Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin

Reimplementation & Demo:
Adel Alkhamisy
Aniket Pandey

Motivation

- In our NLP class we worked on Dialectal Question Answer using BERT which is based on this Transformer.
- Breakthrough for NLP especially machine translation and state-of-art.
- Transformers were developed to solve the problem machine translation and sequence transduction, or neural machine translation... but they do so much more!
  - Great performance in computer vision like ViT (Dosovitskiy et al., 2020)
  - Image Classification (CoCa Transformer)
  - Semantic Segmentation (e.g.: FD-SwinV2-G Transformer)
  - Object Detection (e.g.: FD-SwinV2-G Transformer)

Context

What issue is the paper trying to solve?

- Previous work RNN, LSTM, and GRU slow to train. Factorization trick helps but still slow.
- Suffers from exploding and vanishing gradients.
- Cannot handle very long-term dependency.
  - In Seq-Seq models, decoder only accesses last hidden state. Early information in sentence can be lost.

Question?

How would a RNN, LSTM, and GRU do word masking?

1. Encode word inside cell state.
2. Transfer cell state from one word to another.
3. Decode word from previous cell state.

Definitely not ideal! There has to be a better way...

Self-Attention

- Each element attends to every other element.
- Each element becomes query, key, and value from the input embeddings by multiplying by a weight matrix.

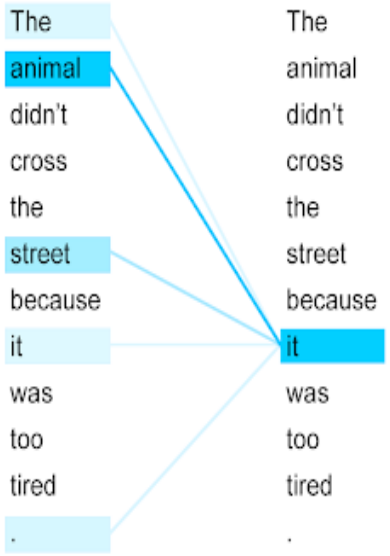
Idea Behind Attention?

Learn how to pick relevant information from input data.

1. Create three vectors from each of encoder's input value (query, key, value).
2. Calculate a score for how much to focus on each part of the input when we encode words at specific positions.
3. How? Actual math was covered in class lecture.

key	value
name	Quinn
position	quarterback
handedness	right

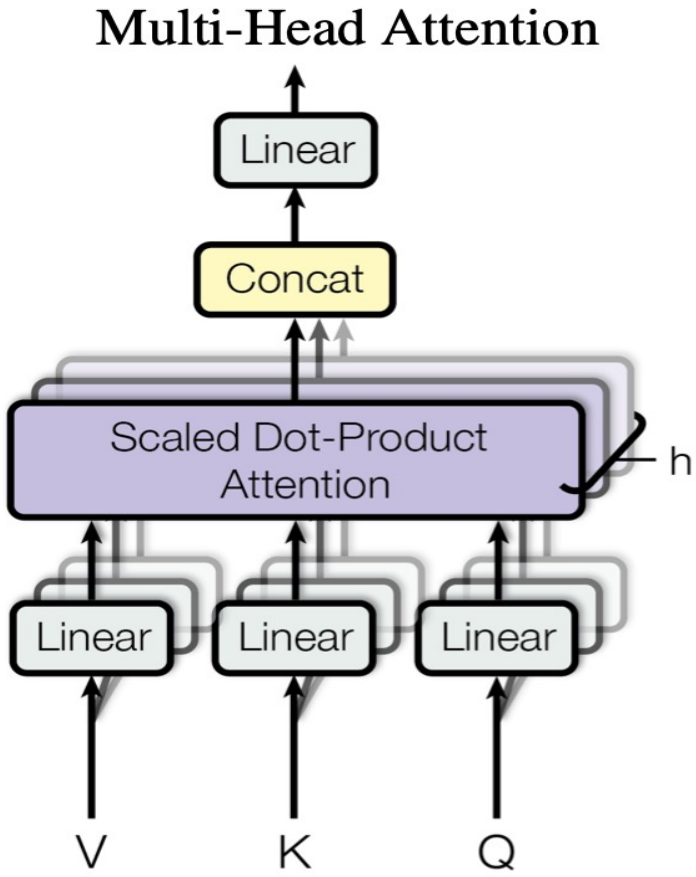
Keys and values are actually word embeddings, not words.



