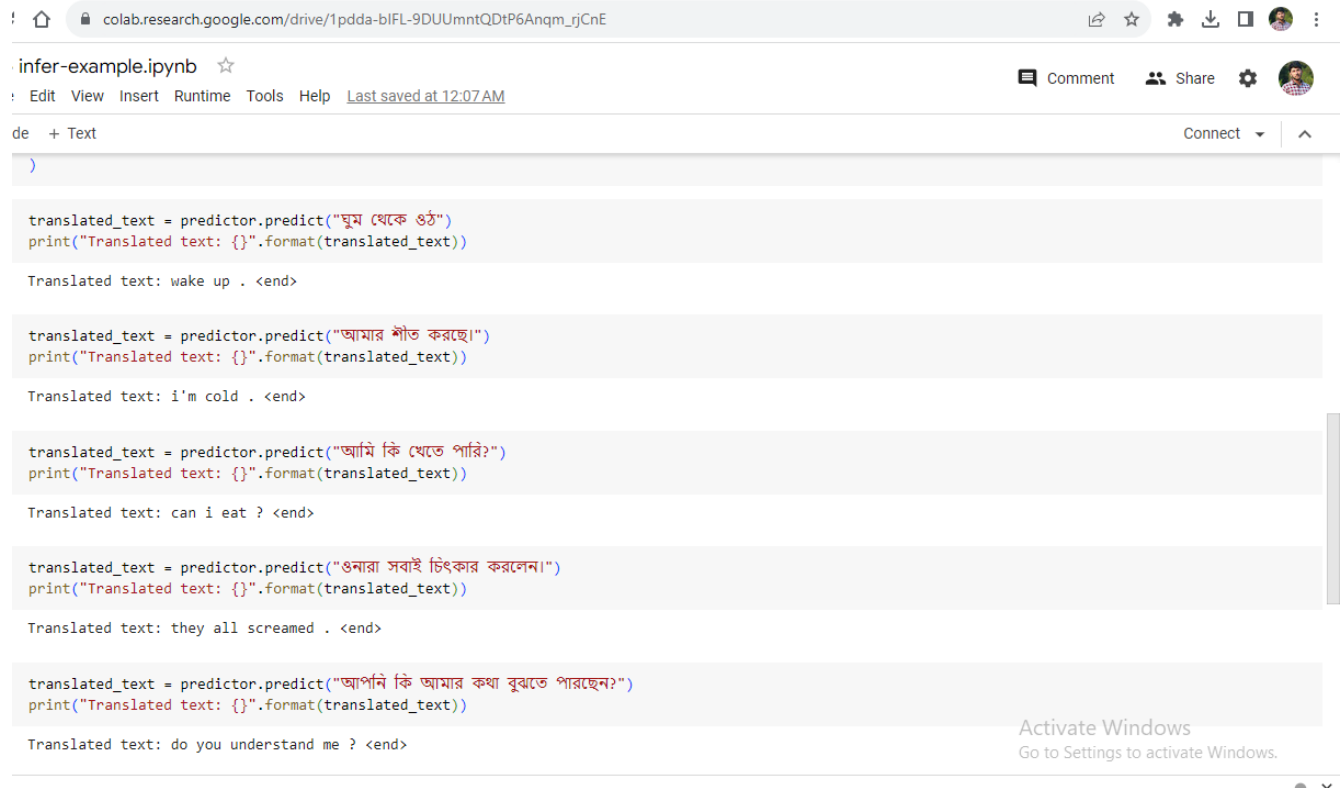
```
ben - Notepad
File Edit Format View Help
Go. যাও। CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #5545004 (tanay)
Go. যা। CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #5545005 (tanay)
Go. যা। CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #5545006 (tanay)
Run! পালস। CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #5548781 (tanay)
Run! পালস। CC-BY 2.0 (France) Attribution: tatoeba.org #906328 (papabear) & #5548783 (tanay)
Who? কে? CC-BY 2.0 (France) Attribution: tatoeba.org #2083030 (CK) & #5548787 (tanay)
Fire! আগুন। CC-BY 2.0 (France) Attribution: tatoeba.org #1829639 (Spamster) & #3232240 (tanay)
Help! বর্জিত। CC-BY 2.0 (France) Attribution: tatoeba.org #435084 (lukaszpp) & #5548780 (tanay)
Help! বাঁসা। CC-BY 2.0 (France) Attribution: tatoeba.org #435084 (lukaszpp) & #5548782 (tanay)
Stop! থামো। CC-BY 2.0 (France) Attribution: tatoeba.org #448320 (FeuDReNaIs) & #5545000 (tanay)
Stop! থামো। CC-BY 2.0 (France) Attribution: tatoeba.org #448320 (FeuDReNaIs) & #5545001 (tanay)
Stop! থাম। CC-BY 2.0 (France) Attribution: tatoeba.org #448320 (FeuDReNaIs) & #5545002 (tanay)
Hello! সমস্কার। CC-BY 2.0 (France) Attribution: tatoeba.org #373330 (CK) & #3042686 (tanay)
I see. বুঝলাম। CC-BY 2.0 (France) Attribution: tatoeba.org #2111796 (CK) & #3220326 (tanay)
I try. আমি চেষ্টা করছি। CC-BY 2.0 (France) Attribution: tatoeba.org #20776 (CK) & #3219704 (tanay)
Smile. একটু হাসো। CC-BY 2.0 (France) Attribution: tatoeba.org #2764108 (CK) & #3232180 (tanay)
Smile. একটু হাসো। CC-BY 2.0 (France) Attribution: tatoeba.org #2764108 (CK) & #3232182 (tanay)
Attack! আক্রমণ। CC-BY 2.0 (France) Attribution: tatoeba.org #1972610 (CK) & #5489253 (tanay)
Get up. ওঠো। CC-BY 2.0 (France) Attribution: tatoeba.org #896158 (pauldhunt) & #3235044 (tanay)
Get up. উঠো। CC-BY 2.0 (France) Attribution: tatoeba.org #896158 (pauldhunt) & #3235045 (tanay)
