

Transformer, BERT, cuBERT, codeBERT

By: Suraj Kumar

Sequence to Sequence Models

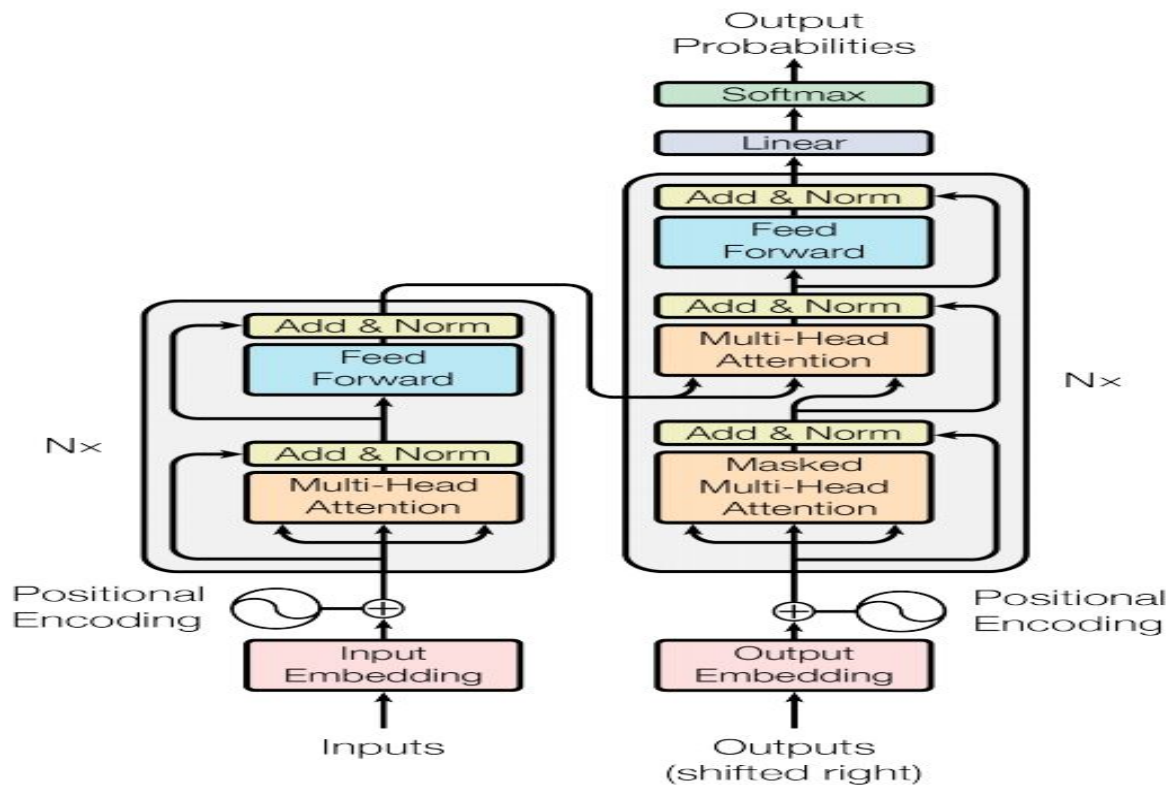
To convert sequence of one type to sequence of other type. E.g. Machine Translation, Question-Answering, POS tagging, Text summarization etc.

Models to handle seq2seq:

1. RNN (LSTM, GRU)
2. RNN with Attention (in 2015)
 - a. Attention means what to focus on.
 - b. Problems: not suitable for long-range dependencies, serial model architecture
3. Transformer (solves both problems)

