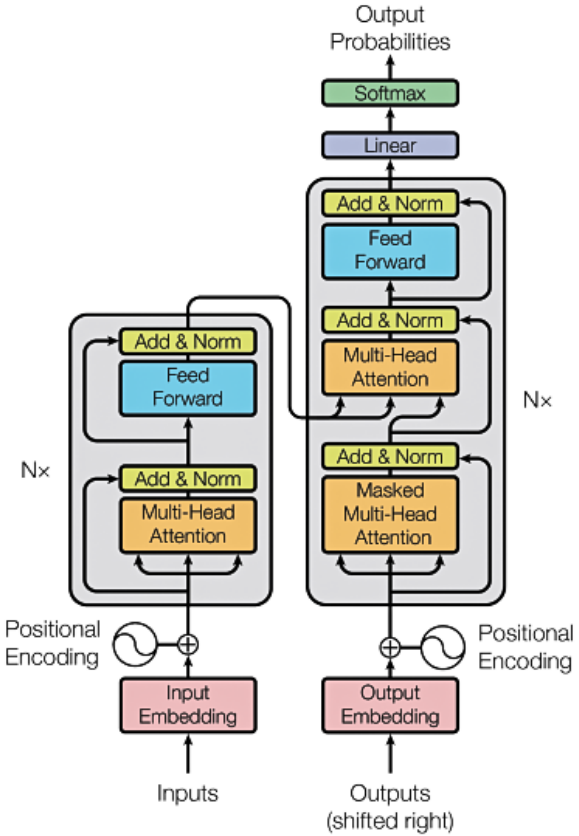
UNIVERSITY OF CENTRAL MISSOURI

Department of Computer Science & Cybersecurity

CHPATER 10. Transformer Model

CS 5720: Neural Network & Deep Learning

Dr. I Hua Tsai, Assistant Professor



Transformers

Transformers vs. LSTM

Like the LSTMs, transformers can handle distant information

But unlike LSTMs, transformers are not based on recurrent connections

- Which means that transformers can be more efficient to implement at scale

Transformers are made up of stacks of transformer blocks, each of which is a multilayer network made by combining:

- simple linear layers
- feedforward networks
- **self-attention layers**

Self-attention allows a network to directly extract and use information from arbitrarily large contexts without the need to pass it through intermediate recurrent connections as in RNNs

Background (1)

The **RNN** and **LSTM** neural models were designed to process language and perform tasks like classification, summarization, translation, and sentiment detection

RNN: Recurrent Neural Network

LSTM: Long Short Term Memory

In both models, layers get the next input word and have access to some previous words, allowing it to use the word's left context

They used word embeddings where each word was encoded as a vector of 100-300 real numbers representing its meaning

Background (2)

Transformers extend this to allow the network to process a word input knowing the words in both its left and right context

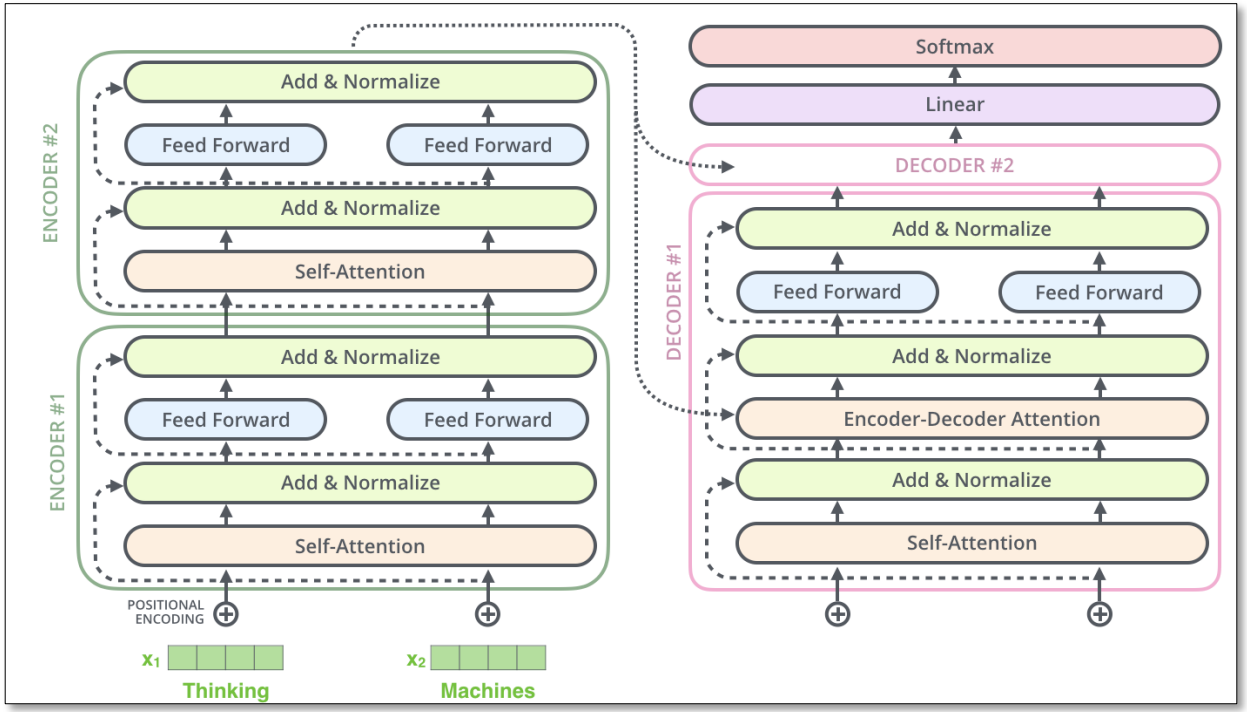
This provides a more powerful context model

Transformers add additional features, like **attention**, which identifies the important words in this context

And break the problem into two parts:

- An encoder (e.g., Bert)
- A decoder (e.g., GPT)

Transformer model



Encoder (e.g., BERT)

Decoder (e.g., GPT)

Attention

- Introduced to address issues with fixed-length decoder output
  - Limited information when input gets larger
- Encoder-decoder seq2seq structures