Got it! বুঝে গেছি। CC-BY 2.0 (France) Attribution: tatoeba.org #320484 (CM) & #5530042 (tanay)
Got it! বুঝেছি। CC-BY 2.0 (France) Attribution: tatoeba.org #320484 (CM) & #5530043 (tanay)
Got it? বুঝেছেন? CC-BY 2.0 (France) Attribution: tatoeba.org #455353 (FeuDReNaIs) & #5545028 (tanay)
Got it? বুঝেছেন? CC-BY 2.0 (France) Attribution: tatoeba.org #455353 (FeuDReNaIs) & #5545029 (tanay)
Got it? বুঝেছি। CC-BY 2.0 (France) Attribution: tatoeba.org #455353 (FeuDReNaIs) & #5545030 (tanay)
I know. আমি জানি। CC-BY 2.0 (France) Attribution: tatoeba.org #319990 (CK) & #3220322 (tanay)
I know. আমার জানা আছে। CC-BY 2.0 (France) Attribution: tatoeba.org #319990 (CK) & #3220324 (tanay)
I lost. আমি হেঁচকে গেছি। CC-BY 2.0 (France) Attribution: tatoeba.org #1755073 (Eldad) & #3220391 (tanay)
I'm 19. আমার ১৯ বছর বয়স। CC-BY 2.0 (France) Attribution: tatoeba.org #442400 (sacredceltic) & #3220383 (tanay)
I'm OK. আমি ঠিক আছি। CC-BY 2.0 (France) Attribution: tatoeba.org #433763 (CK) & #3220355 (tanay)
Listen. শো। CC-BY 2.0 (France) Attribution: tatoeba.org #1913088 (CK) & #5548784 (tanay)
Listen. শুন। CC-BY 2.0 (France) Attribution: tatoeba.org #1913088 (CK) & #5548785 (tanay)
No way! কোন মতেই না। CC-BY 2.0 (France) Attribution: tatoeba.org #2175 (CS) & #5548797 (tanay)
Really? সত্যি? CC-BY 2.0 (France) Attribution: tatoeba.org #373216 (kotobaboke) & #3219710 (tanay)
Really? তাই নাকি? CC-BY 2.0 (France) Attribution: tatoeba.org #373216 (kotobaboke) & #3219713 (tanay)
Thanks. ধন্যবাদ। CC-BY 2.0 (France) Attribution: tatoeba.org #2057650 (nava) & #3220431 (tanay)
Try it. চেষ্টা দেখুন। CC-BY 2.0 (France) Attribution: tatoeba.org #4756252 (cairnhead) & #5545554 (tanay)
Try it. চেষ্টা দেখো। CC-BY 2.0 (France) Attribution: tatoeba.org #4756252 (cairnhead) & #5545555 (tanay)
Ln 1, Col 1    100%    Unix (LF)    UTF-8
```