Source: Google Blog

Multi-Head Attention

- Idea:
  1. Stack linear layers (weight matrices without biases) that are independent each for keys, queries, values.
  2. Concatenate output of attention heads to form (plus non-linearity) output layer.
- Why?
  1. Allows for model to focus on different positions.
  2. Gives attention layer multiple “representation subspaces”
  3. No longer need to oversaturate one attention mechanism.
- Key Point
  1. Calculate 8 different attention heads and combine them.
  2. Attention heads are independent of each other.



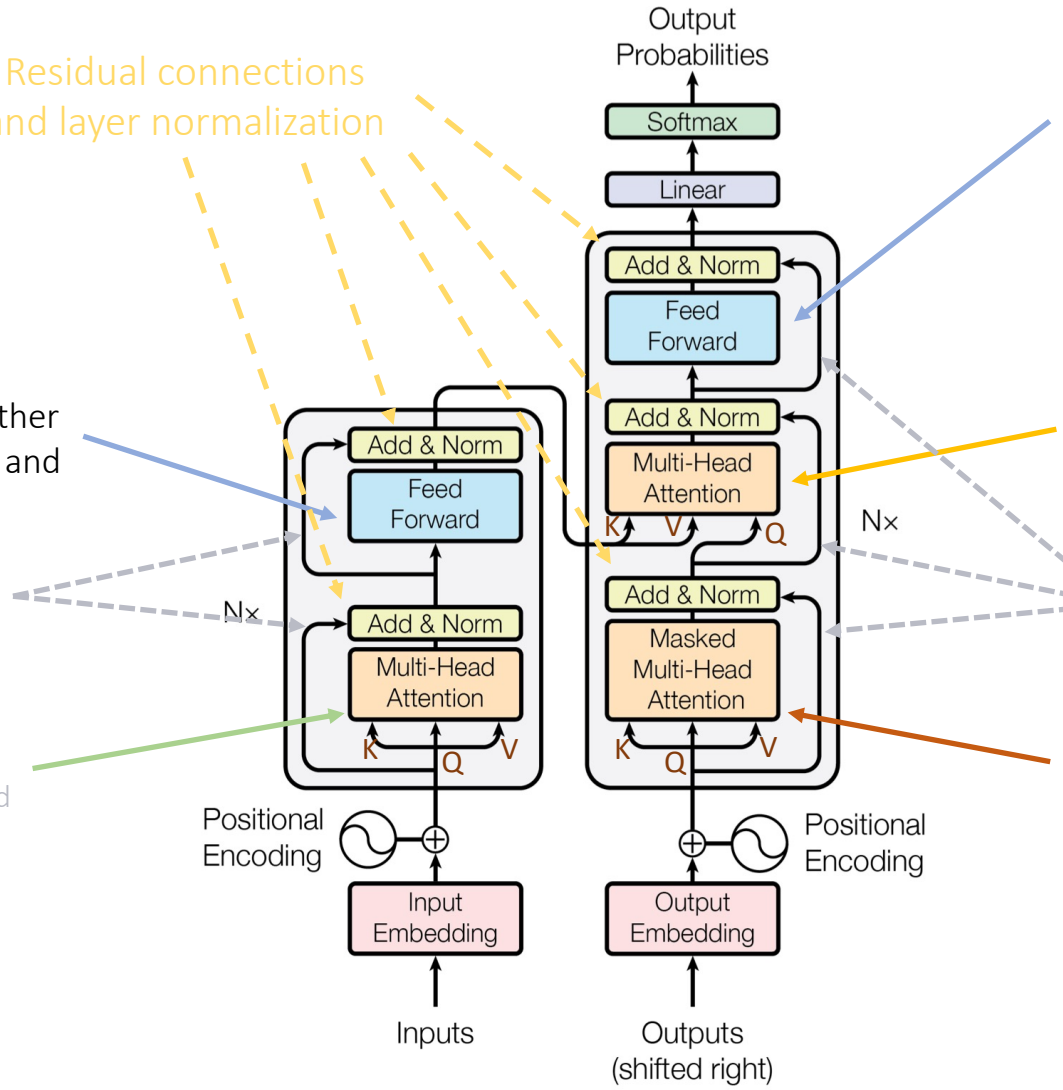
Source: Attention Is All You Need

Positional Encoding

- Need for information about the position and order of tokens in a sequence.
- Positional Embedding: Vector that represents position of each token.
  - Element wise addition the position embedding to the word embedding vector.
  - Positional embedding be fixed or learned.

From paper: “We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}”.