Transformer

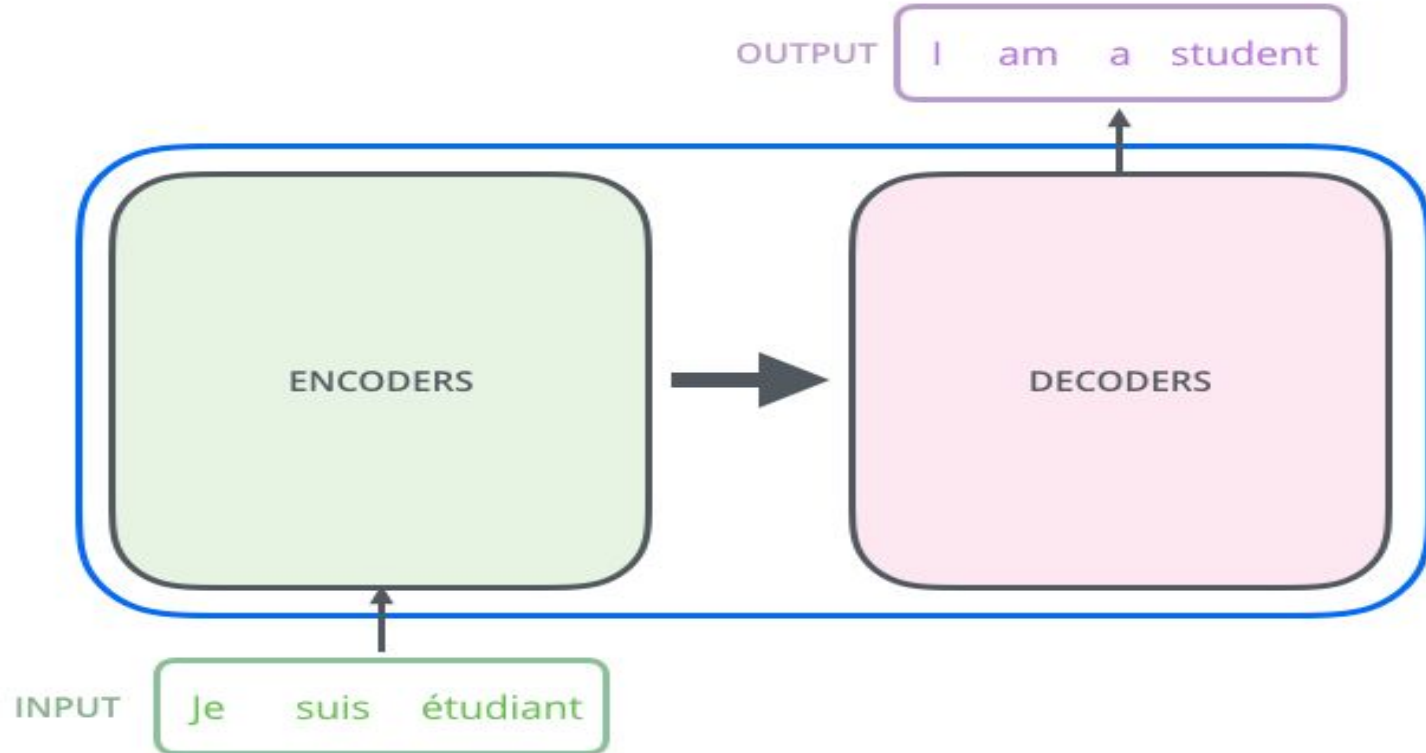
Transformer Architecture



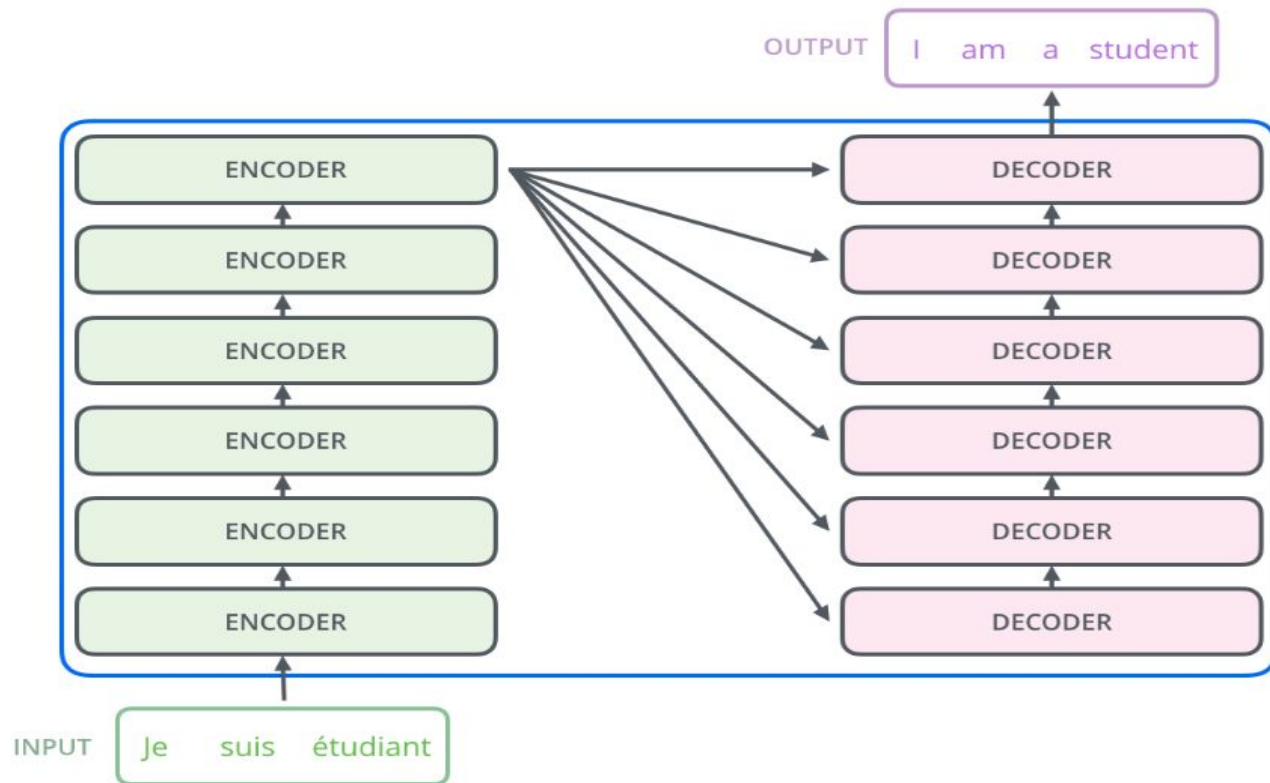
View 1: Black Box



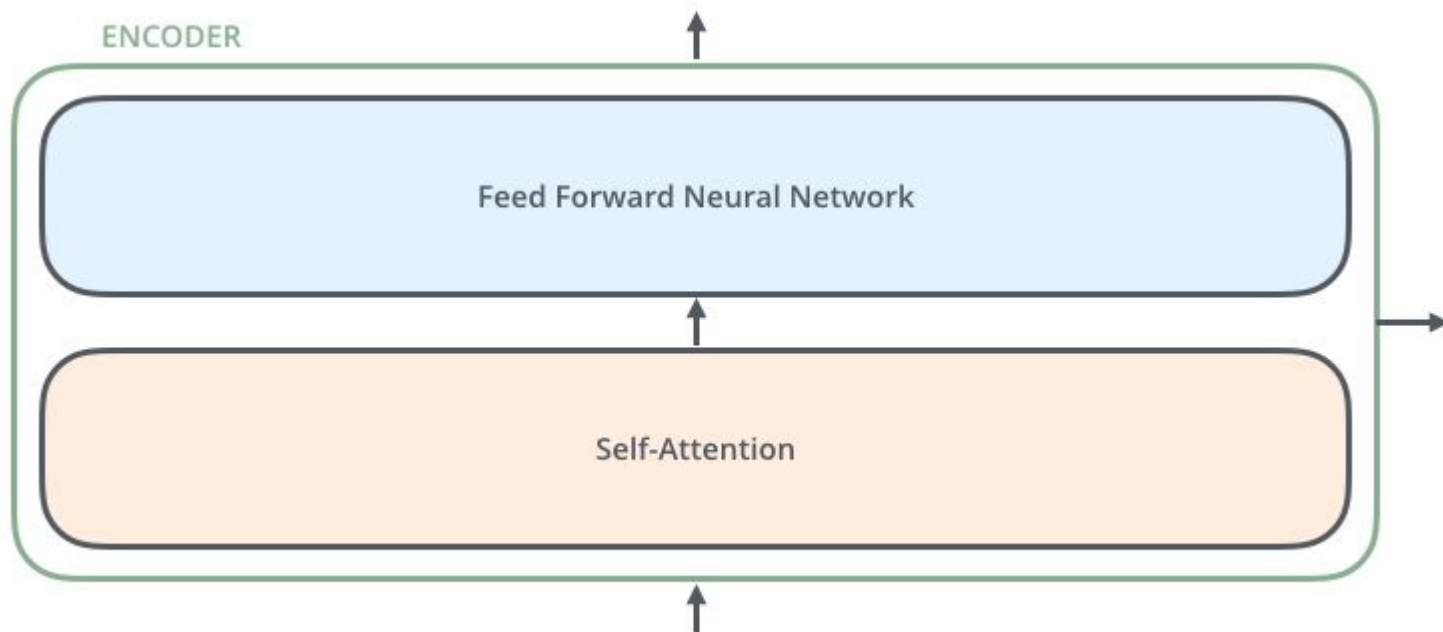
View 2: Adding Encoder & Decoder Stacks



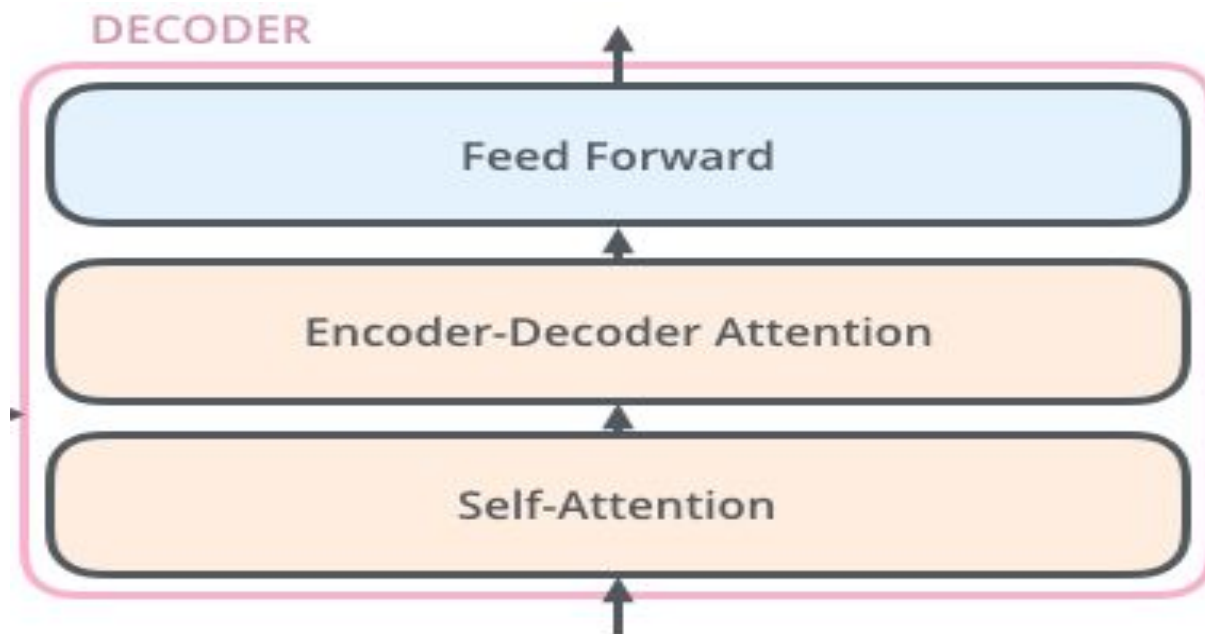
View 3: Expanding Encoder & Decoder Stacks