Transformer models are efficient at NLP tasks

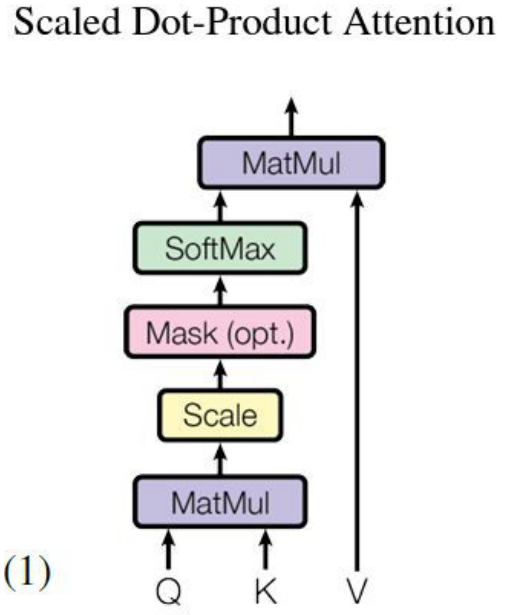
My car is a transformer

- Alignment Score: How well do the inputs align with the output positions
- Weight: Softmax output of alignment scores
- Context: Weighted sum of encoder hidden states and feed into decoder

Self Attention

- Capturing relationships between elements in the same sequence
- Scaled down by a factor to prevent exploding dot products and vanishing softmax gradients when the dimension gets large
- Mask introduced to prevent leftward information flow by setting illegal connections to negative infinity

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$



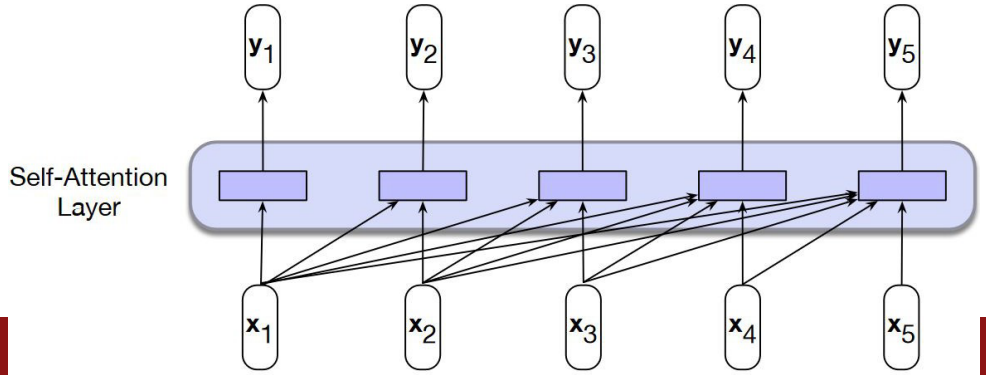
Self-Attention

When processing each item in the input:

- The model has access to all the inputs up to and including the one under consideration
- No access to information about inputs beyond the current one
- The computation performed for each item is independent of all the other computations
  - Easily parallelize both forward inference and training of such models

Attention-based approach: **Compare** an item to a collection of other items in a way that reveals their relevance in the current context

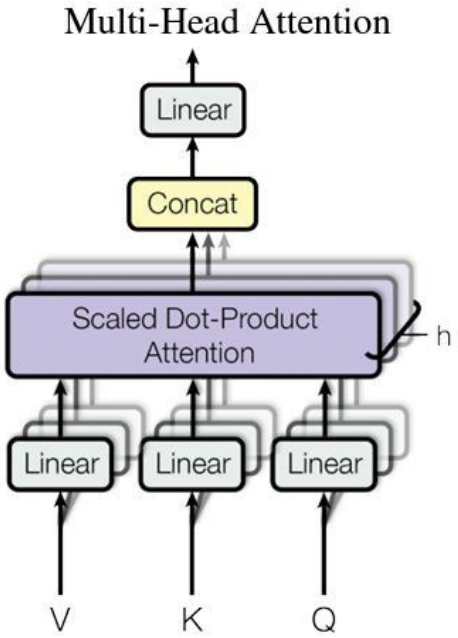
Self-Attention: The set of comparisons is made to other elements within a given sequence; the result of these comparisons is then used to compute an output for the current input



Multi-head Attention

- Multi-head Attention allows the model to jointly attend to information from different representation subspaces at different positions
- Use W to denote different projection matrices
- Logically splits Q, K, V into h number of spaces
- Add a dimension for heads, concat, and project again

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Transformers: Attention is All you Need!

Consider the three different roles that each input embedding plays during the course of the attention process as:

- Query: The current focus of attention when being compared to all the other preceding inputs
- Key: In its role as a preceding input being compared to the current focus of attention
- Value: As used to compute the output for the current focus of attention

To capture these three different roles, transformers introduce weight matrices W^Q, W^K, and W^V

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i; \;\; \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i; \;\; \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

- The inputs x and outputs y of transformers, as well as the intermediate vectors after the various layers, all have the same dimensionality 1×d
- Let's assume the dimensionalities of the transform matrices are $W^Q \in R^{d\times d}$, $W^K \in R^{d\times d}$, and $W^V \in R^{d\times d}$

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Architecture

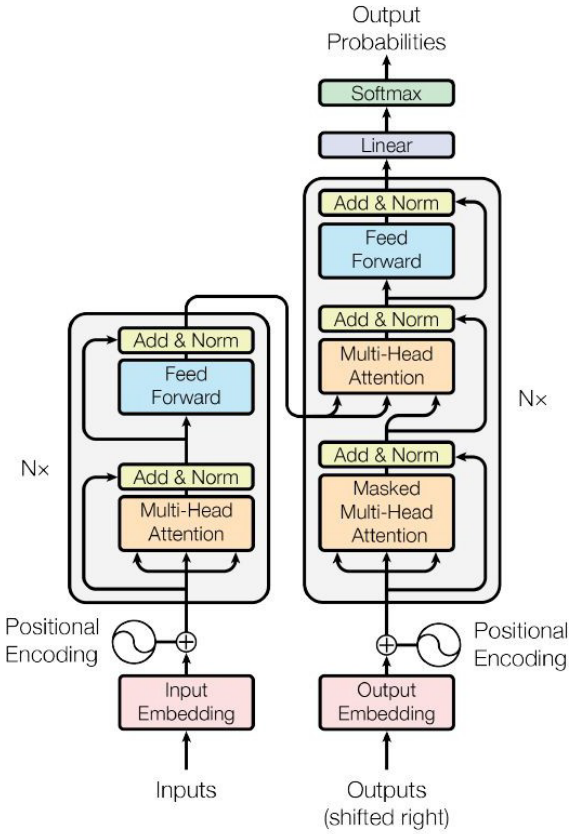
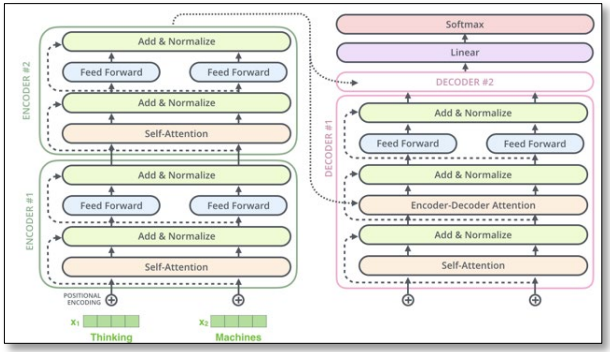


Figure 1: The Transformer - model architecture.