Complete Architecture



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Skip Connection: attention + query

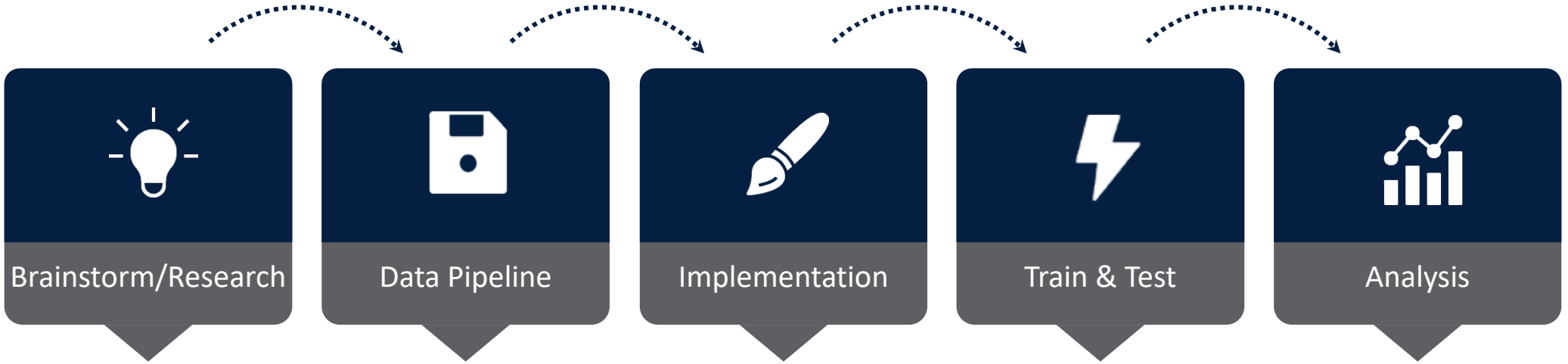
Encoder self-attention: tokens look at each other
queries, key, value are computed from encoder states

Decoder - Encoder self-attention: target tokens look at the source
queries – from decoder state; keys and values from encoder states

Skip Connection: attention + query

Decoder self-attention (masked): tokens look at previous tokens
queries, keys, values are computed from decoder states

Our Approach



Read the paper, watched YouTube videos, and tutorials to understand the details of model.

Created pipeline to efficiently read dataset into project and byte-pair encoding

Developed the model architecture as specified in paper.

Implemented the train pipeline to train the model and save the result. Also added a testing code to test the model and predict based on user input.

Used tensorboard to visualize the results and derived the output generated from the model.

Implementation

Multi30k
The dataset is based on WMT 2016 consists of English to German.

Added preprocessing pipeline and used the library spacy for tokenizing. Specifically, de_core_news_sm and en_core_web_sm. Added padding for start, end, and unknown. Also, finding insights so that Transformer block can be build accordingly.

DATA PIPELINE
Preprocessing

ATTENTION
Self & Multi Head Attention

Attention
Each element attends to every other element.

The self attention takes in embedding size and heads. The hidden dimension is calculated by dividing embedding size by heads. Make sure it is divisible by 8. e.g., 512//8. Add masking to key, value, and query.

Implementation

Encoder
Takes in Key, Value, and Query from input after positional encoding.

Stack of N=6 identical layers consisting of multi-head self attention and second is position wise fully connected feed-forward network followed by residual connection of layer normalization. This produces output of dimension d_{model}=512.

