Transformers for NLP

Katarzyna Baraniak

February 21, 2024

Transfer Learning for NLP

Encoder decoder architecture
- Encoder
- Attention mechanism
- Feed-forward Layer
- Normalisation Layer and Skip Conections
- Positioal encodings
- Decoder

Some most important Transformer architectures

Pre-training and Fine-tuning
- Pretraining encoder-decoder
- Pretraining decoder
- Pretraining encoder
- Fine-tuining

Table of Contents

Transfer Learning for NLP

Encoder decoder architecture
- Encoder
- Attention mechanism
- Feed-forward Layer
- Normalisation Layer and Skip Conections
- Positioal encodings
- Decoder

Some most important Transformer architectures

Pre-training and Fine-tuning
- Pretraining encoder-decoder
- Pretraining decoder
- Pretraining encoder
- Fine-tuining

What is Transfer Learning?

Transfer learning is the method where the knowledge is learned in one domain or task and is applied to the other.
The model that is trained in the original task is called **pre-trained model**.
The process of adaptation to the new task is called **fine-tuning**.

Transformer and Large Language Models

Transformer is a neural network architecture that is most commonly used for language modeling.
It introduces the *attention mechanism* and *positional encodings* that helps to understand the relations between words over a long distance.
There are also transformers for other domains like computer vision or reinforcement learning.
Large Language Models are neural networks usually based on transformers that is able to generate human-like text or solve nlp related tasks.

What transformers can do?

They can solve many classic NLP problems and more

- text generation
- question answering
- summarization
- zero shot classification
- mask filling
- sentiment analysis
- named entity recognition
- ...

Timeline of Transformers

2017

- Transformer

2018

- ULMFiT
  Elmo
- GPT
- BERT

2019

- GPT-2
  RoBERTa
- DistilBERT
- XLM-R

2020

- GPT 3
- DeBERTa
- T5

2021

- GPT-Neo
  GPT-J

2022

- Instruct-GPT
  Bloom
- PaLM

2023

- Llama
  GPT-4
- Llama2

2024

- OLMo

Types of Transformers' architectures [TVWW22][JM23]

3 types of transformers, differs not only by architecture but also usage and the training process

- encoder-decoder(the original idea)
- encoder-only
- decoder-only

Types of Transformers' architectures [TVWW22][JM23]

- encoder-decoder
  - creates complex mapping from one sequence to another
  - combines architecture of encoder and decoder
  - machine translation, summarization task *
  - T5, BART

Types of Transformers' architectures [TVWW22][JM23]

- encoder-decoder
  - creates complex mapping from one sequence to another
  - combines architecture of encoder and decoder
  - machine translation, summarization task *
  - T5, BART
- encoder-only
  - converts textual input in numerical representation
  - uses bidirectional attention to create token representation depending on both left and right context
  - text classification, NER *
  - BERT, RoBERTa, DistilBERT

Types of Transformers' architectures [TVWW22][JM23]

- encoder-decoder
  - creates complex mapping from one sequence to another
  - combines architecture of encoder and decoder
  - machine translation, summarization task *
  - T5, BART
- encoder-only
  - converts textual input in numerical representation
  - uses bidirectional attention to create token representation depending on both left and right context
  - text classification, NER *
  - BERT, RoBERTa, DistilBERT
- decoder-only
  - autocomplete the input text by predicting the next word
  - autoregressive attention (casual attention)- token representation depends only on the left context
  - all tasks that requires text generation: question answearing, chatbots *
  - all GPT models

Table of Contents

Transfer Learning for NLP

Encoder decoder architecture
- Encoder
- Attention mechanism
- Feed-forward Layer
- Normalisation Layer and Skip Conections
- Positioal encodings
- Decoder

Some most important Transformer architectures

Pre-training and Fine-tuning
- Pretraining encoder-decoder
- Pretraining decoder
- Pretraining encoder
- Fine-tuining

Encoder-decoder architecture 2017 [VSP+17]

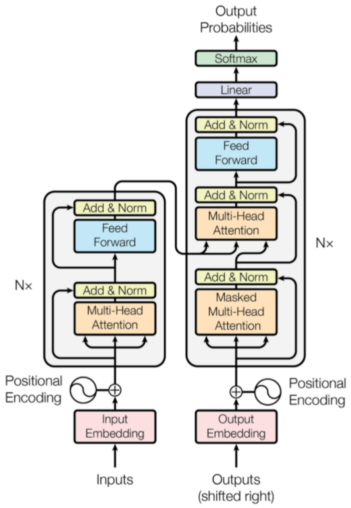


Figure 1: The Transformer - model architecture.