- Layers are stacked N times
  - Encoder Self Attention
  - Feed-Forward Layer
  - Decoder Self Attention
  - Encoder-Decoder Self Attention
  - Feed-Forward Layer
- Embedding convert input/output to model dimension
- Positional Encoding injected to provide positional information

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Transformers, GPT-2, and BERT



1. A transformer uses an **encoder stack** to model input, and uses **decoder stack** to model output (using input information from encoder side)
2. If we do not have input, we just want to model the “next word”, we can get rid of the encoder side of a transformer and output “next word” one by one. This gives us **GPT**
3. If we are only interested in training a language model for the input for some other tasks, then we do not need the decoder of the transformer, that gives us **BERT**

Training a Transformer

Transformers typically use semi-supervised learning with

- Unsupervised pretraining over a very large dataset of general text
- Followed by supervised **fine-tuning** over a focused data set of inputs and outputs for a particular task

Tasks for pretraining and fine-tuning commonly include:

- language modeling
- next-sentence prediction (aka completion)
- question answering
- reading comprehension
- sentiment analysis
- paraphrasing

Pretrained models

Since training a model requires huge datasets of text and significan computation, researchers often use common pretrained models

Examples (circa December 2021) include

- Google's BERT model
- Huggingface's various Transformer models
- OpenAI's and GPT-3 models

BERT
Bidirectional Encoder Representations from Transformers

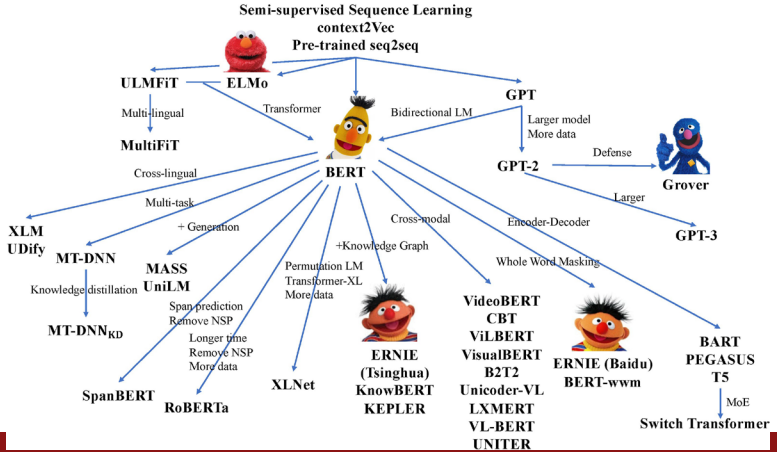
Background

Language model:

- Assign probability distribution over sequences of words that matches distribution of a language
- Predict next token given all previous tokens in the sequence.

Language Representation:

- Transforms discrete one-hot encoding(Bag-of-words) to low-dimensional dense vectors
- Feature Extraction



The evolution of language representations

FROM Pre-trained Word Embeddings TO Pre-trained Language Models

Feature-Based

Finetune-Based

Gen 1

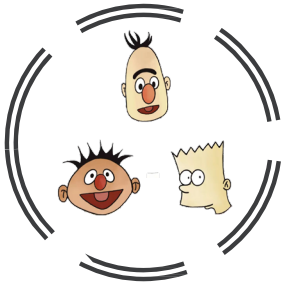
Gen 2

Gen 3

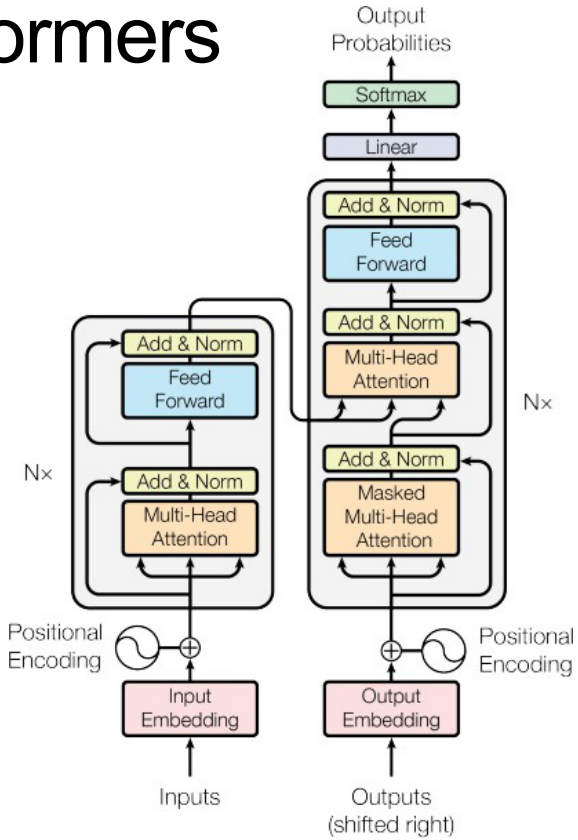
Static Embeddings
word2vec, gloVe, fasttext