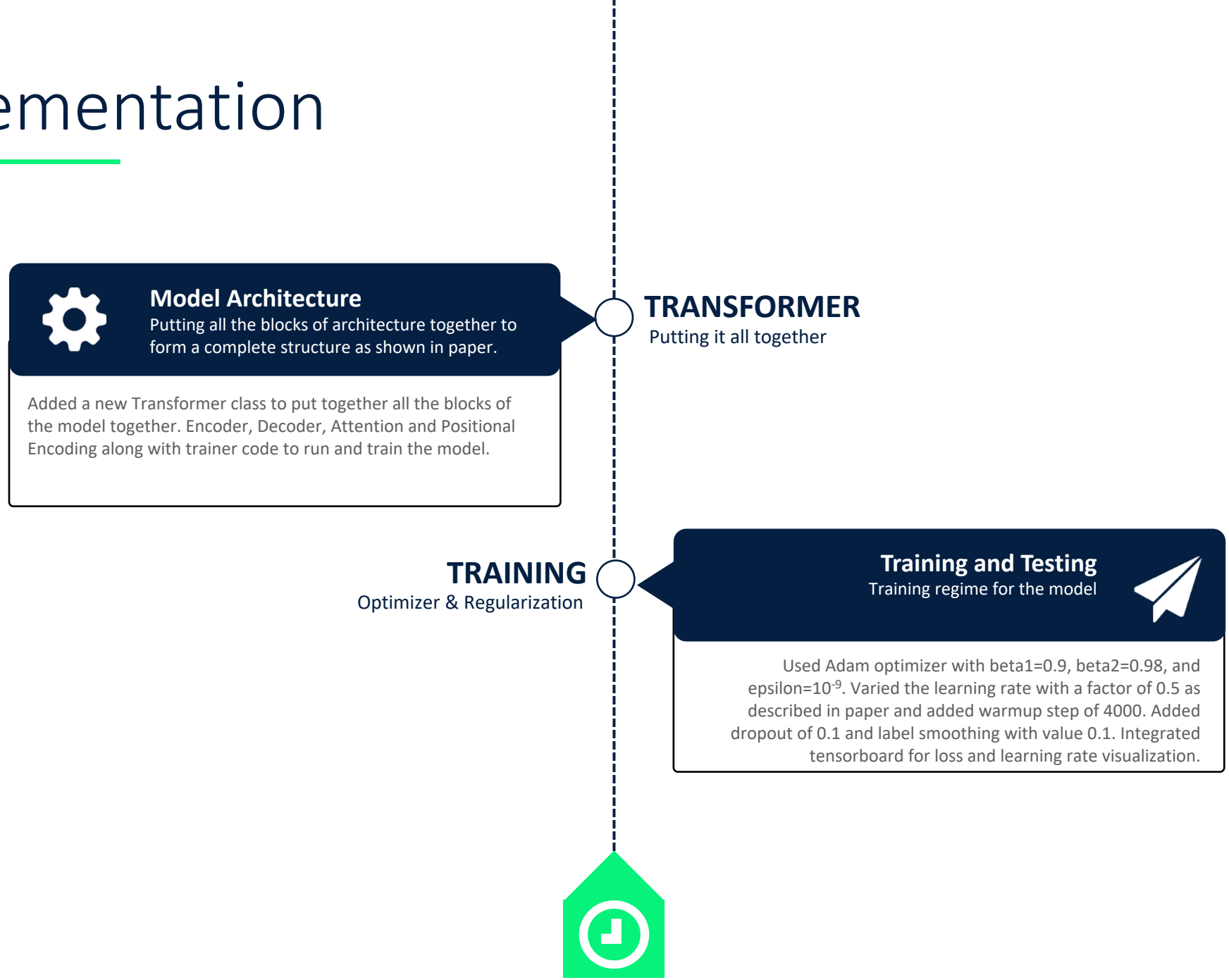
ENCODER BLOCK
6 layers of encoder

DECODER BLOCKS
6 layers of decoder

Decoder
Takes in output positional embedding and combines it to the output of encoder block.

Similar to encoder it is a stack of N=6 identical layers. In addition to two sublayers in the encoder block, it has a multi head attention layer over output of encoder block. Modify the self attention layer to prevent positions from attending to subsequent positions.

Implementation



Difference from Original Study

- The implementation is in Tensorflow.
- They use WMT 2014 English to German and English to French dataset
  - 4.5 million sentence pairs (en-de)
  - 36 million sentence pairs (en-fr)
- The author doesn't report their random seeding. And even if they did, even slight differences in the code structure could lead to different random sampling.
- For English to German, they train their base model for 100,000 steps or 12 hours.
- For English to French, they train the model 300,000 steps for 3.5 days.
- Hardware: 8 NVIDIA P100 GPUs.

- Our implementation is in PyTorch.
- We use Multi30k based on WMT 2016 dataset.
  - Train 29000 sentence pairs
  - Valid 1040 sentence pairs
  - Test 1000 sentence pairs
- We do a random seeding of 1337**
- For English to German, we train our model for 6 hours over 1000 epochs.
- Hardware: 4 NVIDIA A100-SXM-80GB GPUs.
  - 4 cores/gpu
  - 8 gb/core