Figure: Transformer model architecture. Source of this figure : [VSP+17]

Encoder-decoder architecture

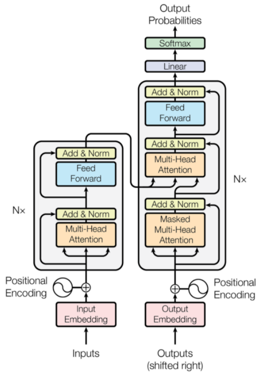


Figure 1: The Transformer - model architecture.

- Encoder (left part)- Calculates hidden state (context) - the sequence of embedding vectors based on the input sequence of tokens.
- Decoder (right part) - Generates the output sequence of tokens based on the encoder's hidden state.

Encoder-decoder architecture

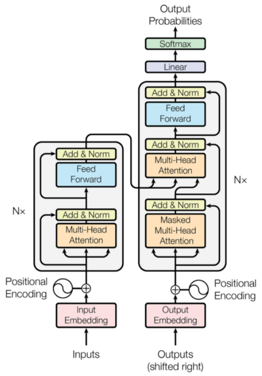


Figure 1: The Transformer - model architecture.

- input text is tokenized and converted into **embeddings**. **Positional encodings** are added to store information about the position of each token.
- encoder and decoder are built from **several layers** or "blocks"
- encoder's output is passed to **each** decoder layer. Decoder calculates the probability for the next token. The output is passed as **additional input** do decoder to predict the next token (shifted right)

Encoder

The main role of encoder is to create embedding that encode each token preserving contextual information from the whole sequence. Sublayers of encoder are:

- multi-head self-attention
- fully connected feed forward layer

Additionally, each sublayer has skip connection and layer normalization.

Attention Mechanism

The genaral idea of attention mechanism is to provide information about the most important parts of a sequence for a current task. There are many variants of attention.

- additive attention
- self attention
- scale dot product attention
- multi-head attention

Self-attention [RLMD22]

Applied within a single sentence. Tokens in an input sequence of length T are represented by embeddings $X = (x_1, \ldots, x_T)$. Three steps to calculate self- attention:

1. **attention score** ω_{ij} is a dot product:

$$\omega_{ij} = x_i^t x_j \tag{1}$$

   ω_{ij} looks for the relation between words in same sequence

2. **attention weights** α_{ij} computed by normalizing scores using softmax:

$$\alpha_{ij} = softmax(\omega_{ij}) = \frac{exp(\omega_{ij})}{\sum_{k=1}^{T_x} exp(\omega_{ik})}, \tag{2}$$

   .

3. The context vector c_i for i^{th} word is:

$$c_i = \sum_{j=1}^{T} \alpha_{ij} x_j, \tag{3}$$

This type of attention is also called dot-product attention [LPM15].

Self-attention

Example

Given word *"flies"* we can assume it refers to the insects but when it appears in a sentence *"Time flies like an arrow"* you now that "flies" is a verb.
Attention mechanism set more attention weight α_{ij} to more related words:

"Time **flies** like an arrow"

Scale dot product attention

The mechanism of this attention works as a function that maps vectors of a query and set of key-value pairs to an output [BCB14]. Comparing notation to self-attention:

- keys k_i similar functionality to xi from attention scores equation $\omega_{ij} = x_i^t x_j$
- queries q_i similar to x_j from attention scores equation $\omega_{ij} = x_i^t x_j$
- values v_i similar to x_j from context vector that is multiplied by attention weights $c_i = \sum_{j=1}^{T} \alpha_{ij} x_j$

Scale dot product attention - meaning of query, key, value

The exact meaning of query, values and keys depends on the performed task.

Example

For machine translation it may be vectors of input words for queries, vectors of possible output words for keys and best matched output words for values.

Example

For language models the key, values and queries are equal and may be interpreted as vectors of words in a sentence.