Contextual Embeddings
CoVe, ElMo

Pre-trained Language Model
BERT, GPT, BART…



BERT: Bidirectional Encoder Representations from Transformers

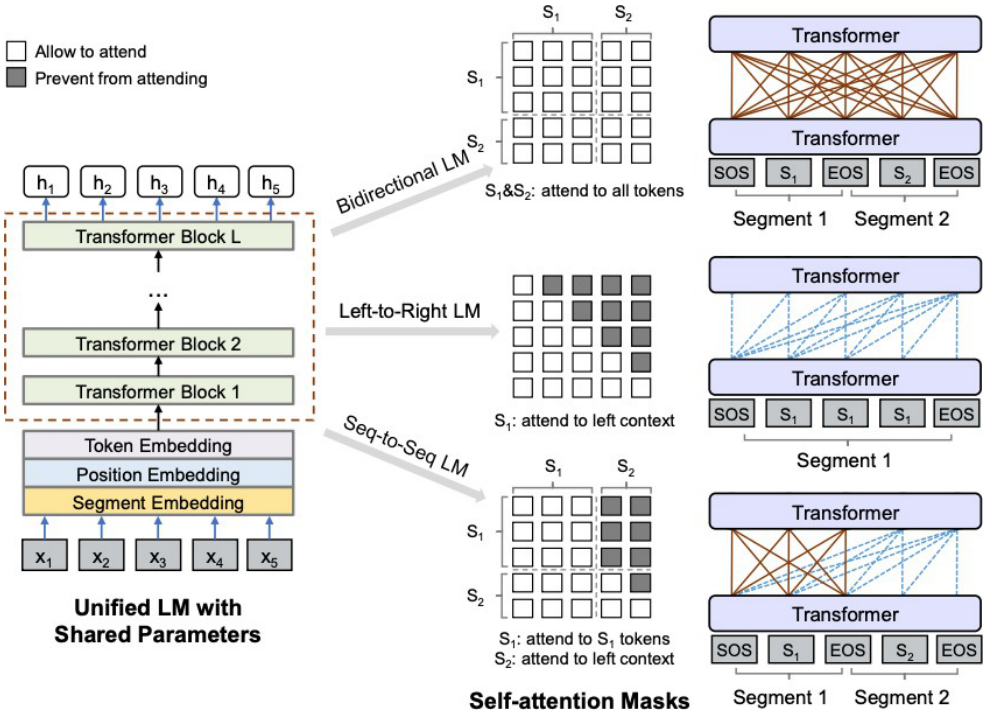


BERT ≈ Transformer Encoder+ MLM Head

GPT ≈ Transformer Decoder
– Encoder-Decoder Attention

Seq2Seq PLM(BART, T5 etc.) ≈ Transformer

BERT: Bidirectional Encoder Representations from Transformers



BERT

- Bidirectional, Good for NLU
- MLM, CLS as pretrained task
- Cannot do NLG

GPT

- Left-to-right , Unidirectional
- LM as pretrained task

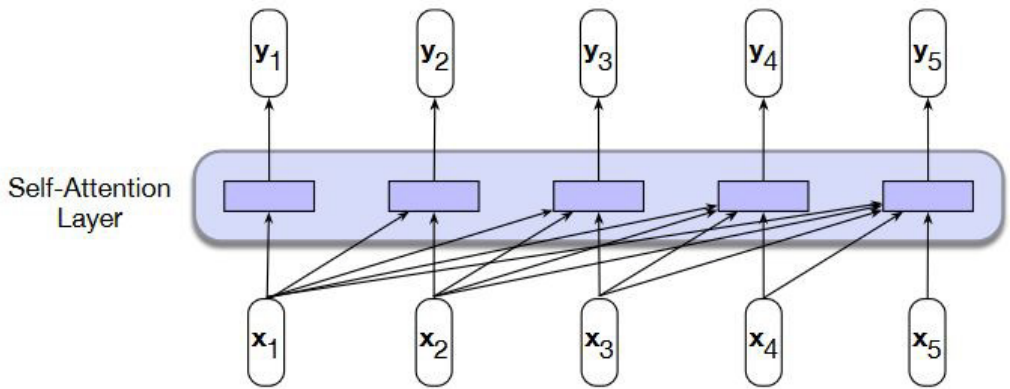
BART, T5

- Seq-to-Seq, can solve all tasks
- More diverse pretrained tasks

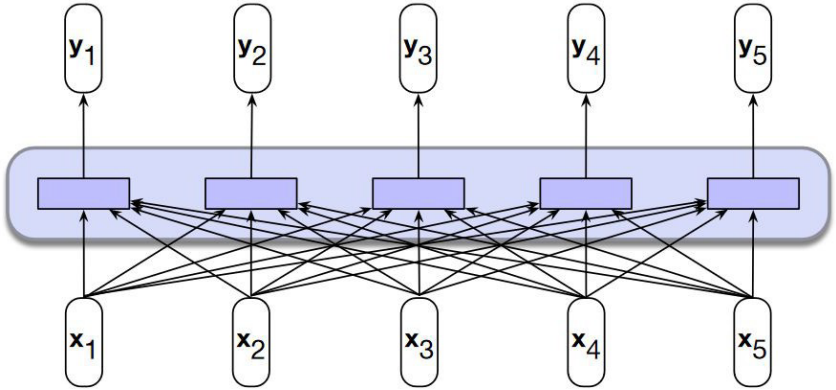
From: Unified Language Model Pre-training for Natural Language Understanding and Generation

Bidirectional Transformer Encoders

The focus of bidirectional encoders is on computing contextualized representations of the tokens in an input sequence that are generally useful across a range of downstream applications



backward looking, transformer model



bidirectional transformer model

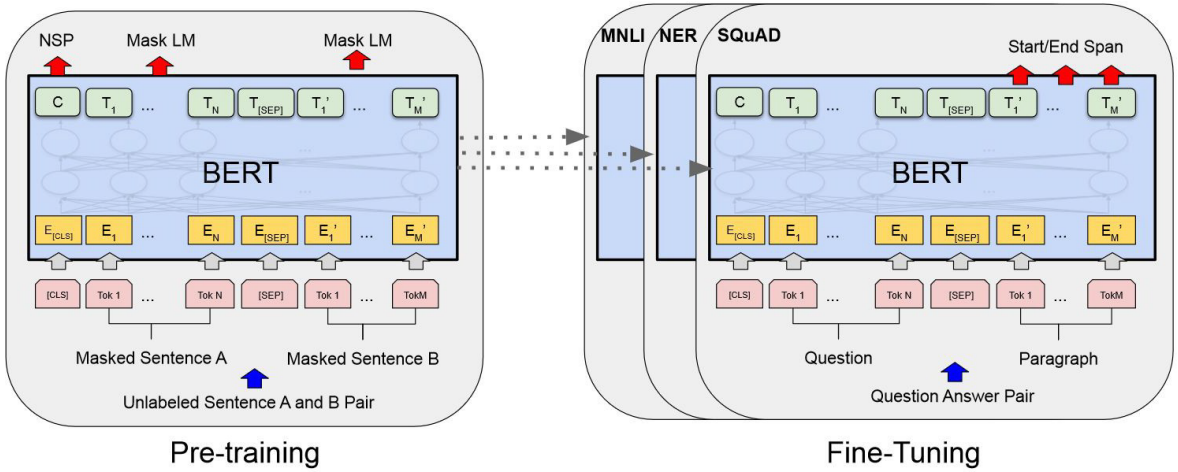
Overview

BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text

An encoder that reads the text input and a decoder that produces a prediction for the task. Transformer encoder reads the entire sequence of words at once