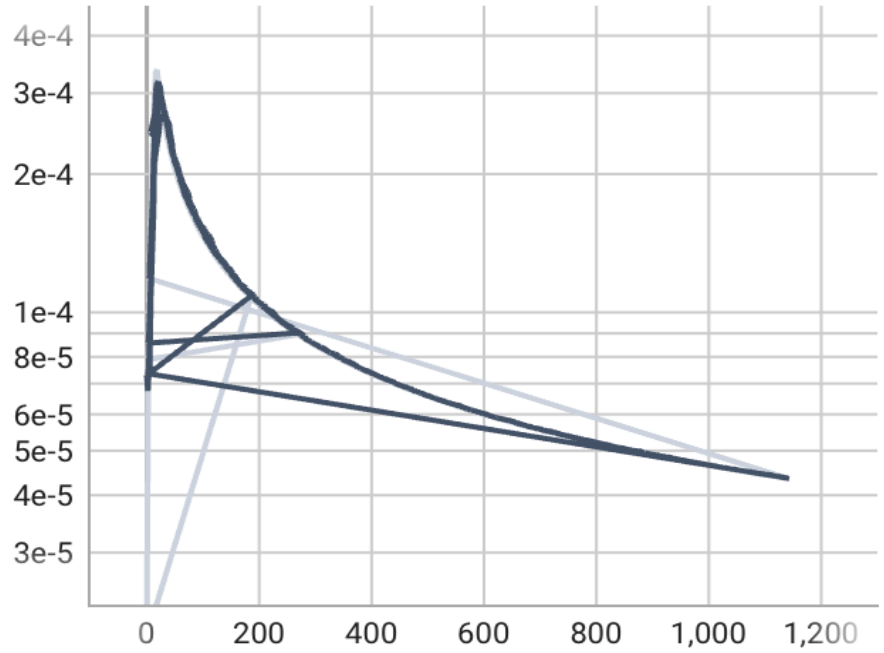
**The Illustrated Transformer

Result

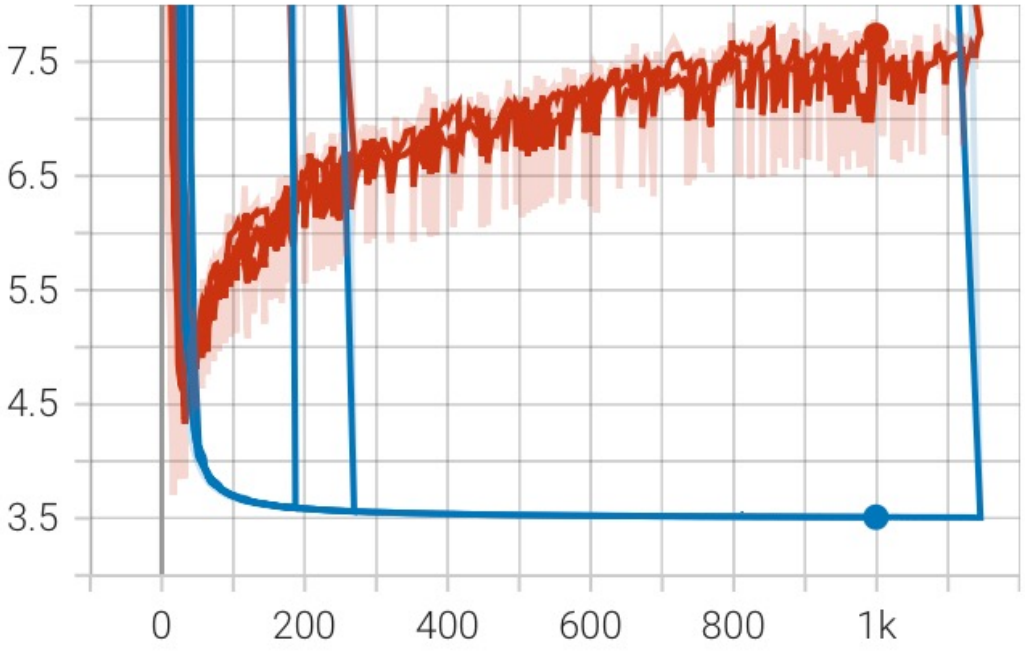
- We trained for 100, 500, 1000, 1500, and 2000 epoch
- The best performance was on 1000 epochs.
- On 1500 epoch the kernel crashed with memory issue.
- On 2000 epoch we were not able to run the complete epoch due to memory issue.

Task	Original Paper	Our Work
English to German	28.4	29.4
English to French	41.8	NA

Result Analysis



Learning Rate



Loss

Limitation

- The baseline transformer uses $O(n^2)$ in memory computation.
  - Increasing size of length increases computation quadratically.
- Fixed length unlike RNN of arbitrary length.
- Inability to process input sequentially.
  - Not like how human brain works.
- Paper addressing limitation
  - Limited access to long memory.
  - Limited ability to update state.