Encoder



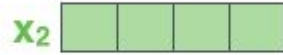
Decoder



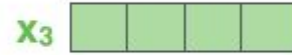
Embedding of texts



Je

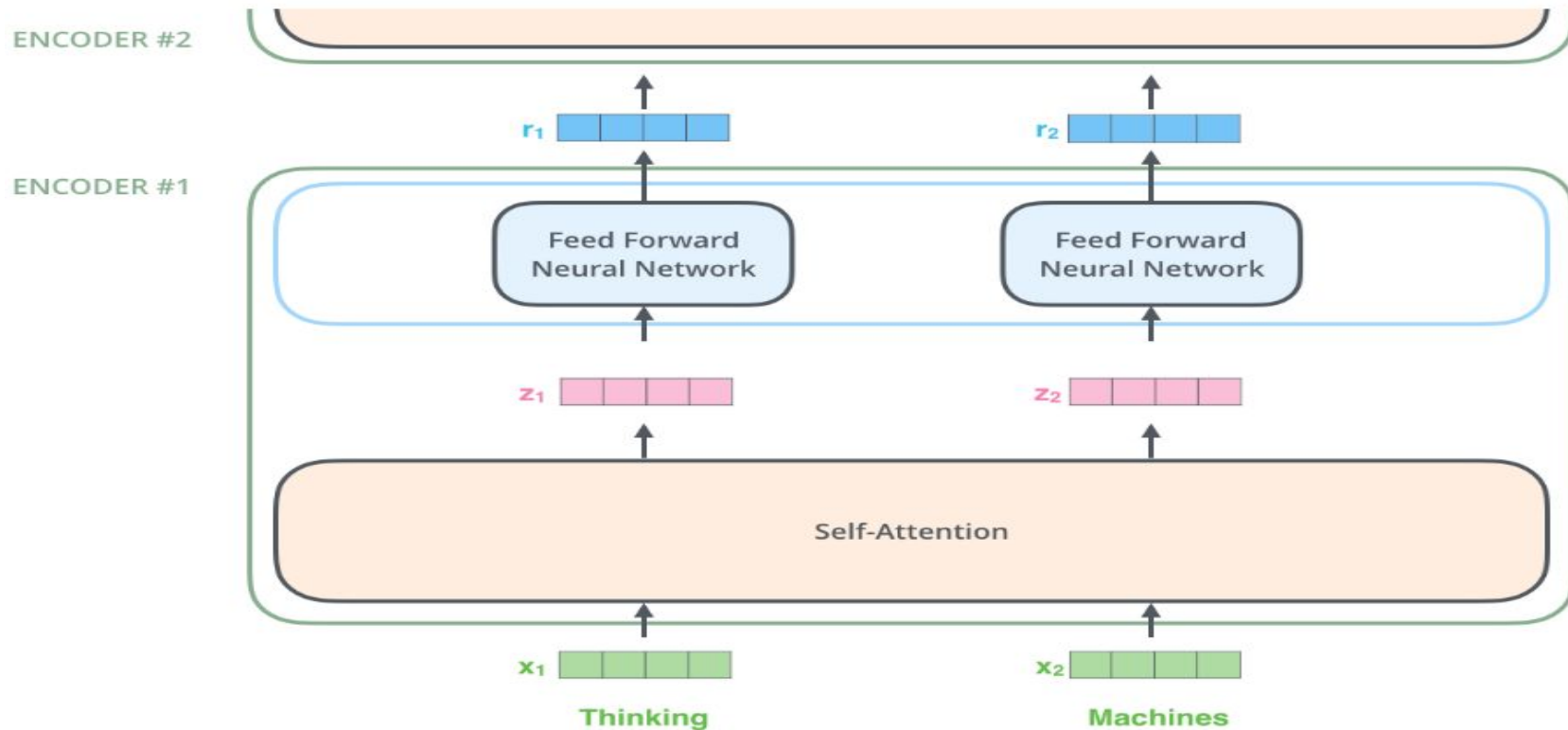


suis



étudiant

Feed Embeddings to Encoder



Self Attention

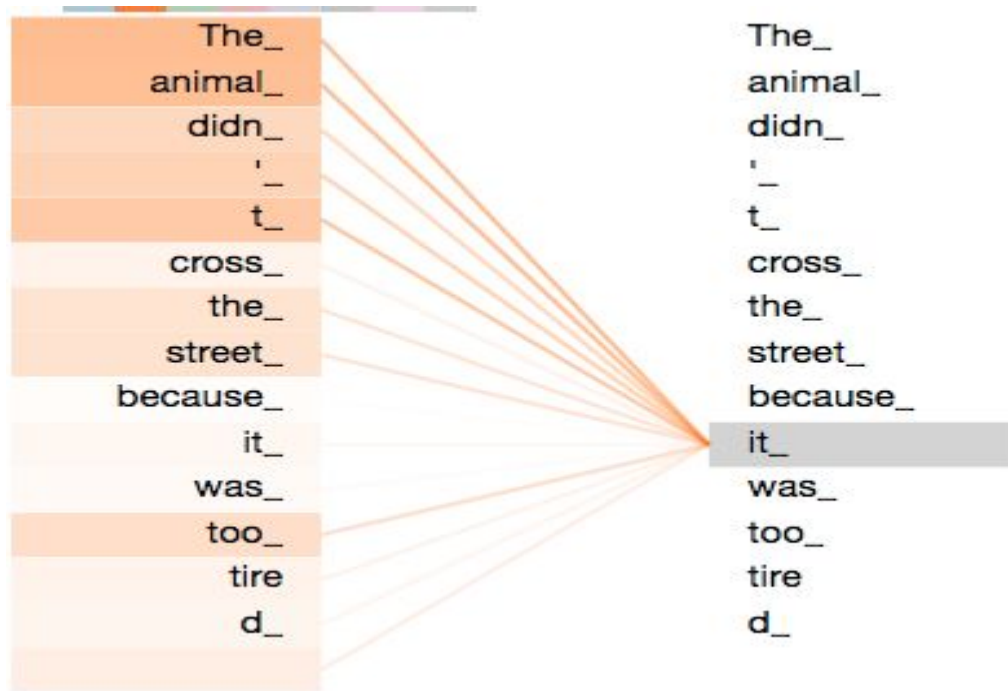
"The animal didn't cross the street because it was too tired"

What does **it** refers to? (animal or street)

Easy for human but not easy for algorithm

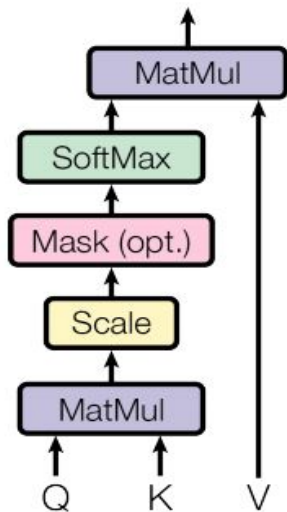
Goal of self-attention: Associate **it** with **animal** via looking at other positions in the input sequence

Self Attention (Contd...)

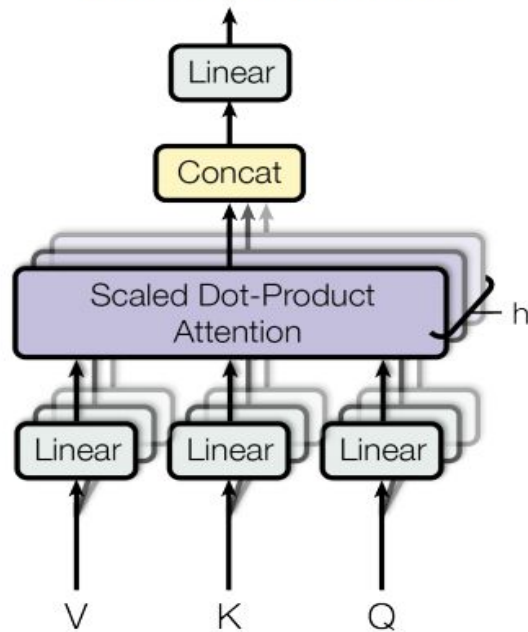


Self Attention Architecture

Scaled Dot-Product Attention



Multi-Head Attention



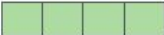
Self Attention Calculations (Step 1)

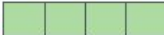
Input

Thinking

Machines

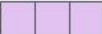
Embedding

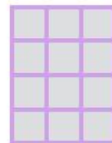
x_1 

x_2 

Queries

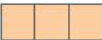
q_1 

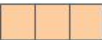
q_2 



W^Q

Keys


k_1 

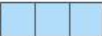
k_2 

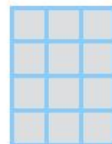


W^K

Values

v_1 

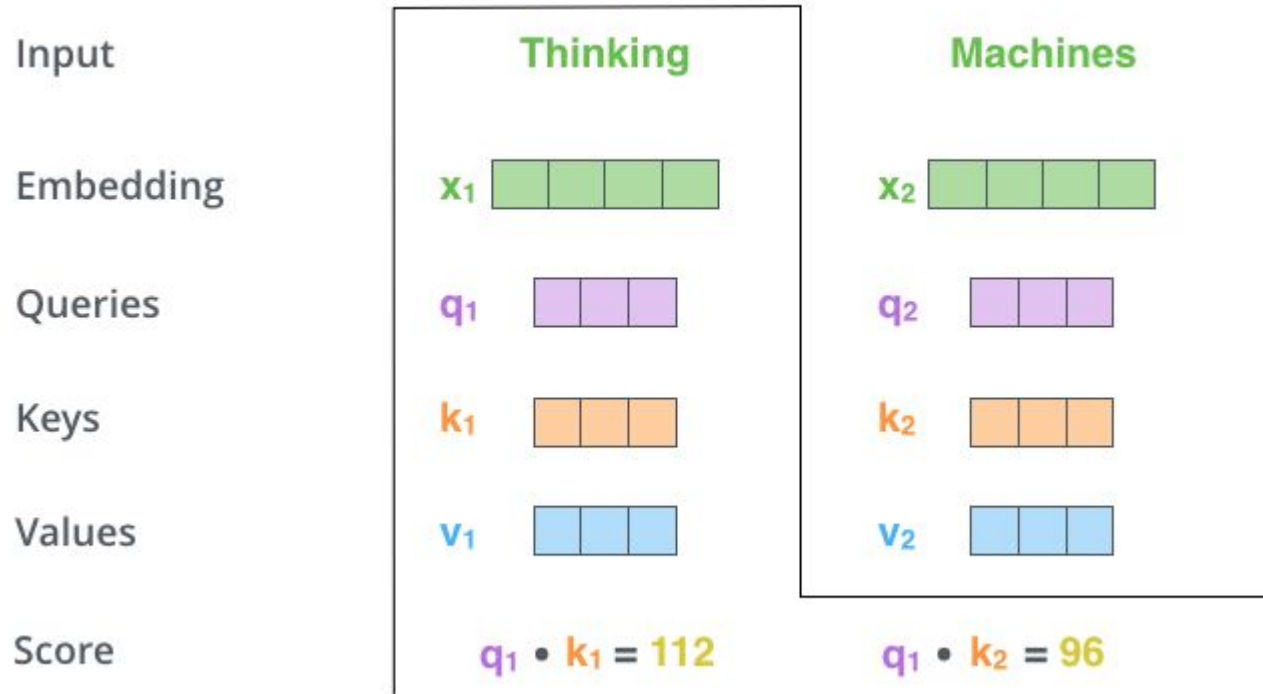
v_2 



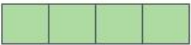
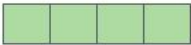
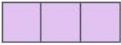
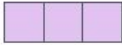
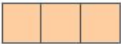
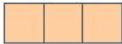

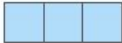
W^V

Queries, Keys, Values
are abstractions that are
useful for calculating and
thinking about attention

Self Attention Calculations (Step 2)



Self Attention Calculations (Step 3 & 4)

Input	Thinking	Machines
Embedding	x_1 	x_2 
Queries	q_1 	q_2 
Keys	k_1 	k_2 
Values	v_1 	v_2 
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)	14	12
Softmax	0.88	0.12

Self Attention Calculations (Step 5 & 6)

Input

Embedding

Queries

Keys

Values

Score

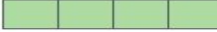
Divide by 8 ($\sqrt{d_k}$)

Softmax

Softmax
X
Value

Sum

Thinking

x_1 

q_1 

k_1 

v_1 

$q_1 \cdot k_1 = 112$

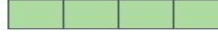
14

0.88

v_1 

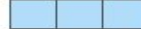
z_1 

Machines

x_2 

q_2 


k_2 

v_2 

$q_2 \cdot k_2 = 96$

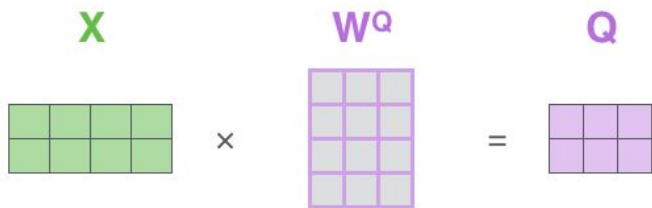
12

0.12

v_2 

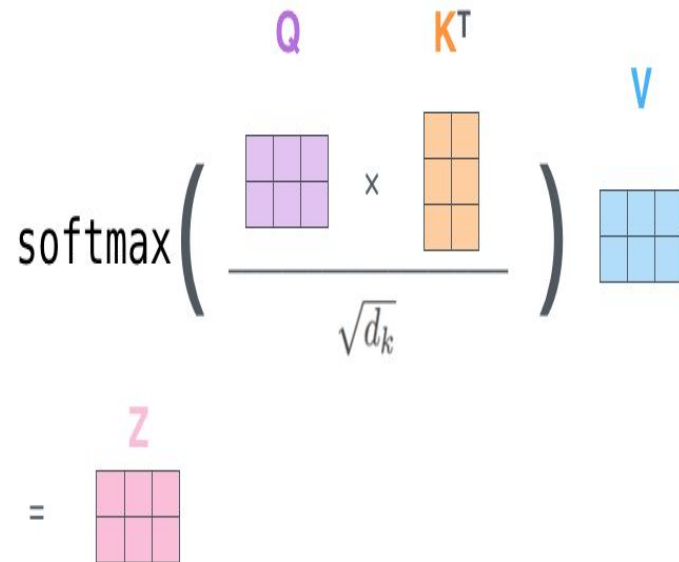
z_2 

Actual Implementation using Matrices

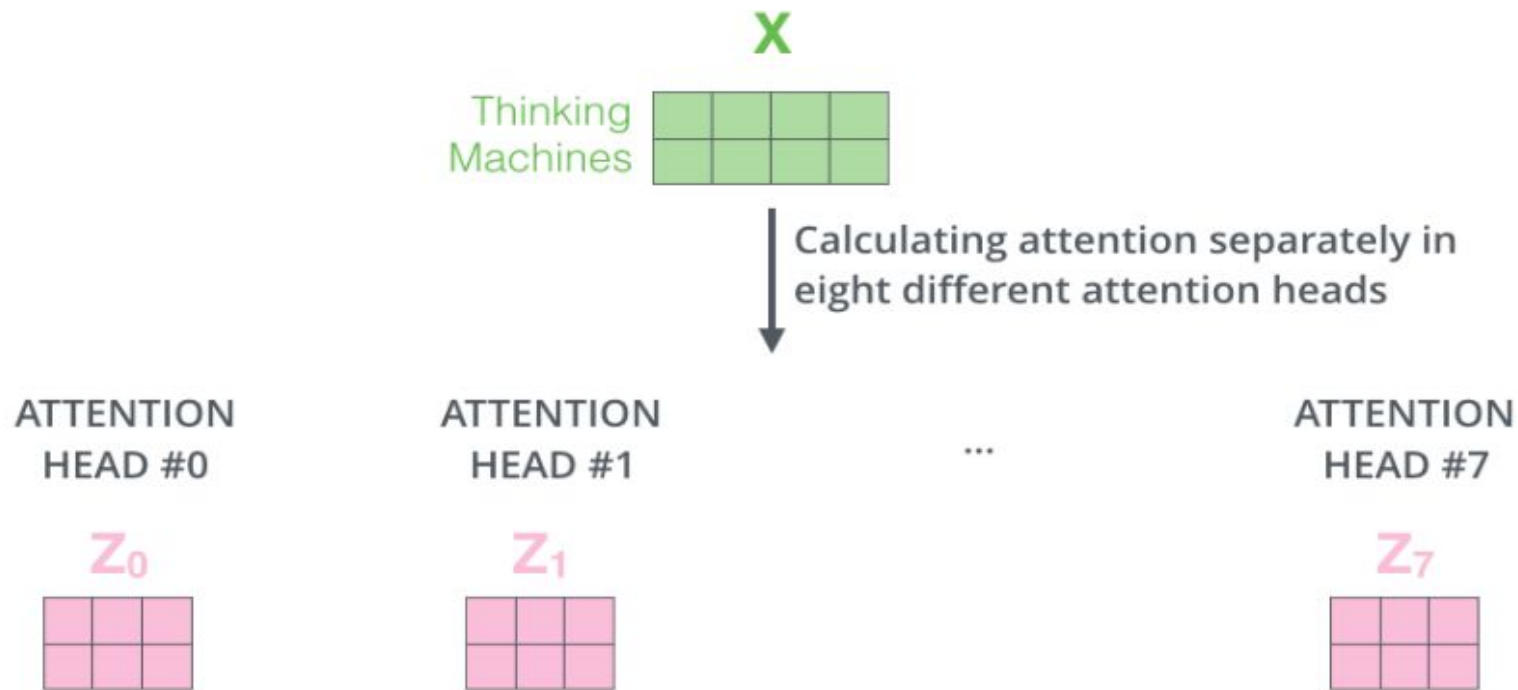
$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$


$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$


$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

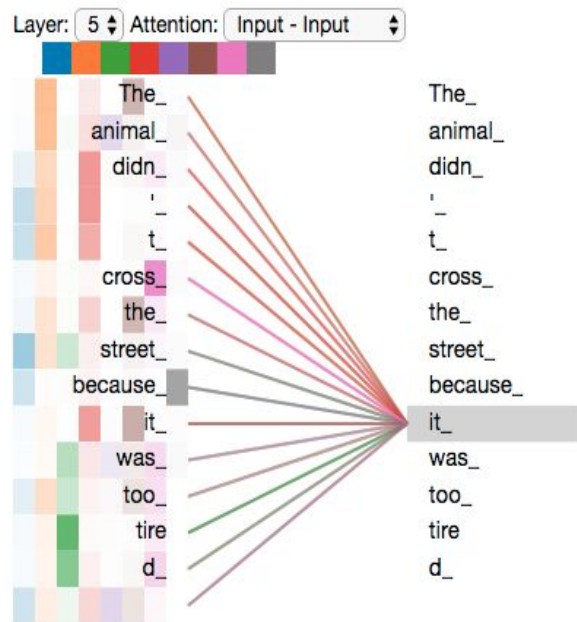
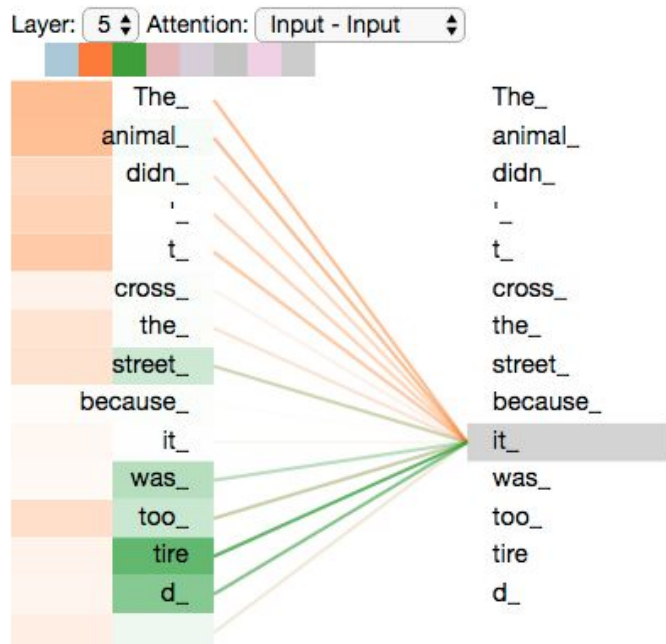

$$\text{softmax}\left(\frac{\mathbf{Q} \times \mathbf{K}^T}{\sqrt{d_k}}\right) \mathbf{V} = \mathbf{Z}$$


Multihead Self Attention



Why Multihead Self Attention?

Jointly attend to information from different representation subspaces



Feedforward Network

The feed-forward layer is not expecting eight matrices – it's expecting a single matrix (a vector for each word). So we need a way to condense these eight down into a single matrix.