Scale dot product attention - steps

Consists of four steps:

1. project token embeddings into three matrices *queries* Q , *keys* K, *values* V
2. find attention scores - how much queries and keys are related or similar to each other, in that case dot product: QK^T,
3. find attention weights - so the attention scores are normalized by scaling factor and softmax: $softmax(\frac{QK^T}{\sqrt{d_k}})$
4. find context vectors - multiply attention weights by Value matrix

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \tag{4}$$

Scale Dot-Product Attention

It is similar to the dot product attention except of scaling queries and keys by the factor $\frac{1}{\sqrt{d_k}}$ and d_k is the dimensionality of input queries and keys.

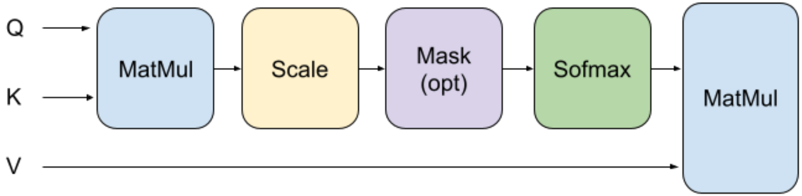


Figure: Scaled Dot-Product Attention

Scaled Dot-Product attention is a part of Multi-head Attention that is used in Transformer Models.

Multi-head attention

Consist of multiple set of linear projections called **attention heads**. Each attention head focus on different aspects of similarity for example: one head can focus on nearby adjectives nd another on subject-verb interaction.
Each attention head uses three different independent linear transformations to generate the query, key and value vectors to learn parameters for different semantic aspects of the sequence.

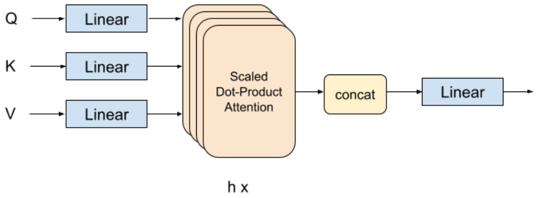


Figure: Multi-head attention

Multi-head attention

The model linearly projects queries, keys and values h times with different linear projections. Then the attention function is applied on each of these projections parallel to calculate heads of attention model:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V), \tag{5}$$

where the projections are matrices of parameters W_i
Then, heads are concatenated and projected again to calculate Multi-Head Attention:

$$MultiHead(Q, K, V) = Concat(head_i, \ldots, head_h)W^O, \tag{6}$$

where W^O is also a parameter matrix.

Feed-forward layer

- similar for encoder and decoder
- two-layer fully connected neural network
- often referred as **position-wise feed forward layer**: process each embedding independently, instead of an embedding's sequence

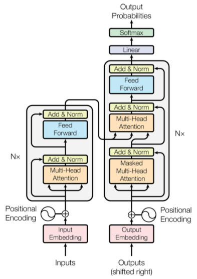


Figure 1: The Transformer - model architecture.

Normalization Layer and Skip Connection

Normalisation layer is used to normalize its input to have zero mean a unit variance.
Skip connection or *residual connection* is a connection that pass it's input tensor without processing. It can be used to skip layers that process the input, then add unprocessed and processed tensors together.

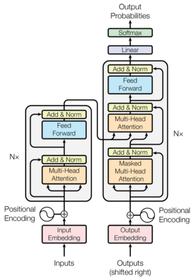


Figure 1: The Transformer - model architecture.

Positional encodings

It is the way transformer model the position of each token in the input sequence.
Position dependent pattern is added to the token embeddings.
This pattern is characteristic for each position so all the layers in each stack can use this information during the transformations.
Possible patterns:

- learnable patterns - embedding of position is learned based on position index
- absolute positional representation- static patterns base on cosine and sine signals to encode the position
- relative positional representation - encode relative position between tokens by modifying attention mechanism

Decoder

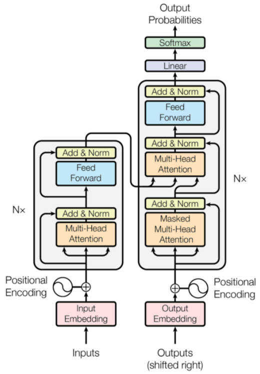


Figure 1: The Transformer - model architecture.