Demo

Code Repository and Future Task

- The complete working code with instructions on how to run and details is available at https://github.com/aniket414/vaswani-et-al-2017
- If you like our work and want to contribute to the project, we are working on English to Hindi translation.
  - Dataset: IIT Bombay English-Hindi Corpus
  - Tokenizer: Indic language NLP library

Acknowledgement

- The byte pair encoding parts are borrowed from subword-nmt.
- Andrej Karpathy youtube video “Let's build GPT: from scratch, in code, spelled out”.
- Aladdin Persson youtube video “Pytorch Transformers”.
- The Illustrated Transformer by Jay Alammar.
- Pytorch tutorial “LANGUAGE TRANSLATION WITH NN.TRANSFORMER AND TORCHTEXT”.

Thank You!

Why doesn’t the Transformer converge?

- It is a known issue with Deep Transformers that they don’t converge well.
- Transformer employs residual connection and layer normalization to ease the optimization difficulties caused by its multi-layer encoder/decoder structure. While several previous works show that even with residual connection and layer normalization, deep Transformers still have difficulty in training, and particularly a Transformer model with more than 12 encoder/decoder layers fails to converge.
- The original paper** and official implementation# make a small change in implementation paper however they still aren’t able to converge significantly.

**Vaswani et al., 2017
#Vaswani et al., 2018

How to converge?

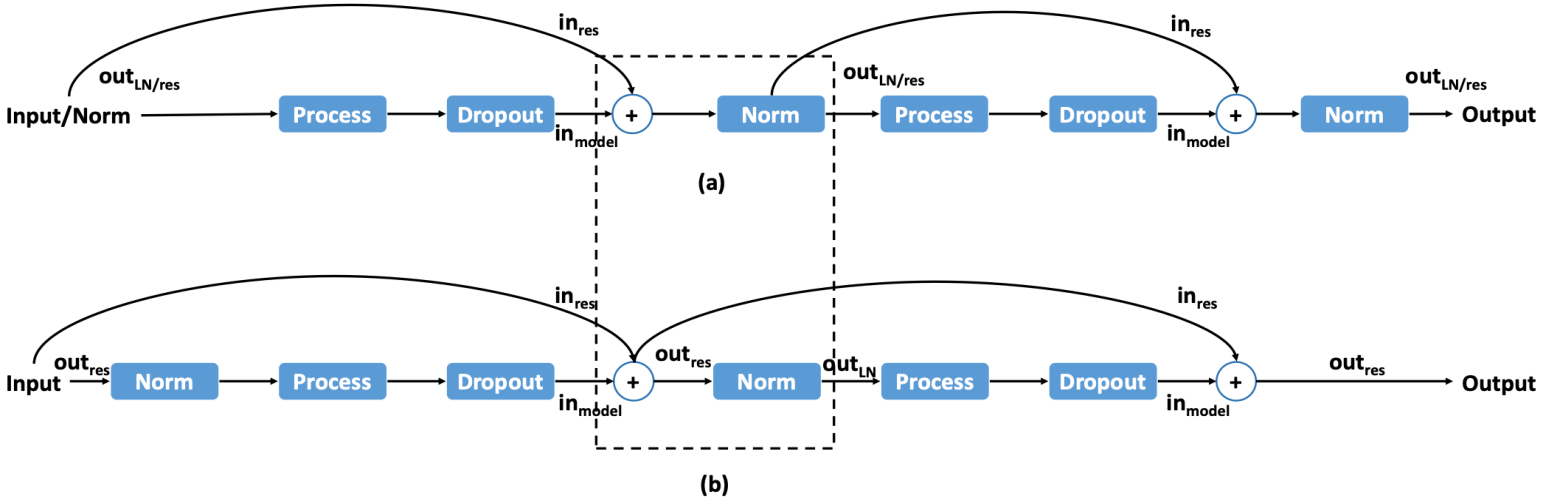


Figure 1: Two Computation Sequences of Transformer Translation Models: (a) the one used in the original paper, (b) the official implementation. We suggest to regard the output of layer normalization ($out_{LN/res}$) as the output of residual connection rather than the addition of in_{res} and in_{model} for (a), because it ($out_{LN/res}$) is the input (in_{res}) of the next residual connection computation.

Possible convergence: Lipschitz Constrained Parameter Initialization.

Source: [Hongfei Xu et al., 2020]