Solution: Concat the matrices and then multiply them by an additional weight matrix

Feedforward Network

1) Concatenate all the attention heads

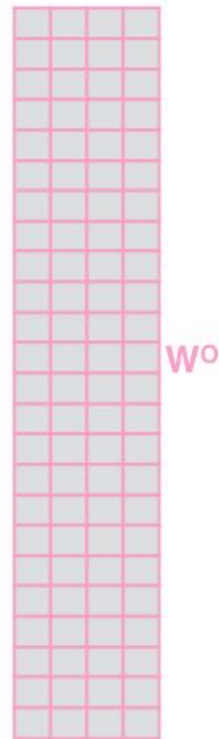


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

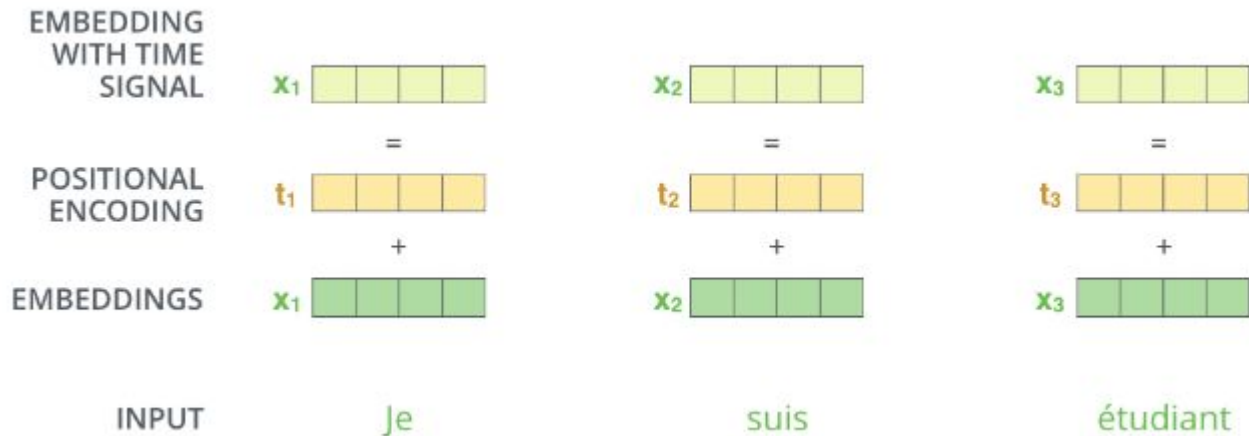


2) Multiply with a weight matrix W^O that was trained jointly with the model

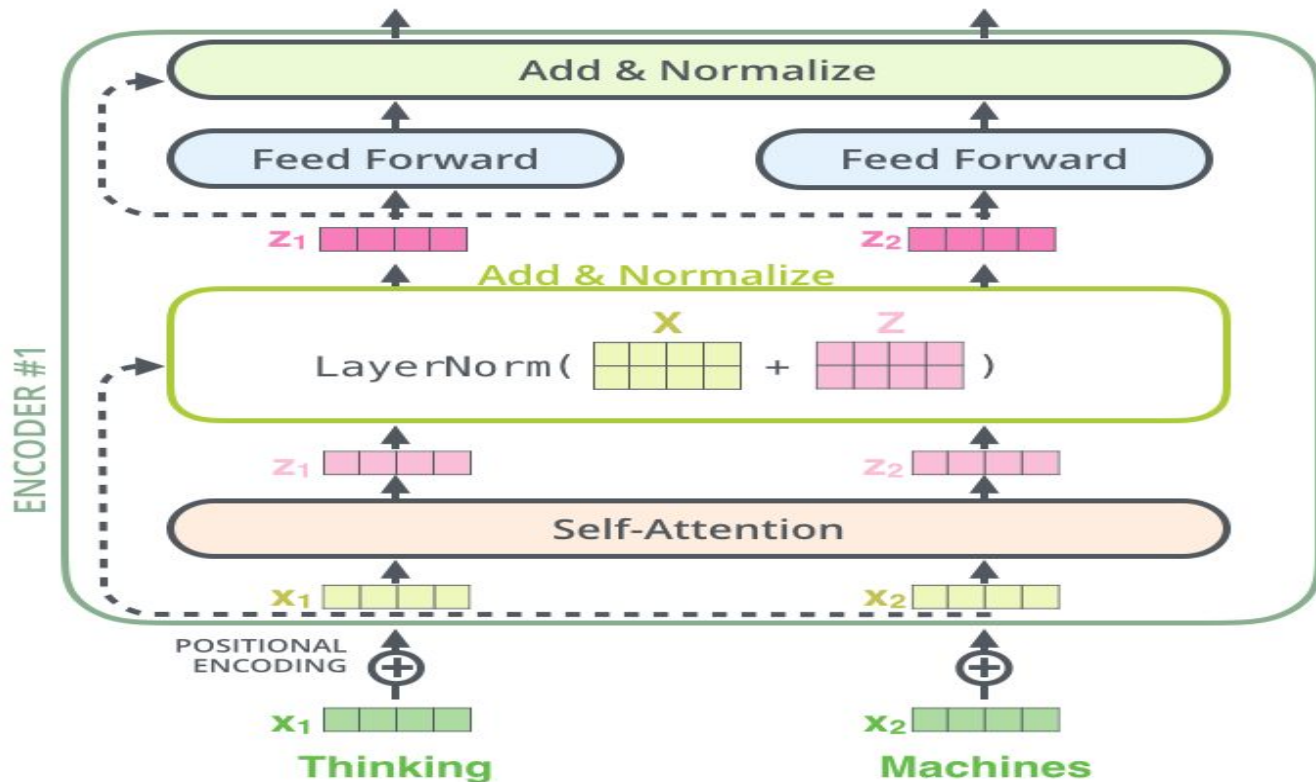
X



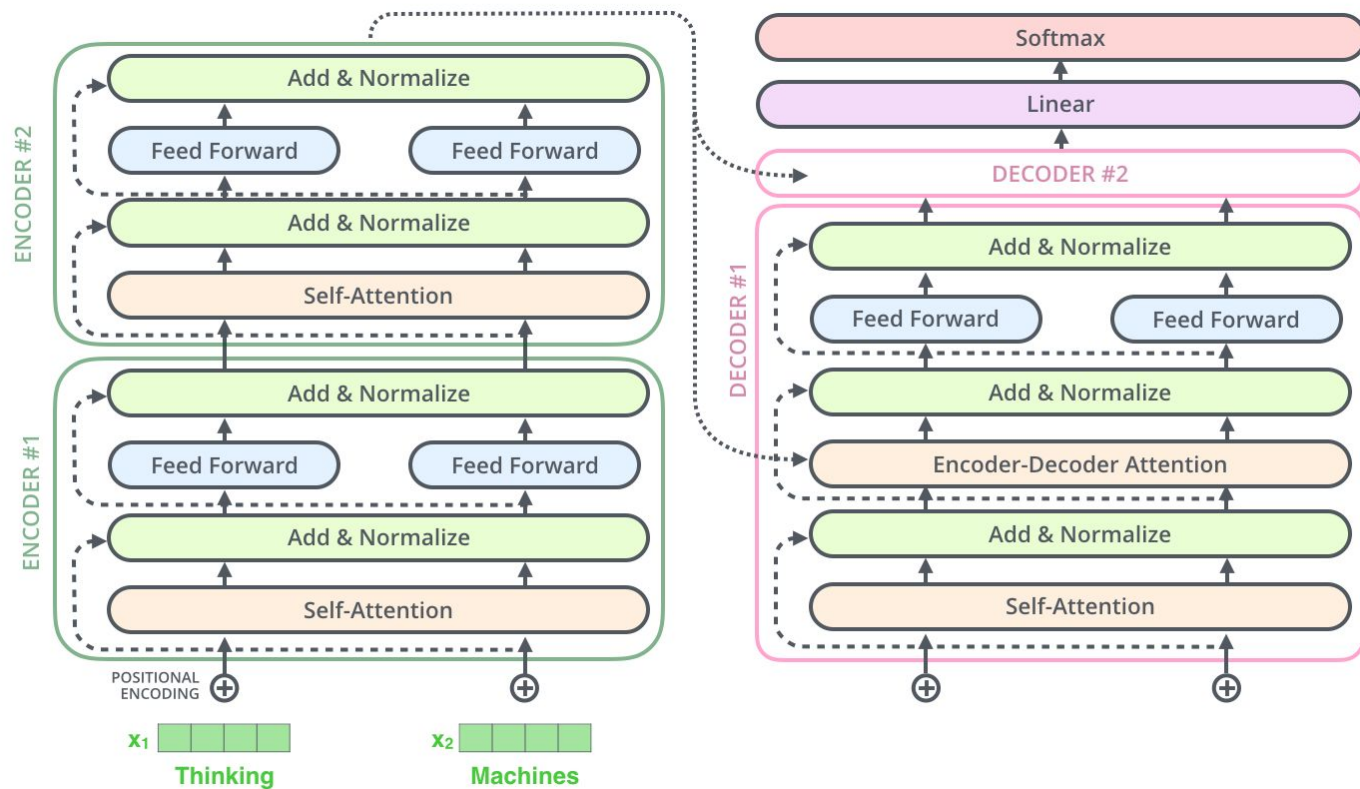
Adding Positional Embeddings to Input



Layer Normalization & Residual Connection



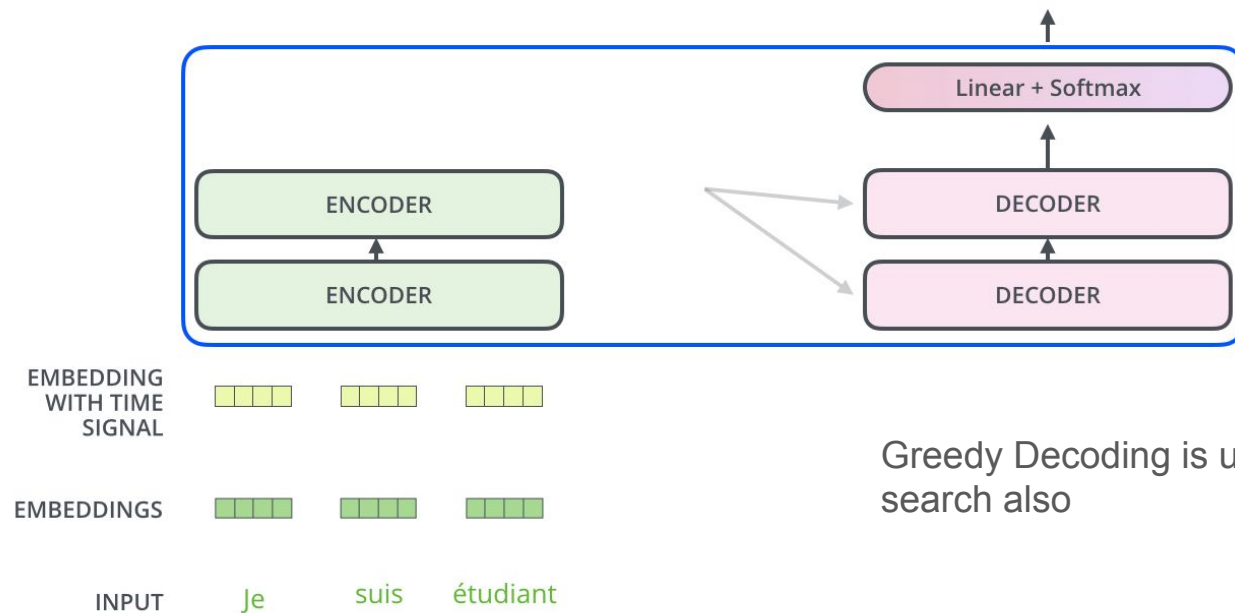
Decoder



Decoding (t=1)

Decoding time step: 1 2 3 4 5 6

OUTPUT

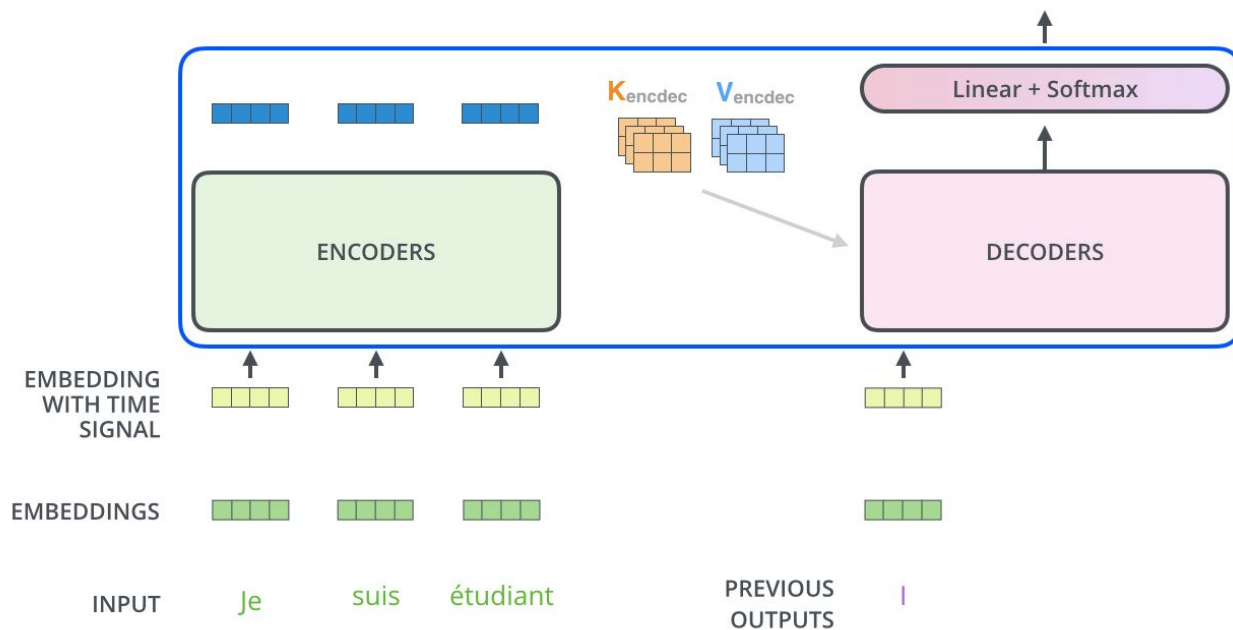


Greedy Decoding is used. But can use beam search also

Decoding (t=2 onwards)

Decoding time step: 1 2 3 4 5 6

OUTPUT |



Self Attention in Decoder

It is only allowed to attend to **earlier positions** in the output sequence. This is done by masking future positions (setting them to `-inf`) before the softmax step in the self-attention calculation.

Encoder-Decoder Attention

The Encoder-Decoder Attention layer works just like multiheaded self-attention, except it creates its **Queries matrix from the layer below it**, and takes the **Keys and Values matrix from the output of the encoder stack**.

Final Linear & Softmax Layer

The decoder stack outputs a vector of floats. **How do we turn that into a word?**
Using **Linear Layer which is followed by a Softmax Layer**.

Linear Layer: It is fully connected neural network that **projects** the vector produced by the stack of decoders, into a much, much larger vector called a **logits vector**.

Let's assume that our model knows 10,000 unique English words that it's learned from its training dataset. This would make the logits vector 10,000 cells wide – **each cell corresponding to the score of a unique word**. The **softmax layer then turns those scores into probabilities** (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

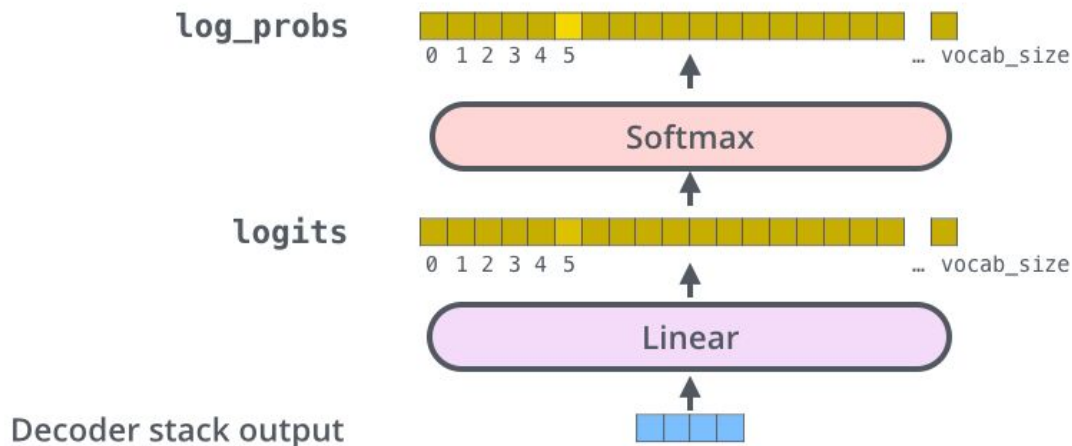
Final Linear & Softmax Layer (Contd...)

Which word in our vocabulary
is associated with this index?

am

Get the index of the cell
with the highest value
(argmax)

5



Vocabulary Index

The output vocabulary of our model is created in the preprocessing phase before we even begin training.

Output Vocabulary

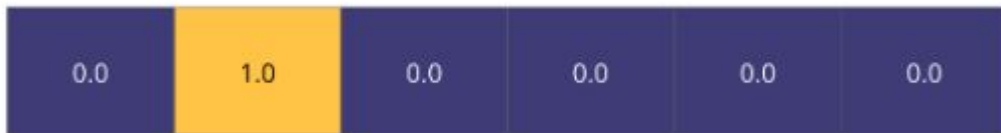
WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

Training Transformer Network

During training, an untrained model would go through the **exact same forward pass** as discussed till now. But since we are training it on a labeled training dataset, we can **compare its output with the actual correct output** and optimize parameters using **SGD** and **backpropagation**.

Training Transformer Network (Contd...)

One-hot encoding of the word "am"



Untrained Model Output



Correct and desired output



a

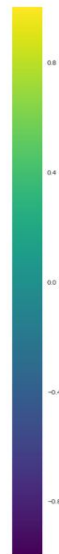
am

I

thanks

student

<eos>



Loss Function

How do you compare two probability distributions?

1. Cross-entropy
2. Kullback–Leibler divergence

NLP Language Representation

NLP Tasks

To solve the NLP tasks:

1. Either learn **language representations** from **scratch** or learn it beforehand via **pre-training** followed by **fine-tuning**
2. Pre-training methods have been shown to achieve state-of-the-art results in many tasks

Pre-training of Model

The **objective during pre-training** (to learn general language representations) is:

1. Unidirectional Language Modelling

- a. Limits the choice of architectures that can be used during pre-training
- b. Not suitable for token-level tasks like question answering, where it is crucial to incorporate context from both directions
- c. E.g. OpenAI GPT

2. Masked Language Modelling (MLM)

- a. Considers both left and right context deeply
- b. E.g. BERT

3. Next Sentence Prediction (NSP)

- a. Jointly pretrains text-pair representations
- b. E.g. BERT

How to use pre-trained language model for downstream task?