Since BERT's goal is to generate a language model, only the encoder mechanism is necessary

Pre-training and Fine-tuning



- The same architectures are used in both pre-training and fine-tuning
- `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token
- **Pre-training:** two tasks are considered
  1. **Masked LM:** mask some percentage of the input tokens at random, and then predict those masked tokens
     Mask 15% of all WordPiece tokens in each sequence at random
  2. **Next Sentence Prediction:** understanding the relationship between two sentence (50% of positive pairs)
     Used BooksCorpus (800M words) and Wikipedia (2,500M words)

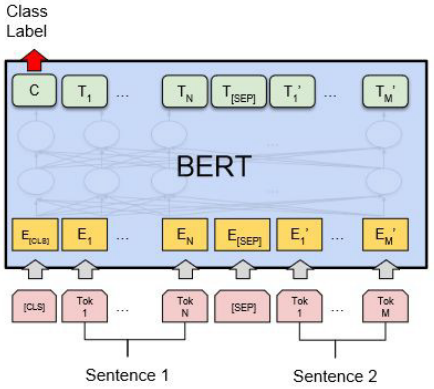
Fine-Tuning

Compared to pre-training, fine-tuning is relatively inexpensive

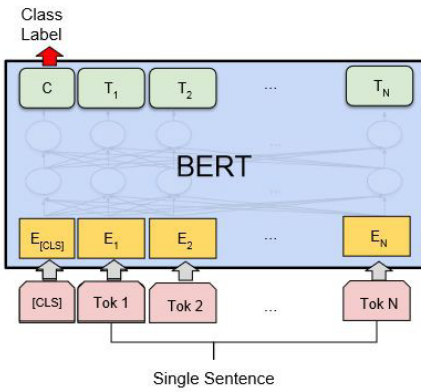
Most model hyperparameters are the same as in pre-training, except for parameters such as number of training epochs

Tasks:

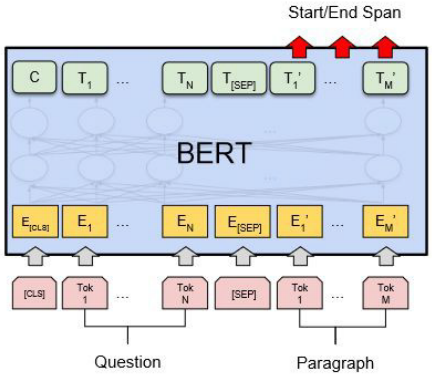
- Question answering
- Abstract summarization
- Sentence prediction
- Conversational response generation
- Sentiment classification



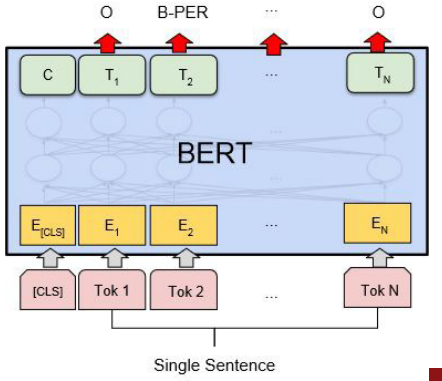
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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