Generates the output tokens based on the input context. Main sublayers of decoders are

- Masked multi-head self-attention layer- it's aim is to ensure that the model do not have the information about the target token, it can only make predictions based on the past output and the current token being predicted
- Multi-head (Encoder-decoder) attention layer- it uses the encoder's output key and value vector with queries from decoder intermediate representations. It learns relations between tokens from two different sequences.

Table of Contents

Transfer Learning for NLP

Encoder decoder architecture
- Encoder
- Attention mechanism
- Feed-forward Layer
- Normalisation Layer and Skip Conections
- Positioal encodings
- Decoder

Some most important Transformer architectures

Pre-training and Fine-tuning
- Pretraining encoder-decoder
- Pretraining decoder
- Pretraining encoder
- Fine-tuining

Some most important Transformer architectures

Encoder	Encoder – Decoder	Decoder
BERT	T5	GPT
DistilBERT	BART	GPT2
RoBERTa	BIGBIRD	GPT3
XLM	Marian	CTRL
ALBERT		Llama
ELECTRA		Llama2

BERT

Encoder-only model that is pre-trained using two tasks:

- *masked language modeling*
- *next sentence prediction*

(more details in a few slides)

DistilBERT

Created using *knowledge distillation*.
Achieves 97% of BERT performance while using only 60% of memory size and being 60% faster.

RoBERTa

Improves the performance of BERT.
Trained longer than BERT, larger batches and with more training data. Next sentence prediction is not used here.

XLM

Multilingual model trained using *translation language modeling* that extends mask language modeling for multilingual input.

GPT

First decoder-only transformer model. Pretrained by predicting next word based on previous ones. Trained on BookCorpus.

GPT-2

Upscaled GPT model.
First only the smaller model was released due to concerns about possible misuse. Later the full model was released.

GPT3

Larger than previous GPT models - about 175 bilion parameters. It was not released as open source, only interface is available through OpenAI API.

CTRL

Conditional Transformer Languages add "control tokens" to control the style of generated text.

T5

Encoder-Decoder based on original transformer that treats all NLU and NLG tasks as text-to-text problem. For example when performing classification the model has to generate label as text not a class as usual.
Trained on C4 dataset using masked language modeling and superGLUE tasks by converting them to text-to-text tasks.

BART

Encoder decoder architecture that use pretraining methods from both BERT and GPT.
Transformation of encoder part input like sentence permutation, token deletion and document rotation is used so the decoder has to reconstruct the original texts.

Table of Contents

Transfer Learning for NLP

Encoder decoder architecture
- Encoder
- Attention mechanism
- Feed-forward Layer
- Normalisation Layer and Skip Conections
- Positioal encodings
- Decoder

Some most important Transformer architectures

Pre-training and Fine-tuning
- Pretraining encoder-decoder
- Pretraining decoder
- Pretraining encoder
- Fine-tuining

Pre-training sequence-to-sequence models - encoder decoder

Supervised training:
the dataset with pair of input and output that can represent labels is needed.
Example 1:
input is a text in one language and the output is a text in another language
Example 2:
input is a code and the output is code documentation.
With large, diverse and clean dataset the model can be trained to solve efficiently tasks that takes as input the whole sequence and outputs another one related to input.

Pre-training casual language model - decoders

The goal is the same as in RNN language model or any *autoregressive* model. Find the most probable sequence of tokens $y_1, y_2, ..., y_t$ from all possibilities, given initial text $x_1, x_2, ..., x_3$

$$P(y_1, ...y_t) = \prod_{t=1...N} p(y_t|y_{1:t-1}, x) \tag{7}$$

In case of language modeling output layer calculates the logit z_{ti} of each token and the probability distribution over vocabulary. $P(y_t = w_i|x, y_{1:t-1}) = softmax(z_{t,i})$

Pre-training casual language model - decoding the next word

Choosing the next word is not straightforward. There are several approaches:

- **Greedy Search Decoding** - take the word with the highest probability at each time step
- **Beam Search Decoding** - remember top-b most probable next tokens where b is the number of beams. Choose the next set of beam from most probable next token sets from the b most likely sequences. Repeated until maximum length or an EOS token is reached. The most likely sequence is selected by ranking the b beams according to their log probabilities.
- **sampling methods** - adds diversity to the output: random sampling, top-k sampling, top-p sampling, temperature sampling