1. **Feature based:** Includes task-specific architectures that include the pre-trained representations as additional features. E.g. ELMo
2. **Fine tuning:** Introduces minimal task-specific parameters and is trained on the downstream tasks by fine-tuning. E.g. BERT, GPT

Text Embedding History

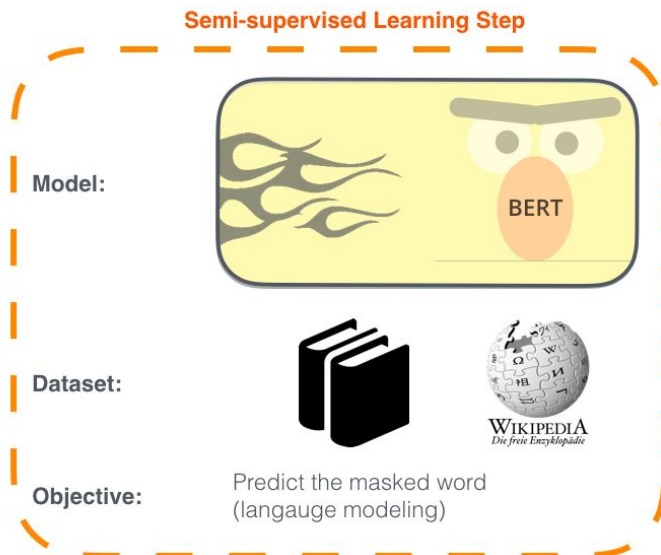
1. Learned during NLP task - trained on limited data
2. Use pre-trained embedding like **word2vec** and **GloVe** - trained on large data
3. **ELMO**: Context matters
 - a. Uses a bi-directional LSTM trained on a specific task (LM) to be able to create those embeddings
4. **OpenAI Transformer**: Pre-training a Transformer Decoder for Language Modeling
 - a. The decoder is a good choice because it's a natural choice for language modeling (predicting the next word) since it's built to mask future tokens
 - b. Decoder layers would not have the encoder-decoder attention sublayer that vanilla transformer decoder layers have
 - c. **Unidirectional**
5. **BERT**: From decoder to encoder

BERT

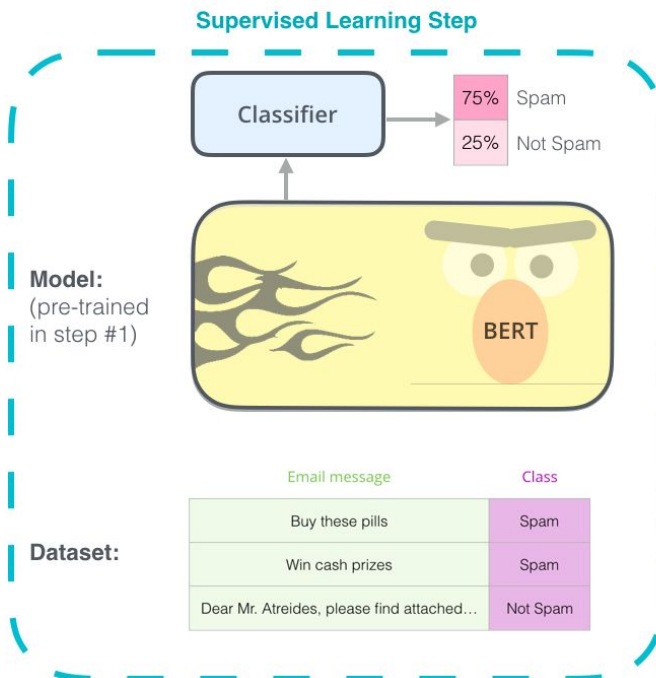
BERT Workflow

1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.



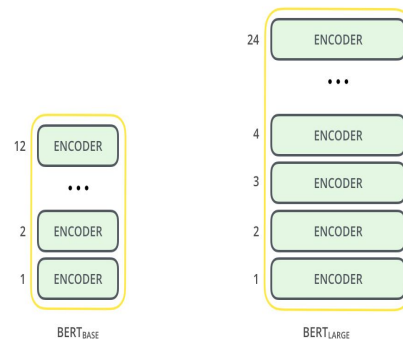
BERT Architecture

1. A distinctive feature of BERT is its **unified architecture across different tasks**. There is minimal difference between the pre-trained architecture and the final downstream architecture.
2. Two architecture (multi-layer bidirectional Transformer encoder):
 - a. BERT_{BASE} (L=12, H=768, A=12, Total Parameters=110M)
 - b. BERT_{LARGE} (L=24, H=1024, A=16, Total Parameters=340M)

L: #Transformer blocks

H: hidden size of FFNN

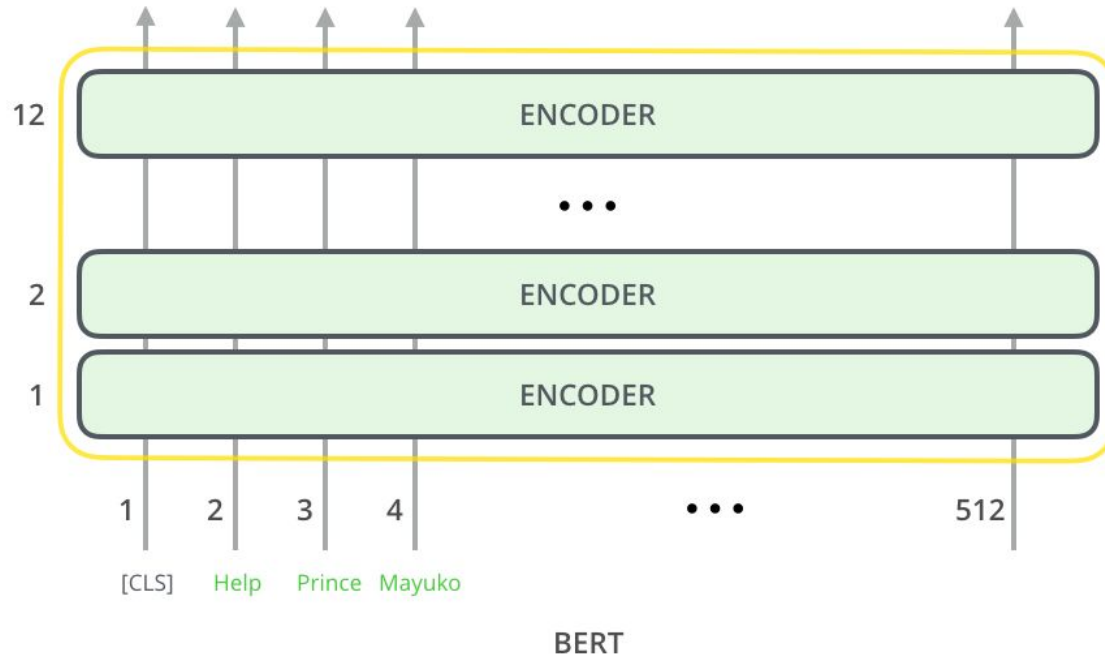
A: number of self-attention heads



Input/Output Representations

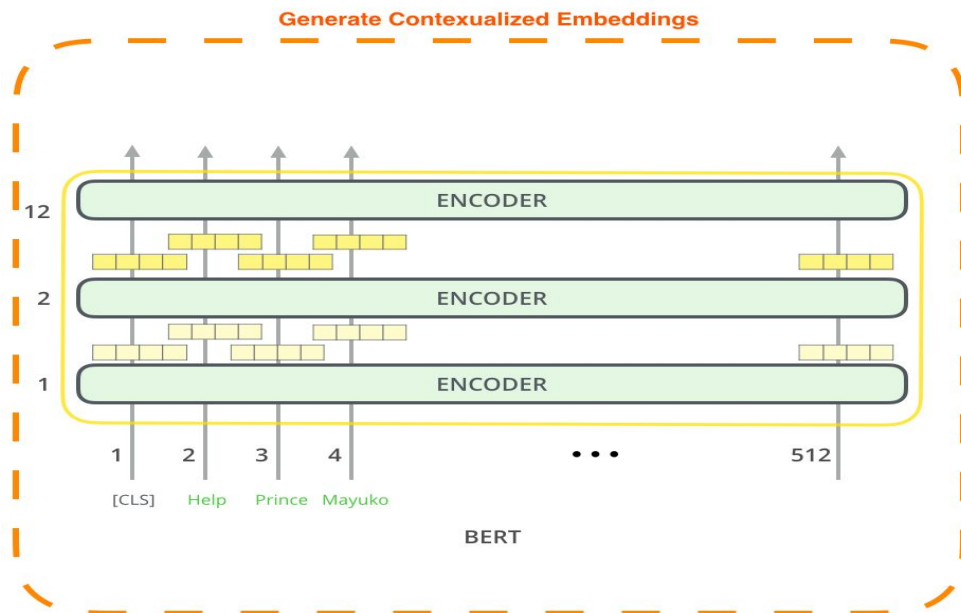
1. Input representation is able to unambiguously represent both a **single sentence** and a **pair of sentences** in one **token sequence**
2. **Sentence** can be an arbitrary **span of contiguous text**, rather than an actual linguistic sentence. A “sequence” refers to the input token sequence to BERT, which may be a single sentence or two sentences packed together.
3. The **first token** of every sequence is always a special classification token (**[CLS]**). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks.

BERT Input

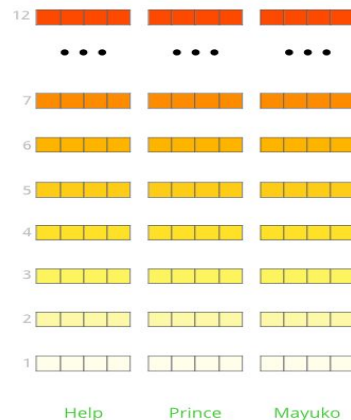


BERT Output

Contextual Embeddings



The output of each encoder layer along each token's path can be used as a feature representing that token.












But which one should we use?

BERT Output (Contd...)

What is the best contextualized embedding for “**Help**” in that context?