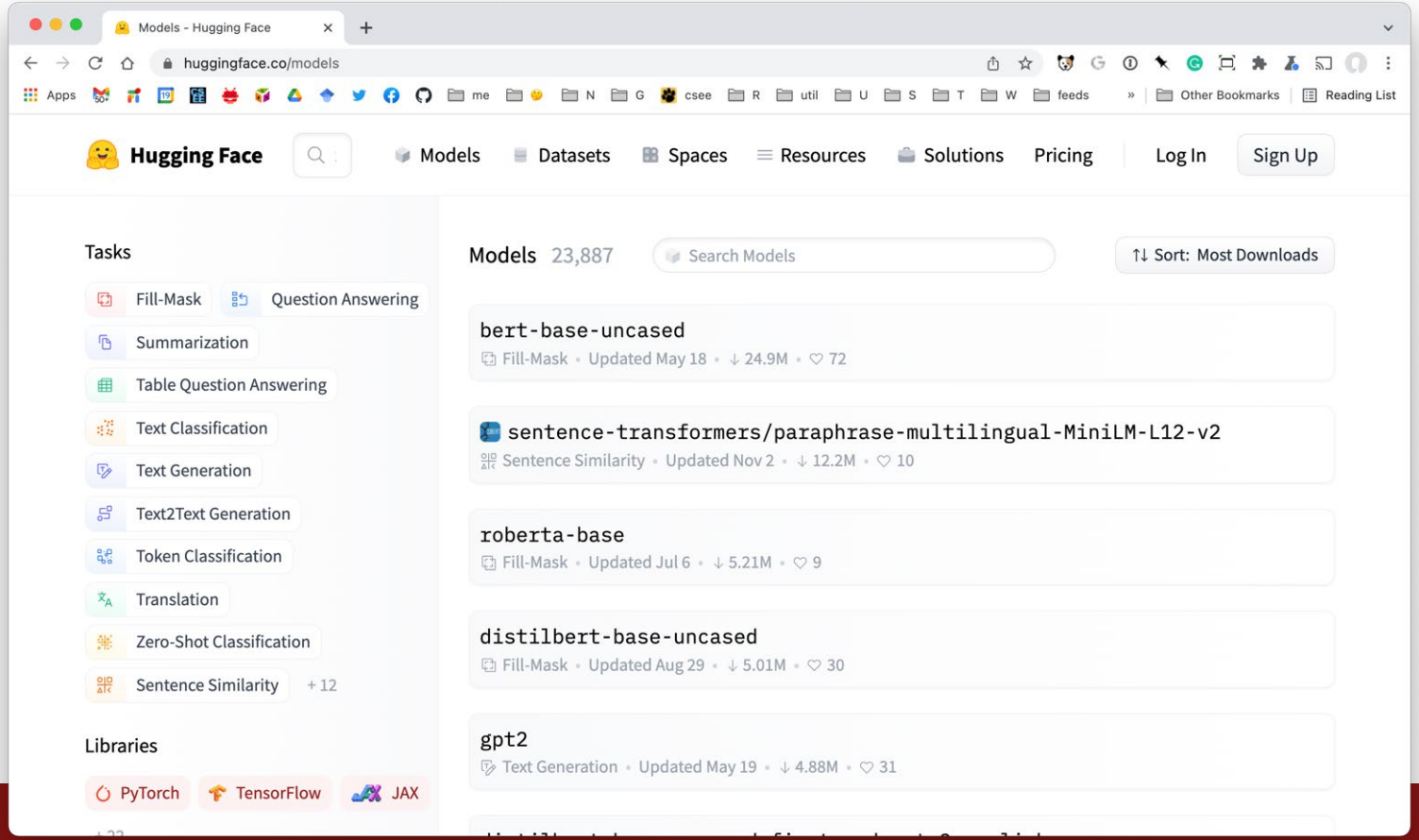


(c) Question Answering Tasks: SQuAD v1.1

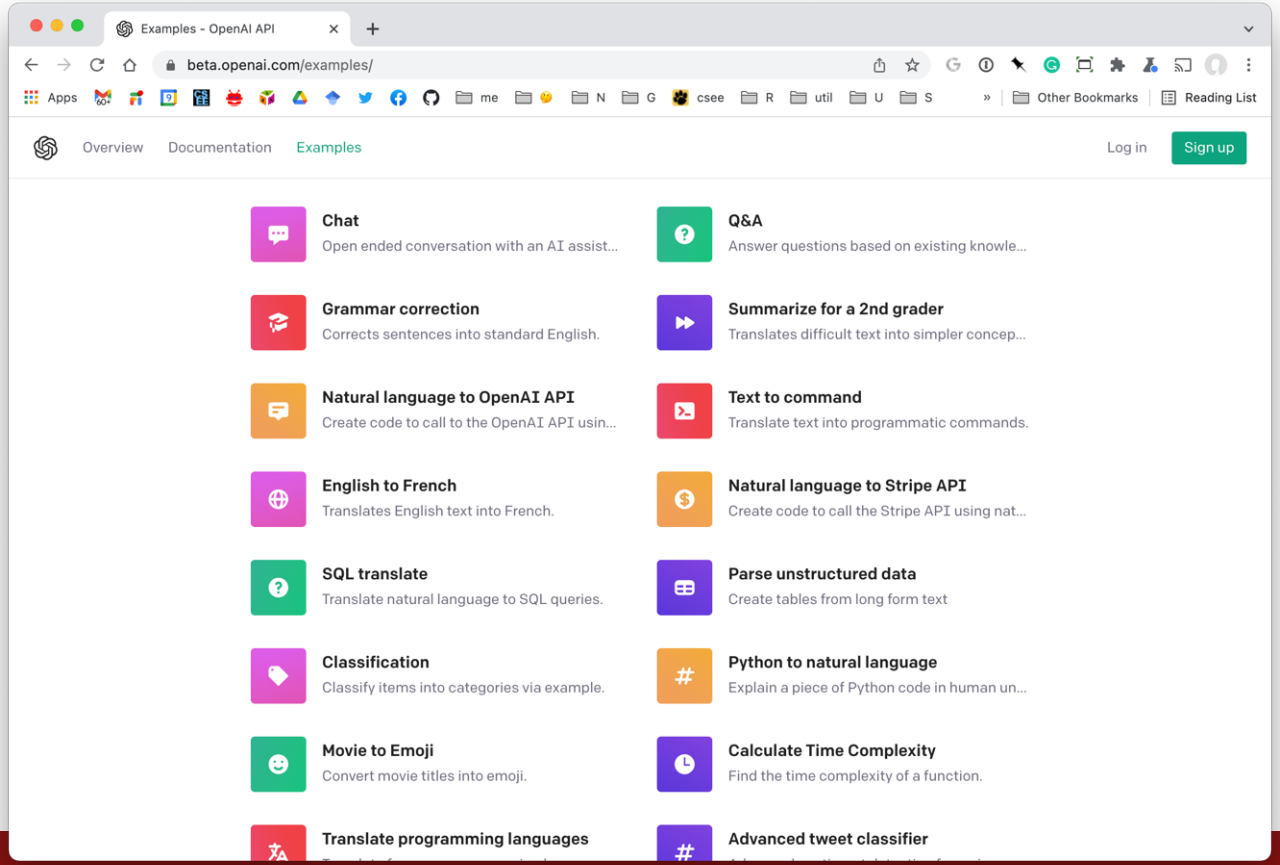


(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

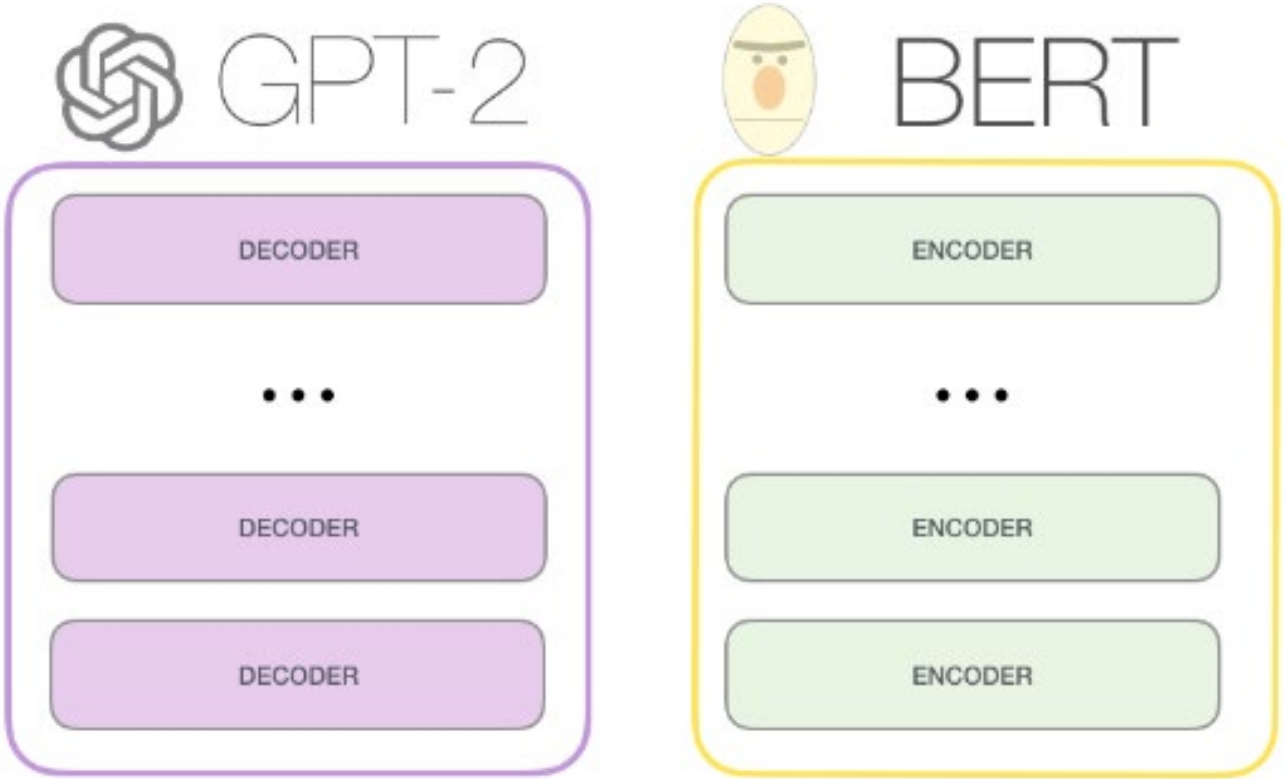
Hugggingface Models



OpenAI Application Examples



GPT-2, BERT


GPT-2
DECODER
DECODER
DECODER
BERT
ENCODER
ENCODER
ENCODER

Other Models Based on Transformers

Generative Pre-trained Transformer (GPT)