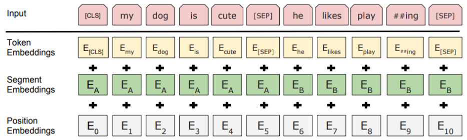
Pre-training Bidirectional Language Model - encoders

This Bidirectional Encoder Representations from Transformers (BERT) [DCLT19] aim is to create contextual word embeddings from a textual data.
It is based on encoder architecture.
The difference between left-to-right language models(like GPT), and BERT is that the latter fuse the left and the right context, which stands for bidirectional Transformer.
Pre-trained using unlabelled text data and different pre-training tasks: **masked language model** and **next sentence prediction**.

Pre-training Bidirectional Language Model

Before pre-training

- segment embedding - specific input/output representation for one or two sentences in one input sequence
- sentence is split into tokens based on a token vocabulary using WordPiece embedding model [WSC$^+$16]
- two special tokens: [CLS] and [SEP].
- final input representation of a token is a sum of position embedding, segment embedding and token embedding.



Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Figure: Source: [DCLT19]

Mask language modeling

In this task, Some percentage of input tokens is replaced by [MASK] token at random and predicting only those masked tokens. When pretraining BERT, 15% of input tokens in each sentence are selected to be masked.
If the token is selected, then only 80% of time it is replaced by [MASK], 10% of time it is replaced by another random token and 10% of time it is not changed. Then the hidden vector of the replacing input token is used to predict the original token.

Example

Input to the model: I *coffee* in [MASK].
Output to be predicted: I live in Warsaw.

Next sentence prediction

Many NLP tasks are based on the relationship between two sentences like question answering or natural language inference.
It is a binary classification task.
Two consecutive sentences were chosen from the monolingual corpus. During training 50% of time the second sentence was the actual successor of the first one with label *isNext* and 50% was a random sentence with label *NotNext*.
Token [CLS] represents both sentences for next sentence prediction.

Fine-tuning

The pretrained model is fine-tuned for a specific task.

Head of a transformer

The head is a dedicated output layer added for a specific task fine-tuning even if the pre-trained model used the same task.

The proper input and output labels are fed into model and end-to-end fine-tuning is done by *adjusting pretrained parameters* Fine-tuninig step is usually inexpensive in terms of time and space complexity comparing to pre-training.

Fine-tuning tasks and model's heads

The head examples:

- **language modeling head** - a linear layer that returns the probability distribution over next words
- **sequence tagging head** -feed forward layer that generates probabilities over set of labels for each input token
- **sequence classification head** - a linear layer that takes as input the embedding of special token [CLS] used to represent the whole sequence
- **question answering head** - a linear layer on top of the hidden-states output to compute span start and span end logits to determine position of answer in a sequence)

Summary

The following topics were discussed:

- Transfer Learning
- Original transformers architecture including:
    - encoder block
    - decoder block
    - attention mechanisms
    - residual connections, layer normalization,
    - positional encoding
- examples of transformers
- pre-training and fine-tuning the model

References I

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio, *Neural machine translation by jointly learning to align and translate*, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *BERT: Pre-training of deep bidirectional transformers for language understanding*, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (Minneapolis, Minnesota), Association for Computational Linguistics, June 2019, pp. 4171–4186.

Daniel Jurafsky and James Martin, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*, vol. 3, 2023.

Thang Luong, Hieu Pham, and Christopher D. Manning, *Effective approaches to attention-based neural machine translation*, Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (2015).

Sebastian Raschka, Yuxi Hayden Liu, Vahid Mirjalili, and Dmytro Dzhulgakov, *Machine learning with pytorch and scikit-learn: Develop machine learning and deep learning models with python*, Packt Publishing Ltd, 2022.

Lewis Tunstall, Leandro Von Werra, and Thomas Wolf, *Natural language processing with transformers*, " O'Reilly Media, Inc.", 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, Advances in neural information processing systems, 2017, pp. 5998–6008.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al., *Google's neural machine translation system: Bridging the gap between human and machine translation*, arXiv preprint arXiv:1609.08144 (2016).