For named-entity recognition task CoNLL-2003 NER

		Dev F1 Score
12 	First Layer	91.0
...	Last Hidden Layer	94.9
7 		
6 	Sum All 12 Layers	95.5
5 		
4 		
3 	Second-to-Last Hidden Layer	95.6
2 		
1 	Sum Last Four Hidden	95.9
		
Help	Concat Last Four Hidden	96.1

Pre-training BERT

Two tasks are used for pre-training:

1. Masked Language Modelling
2. Next Sentence Prediction

Masked LM

Sentence: I love to code in python

Masked Sentence: I [MASK] to code in python

[MASK] means token is missing

Task: Predict the [MASK] (in this case [MASK] will be love)

Goal: To understand relationship between words

Note: We randomly choose 15% words in each sentence and replace it with:

1. [MASK] token 80% of the time
2. A random token 10% of the time
3. Unchanged 10% of the time

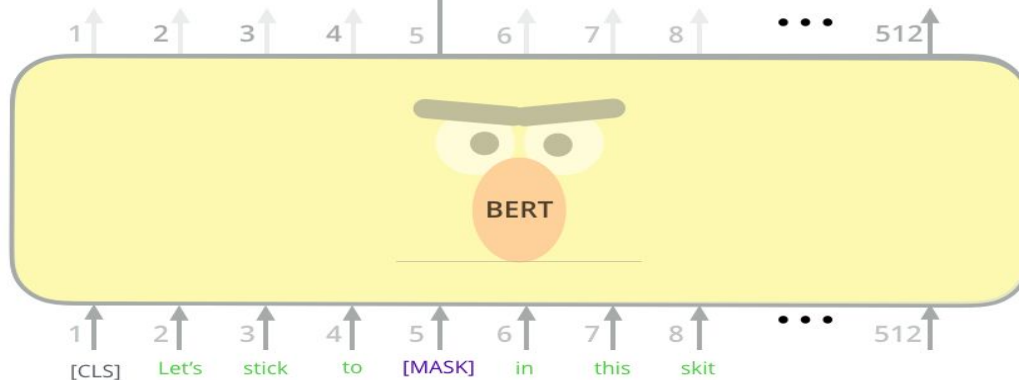
Training Task: MLM (predicting mask words)

Use the output of the masked word's position to predict the masked word

Possible classes:
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzzzyva

FFNN + Softmax



Randomly mask
15% of tokens

Input

[CLS] Let's stick to improvisation in this skit

Next Sentence prediction

Goal: To understand the relationship between sentences

Given 2 sentences A & B:

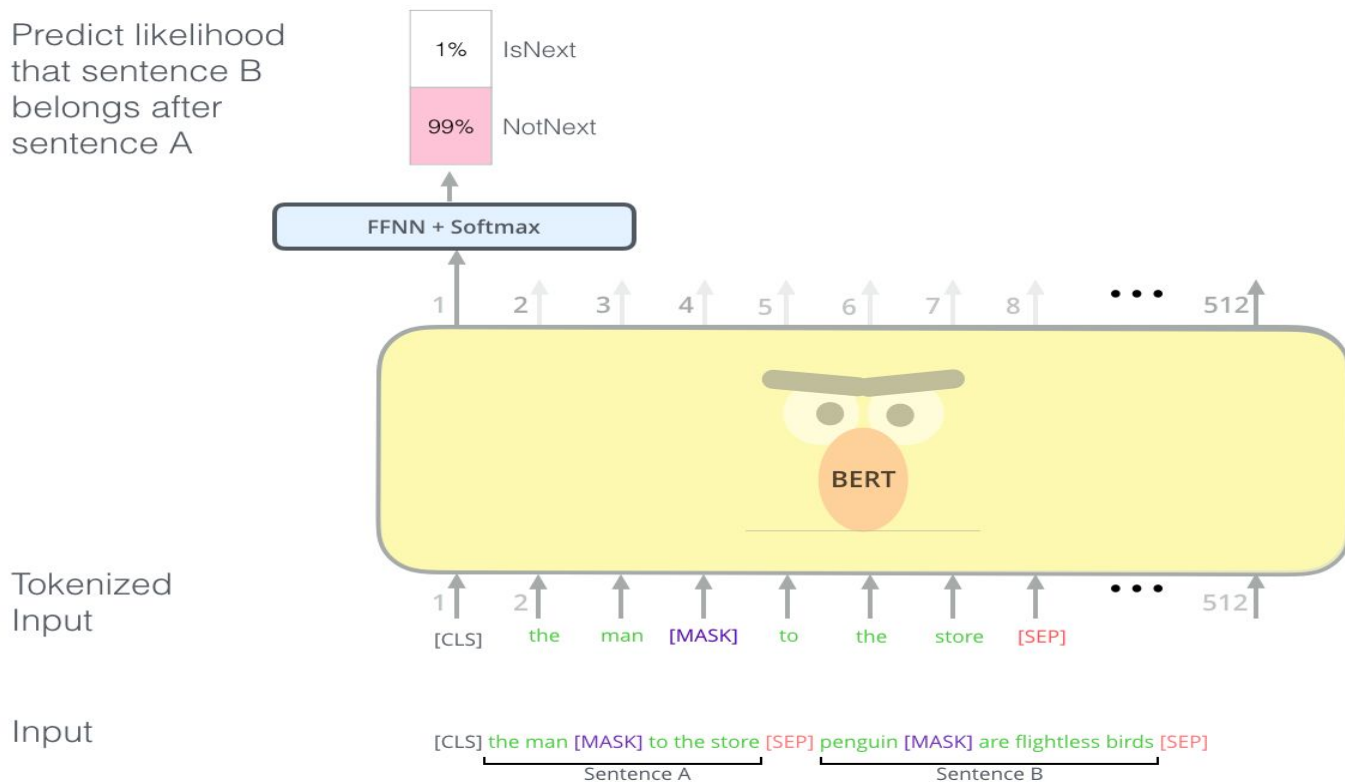
Task: Is B the actual next sentence in the corpus (Simple binary classification

With 2 labels **IsNext** and **NotNext**)

Specifically, when choosing the sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext)

Training Task: NSP (binary classification task)

Predict likelihood
that sentence B
belongs after
sentence A



Pre-training Dataset

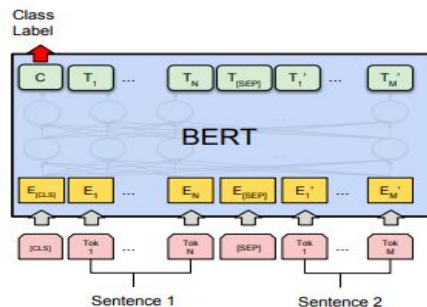
1. BooksCorpus (800M words)
2. English Wikipedia (2,500M words)

Fine-tuning BERT

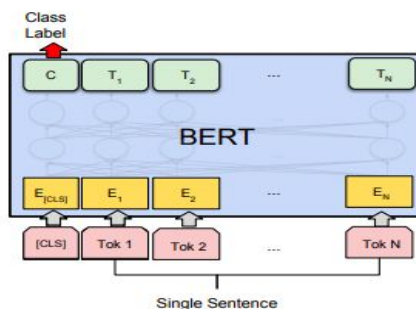
1. Self attention mechanism in the Transformer allows BERT to model many downstream tasks
2. For each task, we simply plug in the task-specific inputs and outputs into BERT and finetune all the parameters end-to-end.

Task	A	B
Textual entailment	Hypothesis	Premise
Q & A	Questions	Passage
Classification	Text	None

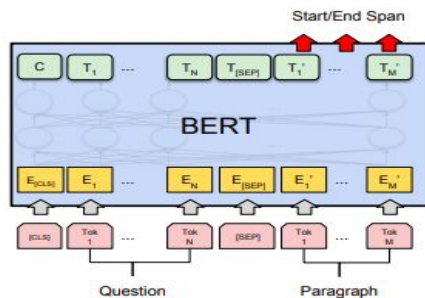
Fine Tuning for various Tasks



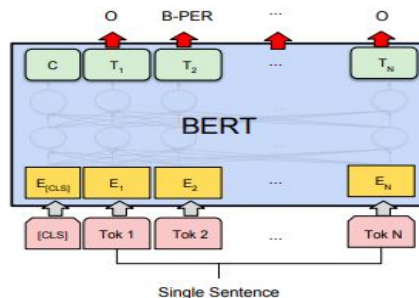
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



(b) Single Sentence Classification Tasks:
SST-2, CoLA



(c) Question Answering Tasks:
SQuAD v1.1



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

cuBERT

1. Everything is same except **dataset** (how we adapt to this dataset on input side)

codeBERT

1. Everything is same except
 - a. **Dataset**
 - b. One additional training task (**Replaced token detection (RTD)** in place of **NSP** which is similar to NSP)
 - c. how we adapt to this dataset and additional training task (code modifications on BERT)

Dataset & Pre-trained Models

Husain et al. (2019) provides open dataset (used in CodeBERT):

1. Includes 2.1M bimodal data-points
2. 6.4M unimodal codes across six programming languages (Python, Java, JavaScript, PHP, Ruby, and Go)

Google BigQuery has (used in cuBERT):

1. Github Data

No one open sourced their pretrained model till date. Good to contribute first pretrained model! An opportunity to do some good open source work along with MTP.....

Thank You

References

1. Ashish Vaswani et. al, Attention is all you need, NIPS 2017
2. Jacob Devlin et. al, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, arXiv 2018
3. [Jay Alammur Blog](#)