GPT (Generative Pre-trained Transformer) is a series of language generation models developed by OpenAI. These models are based on the Transformer architecture (2018)

GPT-2 (Generative Pre-trained Transformer 2) was the second model in the GPT series, released in 2019. It was trained on a large corpus of internet text and was designed for language generation tasks such as question answering, and text summarization

GPT-3 (Generative Pre-trained Transformer 3) Released in 2020, with over 175 billion parameters, and was trained on a much larger and diverse dataset, including web pages, books, and scientific articles

GPT-4 (Generative Pre-trained Transformer 4) Released in 2023, a multimodal model which can accept image and text inputs and produce text outputs

Zero-shot, One-shot and Few-shot, Contrasted with Traditional Fine-tuning

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.

Language Models are Few-Shot Learners

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:   ← task description
cheese =>                      ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

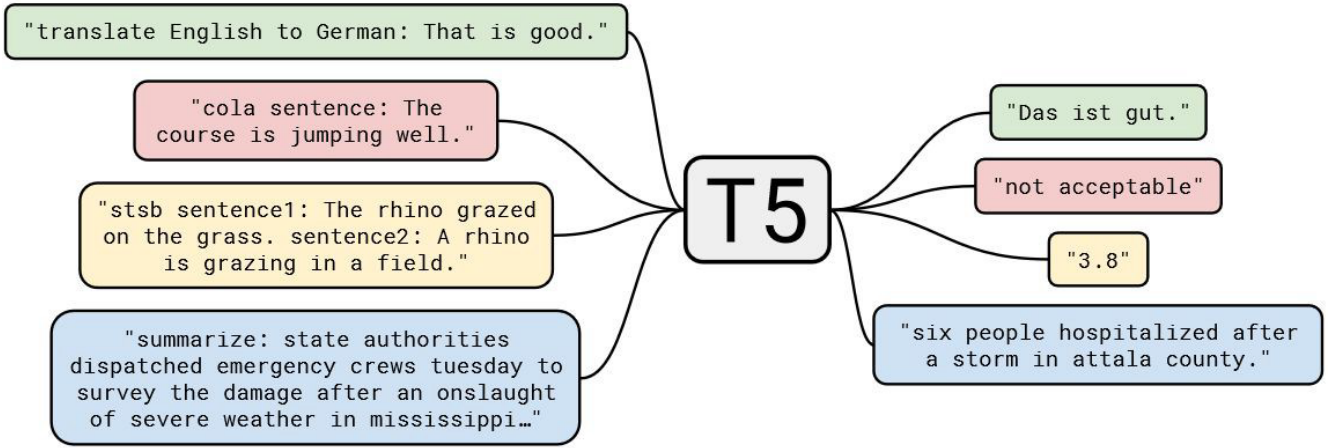
```
Translate English to French:   ← task description
sea otter => loutre de mer     ← example
cheese =>                      ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:   ← task description
sea otter => loutre de mer     ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                      ← prompt
```

Text-to-Text Transfer Transformer (T5)



A diagram of the text-to-text framework (T5)

Every task, including translation, question answering, and classification, is cast as feeding T5 model text as input and training it to generate some target text

Getting Started

Use pip for the installation, which is the package manager for Python. In notebooks, you can run system commands by preceding them with the ! character, so you can install the 🤗 Transformers library as follows:

```
!pip install transformers
```

You can make sure the package was correctly installed by importing it within your Python runtime:

```
import transformers
```

Pipelines

The pipelines are a great and easy way to use models for inference

These pipelines are objects that abstract most of the complex code from the library, offering a simple API dedicated to several tasks, including Named Entity Recognition, Masked Language Modeling, Sentiment Analysis, Feature Extraction and Question Answering

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
classifier("I've been waiting for a HuggingFace course my whole life.")
```

```
No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b
Using a pipeline without specifying a model name and revision in production is not recommended.
```


```
[{'label': 'POSITIVE', 'score': 0.9598049521446228}]
```

```
classifier(
    ["I've been waiting for a HuggingFace course my whole life.",
     "I hate this so much!"]
)
```

```
[{'label': 'POSITIVE', 'score': 0.9598049521446228},
 {'label': 'NEGATIVE', 'score': 0.9994558691978455}]
```

Example (2): Question Answering

```
from transformers import pipeline

question_answerer = pipeline("question-answering")
question_answerer(
    question="Where do I work?",
    context="My name is Sylvain and I work at Hugging Face in Brooklyn",
)
```

{'score': 0.6949767470359802, 'start': 33, 'end': 45, 'answer': 'Hugging Face'}

```
from transformers import pipeline

oracle = pipeline(model="deepset/roberta-base-squad2")
oracle(question="Where do I live?",
       context="My name is Wolfgang and I live in Berlin")
```