After The training we see how the error of prediction being less through a graph.And Finally the model can predict nearly something.



Here is some Output of the model prediction result:



The screenshot shows a Google Colab notebook titled 'infer-example.ipynb'. The notebook contains five code cells, each demonstrating a prediction from a model. The interface includes a top bar with the URL 'colab.research.google.com/drive/1pdda-bIFL-9DUUmntQDtP6Anqm\_rjCnE', a menu bar with 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help', and a toolbar with 'Comment', 'Share', and 'Connect' options. The code cells show the following:

```
translated_text = predictor.predict("ঘুম থেকে ওঠ")
print("Translated text: {}".format(translated_text))

Translated text: wake up . <end>

translated_text = predictor.predict("আমার শীত করছে।")
print("Translated text: {}".format(translated_text))

Translated text: i'm cold . <end>

translated_text = predictor.predict("আমি কি খেতে পারি?")
print("Translated text: {}".format(translated_text))

Translated text: can i eat ? <end>

translated_text = predictor.predict("ওনারা সবাই চিৎকার করলেন।")
print("Translated text: {}".format(translated_text))

Translated text: they all screamed . <end>

translated_text = predictor.predict("আপনি কি আমার কথা বুঝতে পারছেন?")
print("Translated text: {}".format(translated_text))

Translated text: do you understand me ? <end>
```

An 'Activate Windows' watermark is visible in the bottom right corner of the notebook interface.

## Future Work

Transforming text from one language to another, such as from Bangla to English, using Transformer-based neural networks is a complex and ongoing area of research in Natural Language Processing (NLP). While many advancements have been made up to my last knowledge update in September 2021, there are still several potential future directions. we are planning to work on a project involving Transformer-based network for Bangla to English translation, here some steps and ideas are giving below:

### **1. Data collection and Preprocessing:**

-Gather a sizable and diverse Bangla-English parallel dataset for training our model. This dataset should include text in both languages.

-Preprocess the data, which may involve tokenization, lowercasing, and removing special characters or noise from the text.

### **2. Model Selection:**

Choose a Transformer-based architecture as our base model. Common choices include BERT, GPT, or T5. You can also use pre-trained models if they are available for Bangla or English.

### **3. Efficiency and Scalability:**

Optimize your model for efficiency, especially if you plan to deploy it in resource-constrained environments. This may involve model compression or quantization.

## **Conclusion**

We have explored transformer models for a variety of Bangla text classification tasks.. We obtained a state of the art result on datasets from different domains. This project represents a significant contribution to the field of machine translation, particularly for Bangla to English translation. Our Transformer-based model, with its focus on quality, fairness, and practicality, has the potential to serve as a valuable tool for bridging language barriers and facilitating communication across cultures. Future work may include further improvements and scalability enhancements to meet the evolving needs of users and application .The Transformer network amazing project, we hardly try our best to make a good model.

## References

- [1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need" , 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
- [2] Tai, Kai Sheng, Peter Bailis, and Gregory Valiant. "Equivariant transformer networks." In International Conference on Machine Learning, pp. 6086-6095. PMLR, 2019.
- [3] Yun, Seongjun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J. Kim. "Graph transformer networks." Advances in neural information processing systems 32 (2019).
- [4] Lohit, Suhas, Qiao Wang, and Pavan Turaga. "Temporal transformer networks: Joint learning of invariant and discriminative time warping." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12426-12435. 2019.