```
{'score': 0.9190717935562134, 'start': 34, 'end': 40, 'answer': 'Berlin'}
```

In-class Assignment / Quiz

More on Attention Is All You Need

Byte-Pair Encoding (BPE) Tokenization

Uses Huffman encoding for tokenization (greedy algorithm)

Training Steps:

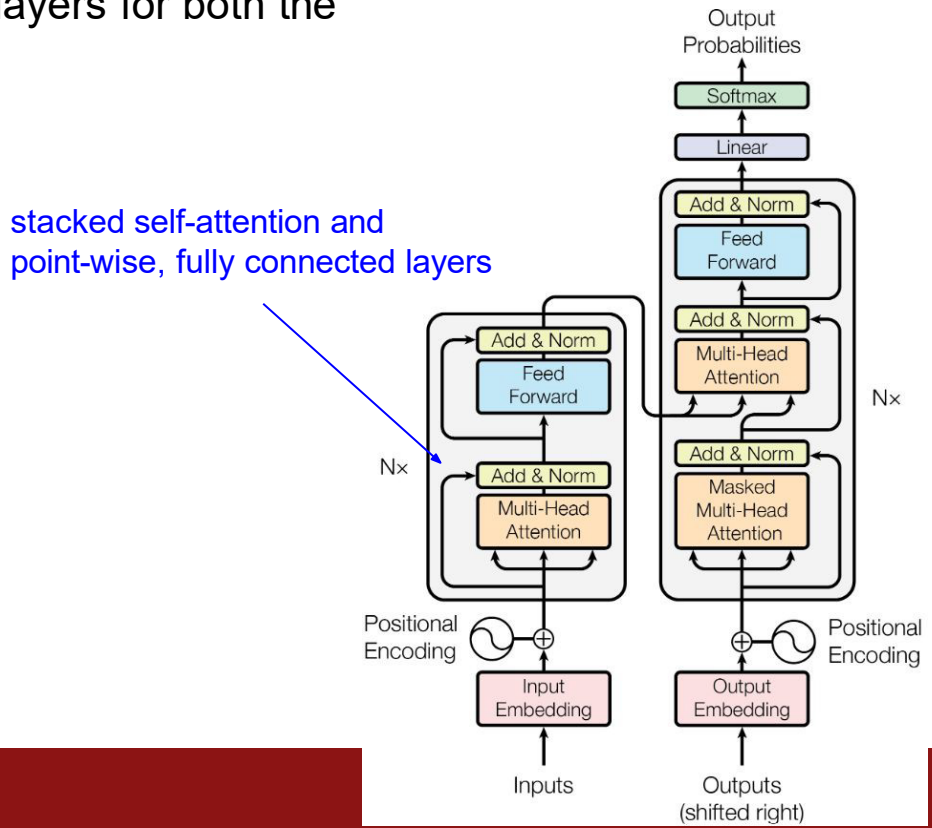
1. Starts with splitting the input words into single characters
   (each of them corresponds to a symbol in the final vocabulary)
   * In practice we commonly add special end of word symbol "__" before space
2. Find the most frequent occurring pair of symbols from the current vocabulary
3. Add this to the vocabulary and size of vocabulary increases by one
4. Repeat steps (2) and (3) till the defined number of tokens are built
   or no new combination of symbols exist with required frequency

Model Architecture

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder

Components of the model:

1. Encoder and Decoder Stacks
2. Attention
3. Poisition-wise Feedforward Networks
4. Embeddings and Softmax
5. Positional Encoding



Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of N = 6 identical layers

- Each layer has two sub-layers
  - First is a multi-head self-attention mechanism
  - Second is a simple, position-wise fully connected feed-forward network

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

Attention

Scaled Dot-Product Attention

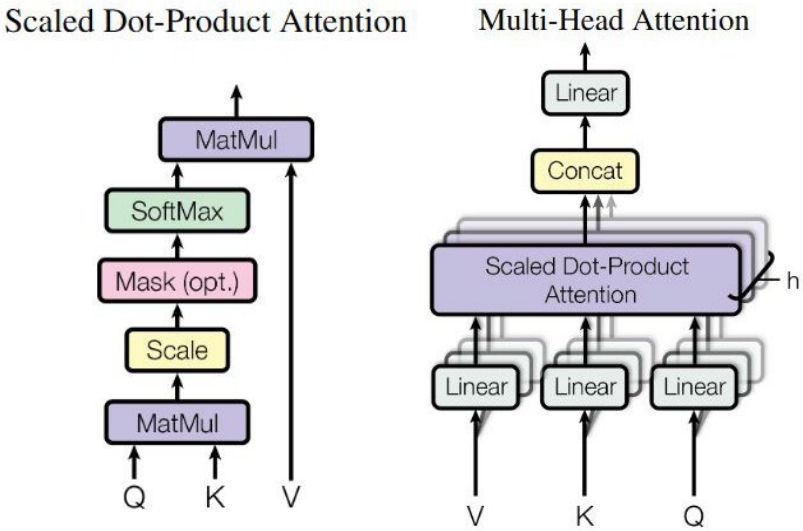
- Compute the attention function on a set of queries simultaneously, packed together into a matrix Q
- The keys and values are also packed together into matrices K and V

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

Multi-Head Attention

Beneficial to linearly project the queries, keys and values h times with different, learned linear projections to d_k, d_k and d_v dimensions

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Positional Encoding

Positional encoding describes the location or position of an entity in a sequence so that each position is assigned a unique representation

Transformers use a smart positional encoding scheme, where each position/index is mapped to a vector: the output of the positional encoding layer is a matrix, where each row of the matrix represents an encoded object of the sequence summed with its positional information

Uses sine and cosine functions of different frequencies:

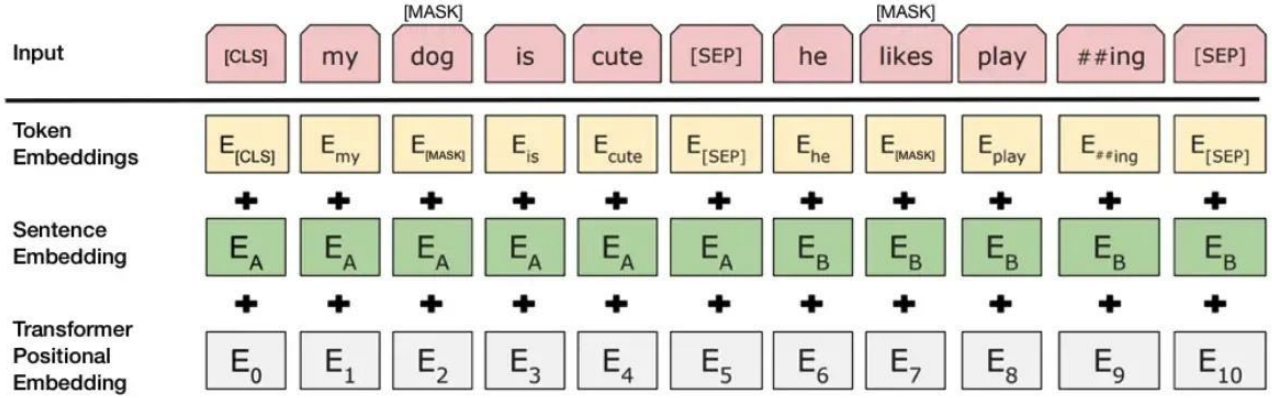
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension

This YouTube video is recommended

BERT: Under Hood



To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings, with a vocabulary of 2
3. A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper