16/2/2022

BERT Research Series

BERT

- Released in October/2018
- Surpassed human performance in language understanding

* Transfer learning
  - "Take the model, modify it by changing the output layer, doing a finetune training"

=

Leverage pretraining =

"Bert is huge and expensive to train from scratch"

"BERT maintain"

"self-attention"

"Cttention is all you need"

x

"LSTM with attention"

My playlist today. # _____

Word embedding

Feature representation of a word

<0.4, 0.6, ..., 0.5>

BERT = 30K tokens × 768 features

dictionary = word string → word id

distance between word embeddings

↕

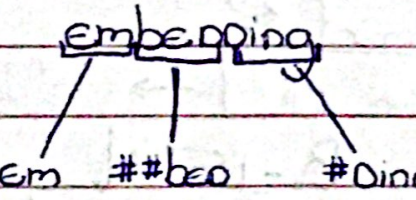
distance between word meanings

=

Word similarity

BERT's vocabulary

1. Bert is pretrained → fixed vocabulary
2. Breakdown unknown words into subwords



3. A subword exists for every character

~ all subwords start with ## except for the first subword

From perusing the vocabulary, we saw that:
- The first 999 tokens (1-indexed) appears to be reserved and are most of the form [unused957].

1 - [PAD]
101 - [UNK]
102 - [CLS]
103 - [SEP]
104 - [MASK]

- Revs 1000-1996 appear to be a dump of individual characters

"They don't appear to be sorted by frequency (e.g. the letters of the alphabet are all sequenced).

- The first word is "the" at the position 1997
  * From there, the words are sorted by frequency
  * From the top ~18 words are whole words, and the number 2016 is ##s, presumably the most common subword

~3k person names in the vocabulary
~ many numbers

My playlist today: #

BERT already encodes a lot about our language
↳ only needs finetuning

Advantages of Fine-Tuning
1. Quicker development
2. Less data required
3. Better results

× Not train specific
"Plug and finature"

BERT shortcomings
1. BERT is very large
   a. Slow finetuning
   b. Slow inference

Layer	Weights
Embedding	~24M
Transformers×12	~7×12=85
TOTAL	~109M
↳ ~417MB	

Distillation ←
Remove weights
while maintaining accuracy

2. Jargon (domain-specific language)

3. Not all NLP applications

Yes
- Classification
- NER
- POS
- Extractive QA

No
- Language model
- Text generation
- Translation

Required formatting

> Special tokens

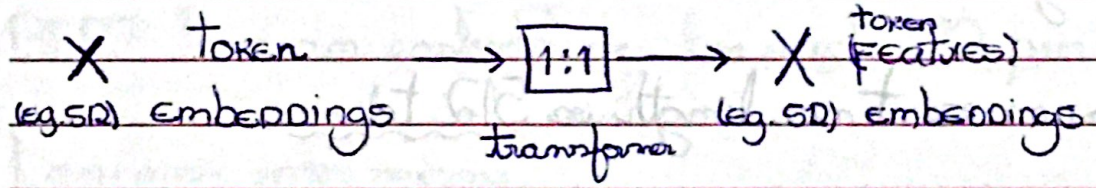
[SEP] separation - separates two sentences

[CLF] classification - must be the 1st token

Architecture

BERT = stack of 12 transformers

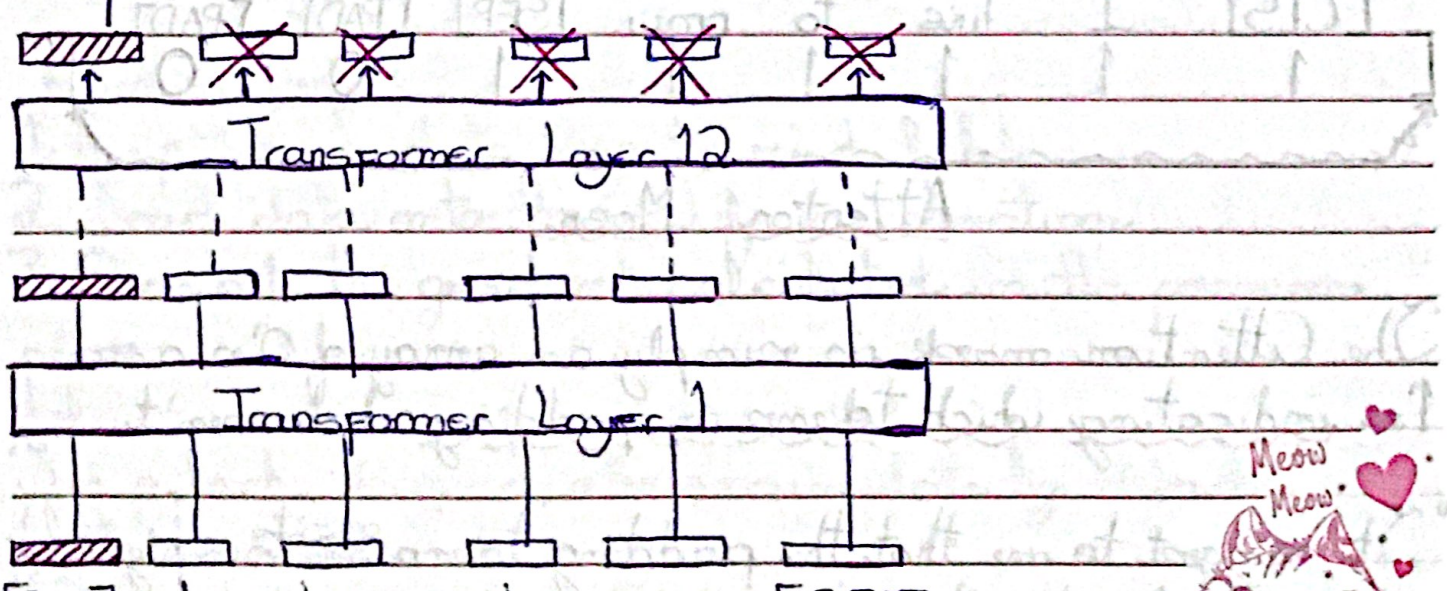
each transformer



Prediction

↑

Classifier



My playlist today # each token has its embeddings

17/02/2022

"The first token of every sequence is always a special classification token ([CLS]). The hidden state corresponding to this token is used as the aggregate representation for classification tasks." (from the BERT paper)

Sequence length & Attention masks

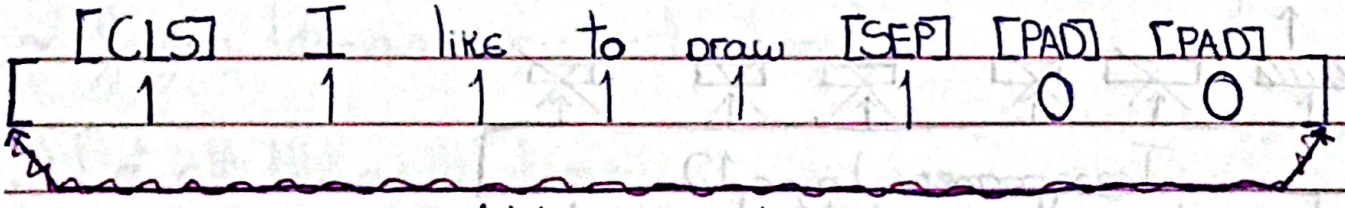
→ How does BERT handle with sentences with varying lengths?

BERT has two constraints:

1. All sentences must be padded or truncated to a single, fixed ~~lenght~~ length.

2. The maximum sentence length is 512 tokens.

Padding is done with a special [PAD] token, which is indexed at 0 in the BERT vocabulary.



Attention Mask

The Attention mask is simply an array of 0's and 1's indicating which tokens are padding which aren't.

"It suggest to me that the padding tokens ~~are~~ aren't simply skipped over" → he show that max seq len affects in prediction acc.

Preprocessing steps

1. Convert sentence into list of vocabulary ID's
2. Pad or truncate
3. Calculate attention masks

Fine-tuning

BertForSequenceClassification

=

Load the pretrained BERT model with a single linear classification on top

BERT has an embedding for each 512 positions

Learning rate decay

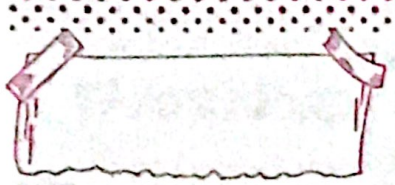
=

Initially larger steps, later smaller steps
↳ depends on scheduler

Training loop

1. Unpack our data inputs and labels
2. Load data onto the GPU for acceleration
3. Clear out the gradient calculated in the previous pass
↓
In pytorch, the gradients accumulate by default (better for RNN)
4. Forward pass (feed the data through the

My playlist today: #




network)
5. Backward pass (backpropagation)
6. Tell the network to update parameters
7. Track variables for monitoring progress

Evaluation loop
1. Unpack our data inputs and labels
2. Load data onto GPU for acceleration
3. Forward pass
4. Compute loss on our validation data and track variables for monitoring progress

Dropout "drop out" and "batch normalization" behave differently during training and test. *

Batch contains three pytorch tensors:
[0]: input idx
[1]: attention masks
[2]

gradient clipping avoids gradients to explode.

18/02/2022

BERT = Encoder$^{\times 12}$ + ~~Decoder~~

Transformer = Encoder$^{\times 6}$ + Decoder$^{\times 6}$

Transformers Illustrated - Jay Allamar

Transformers use attention in a different way → self-attention

Neural Machine Translation (NMT)

↳ Don't use hand engineered features

Transformers outperformed Google NMT

Before Google Translate

English → LSTM-RNN → Vector → LSTM → French

LSTM-RNN: Encoder; LSTM: Decoder

Probability 2nd word in French = $\mathcal{F}$(1st word in English, 2nd word vector)

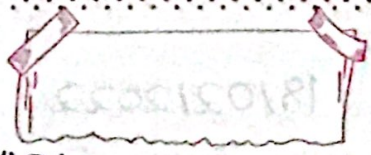
Stopping tecnique → [EOS] End of Sentence in vocabulary

↳ when [EOS] is more likely to be the next word

Google added attention

→ Attention mechanism tells which word should be focused on

My playlist today: #




"The biggest benefit from Transformers comes from how the Transformers lends itself to parallelization."

Transformer = Encoder $^{\times 6}$ + Decoder $^{\times 6}$

> stacks of decoder encoders and decoders
> same architecture but different weights

→ This first half is interpreting well the input data, could we use it for other tasks instead of translation =

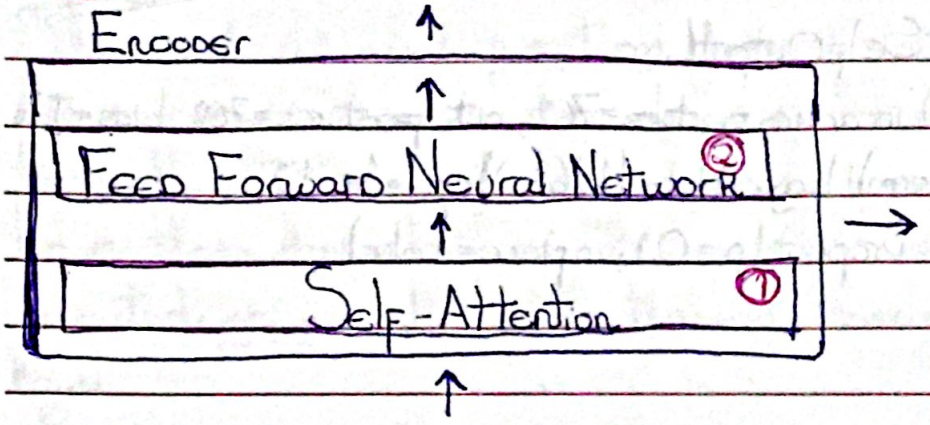
↳ BERT's idea
  ↳ Use Transformer's encoder
  ↳ 12 encoders instead of 6
  ↳ Train in a huge dataset

↳ If we remove the decoder, the translation tasks, how do we train these encoders? ★

"fake-tasks"

1. Masked Language Model (MLM) - predict the masked word
2. Next Sentence Prediction (NSP)
   ↳ Was sentence B found immediately after sentence A?

You can apply BERT for different tasks by changing the output layer



Feed Forward Neural Network
↳ Standard MLP dense neural network layer

~ From BERT HF parameters

(embeddings): BertEmbeddings(
(word_embeddings): Embedding(30522, 768, padding_idx=0) — vocab
(position_embeddings): Embedding(512, 768) — max seq. len
(token_type_embeddings): Embedding(2, 768)
★ (LayerNorm): LayerNorm((768,), eps=1e-12)
(dropout): Dropout(p=0.1, inplace=False)
)

dimension = 768 = 6×128 (or 512+256), 50% longer than original Transformers
★ Layer Normalization (2016) = reduce training time

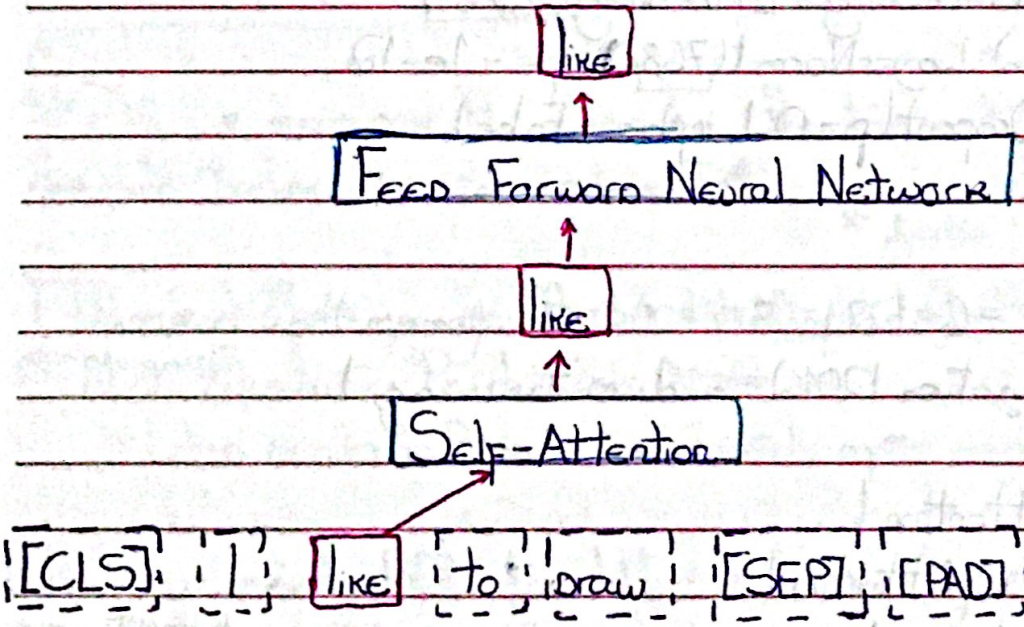
(attention): BertAttention(
(self): BertSelfAttention(
① (query): Linear(in_features=768, out_features=768, bias=True)
(key): Linear(in_features=768, out_features=768, bias=True)
(value): Linear(in_features=768, out_features=768, bias=True)
)

My playlist today: #

(output): BertSelfOutput(
(dense): Linear(in_features=768, out_features=768, bias=True)
(LayerNorm): LayerNorm((768,), eps=1e-12,
(dropout): Dropout(p=0.1, inplace=False)
)
)

② Self-Attention

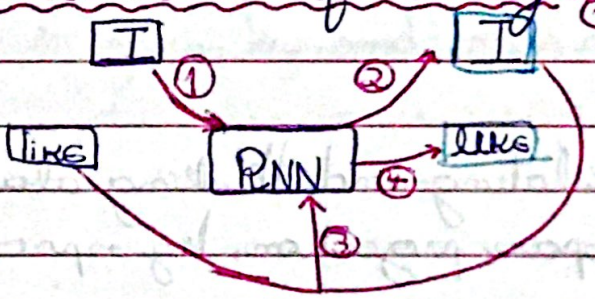
A layer that helps the encoder look at other words in the input sentence as it encodes a specific word. The outputs of the self-attention are feed-forward neural network.



20/02/2022

The word in each position flows through its own path in the encoder. There are dependencies between these paths in the self attention layer. The feed forward layer does not have these dependencies. However, the various paths can be executed in parallel while flowing through the feed forward layer.

RNN has to run sequentially I → like → to
① ② ③ ④



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

Self-attention allows to associate it with animal

Self-attention
↳ Given an input word, score every other word by relevance

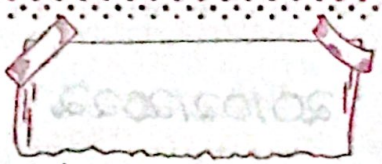
Self-Attention in detail

vectors:

Word embedding x_i → query q_i, Key k_i, Value v_i

"Thinking machines"
x_1 x_2

My playlist today: #




1st step - Calculate query, Key, and values
q_i k_i v_i

Eg "Thinking machines" and encoding (thinking) ← W^Q
x_1 x_2

Thinking (x_1)
- → $x_1 \cdot W^Q = q_1$
- → $x_1 \cdot W^K = k_1$
- → $x_1 \cdot W^V = v_1$

Machines (x_2)
- → $x_2 \cdot W^Q = q_2$
- → $x_2 \cdot W^K = k_2$
- → $x_2 \cdot W^V = v_2$

W^Q (query weights), W^K, W^V

- Obstractions that help calculating and thinking about attention
- Projection features: query space projection, key space ....

2nd step - Calculate scores
Score of word w_i while encoding word w_j is given by:
$score(i,j) = q_j \cdot k_i$

Score (Thinking w_1) $= q_1 \cdot k_1$ Score (Machines w_2) $= q_2 \cdot k_2$

Query space is about the word being encoded ←
Key space is about the word being calculated

3rd and 4th steps - "Score Normalizing"

$$= \text{softmax}\left(\frac{score(i,j)}{\sqrt{len(k_i)}}\right)$$

Divide scores by 8 (square root of Key dimension) and normalize through softmax leading to more stable gradients.

We can do it!

FORONI

→ Value space is about word relevance

5th - multiply by w_i, the value vector by the softmax scores. The intuition here is to drown out irrelevant words.

6th - sum up weighted value word vector

~ All these calculations are made in matrix for faster processing:

1st calculate query, key, and value matrixes

$X \cdot W^Q = Q$

$X \cdot W^K = K$

$X \cdot W^V = V$

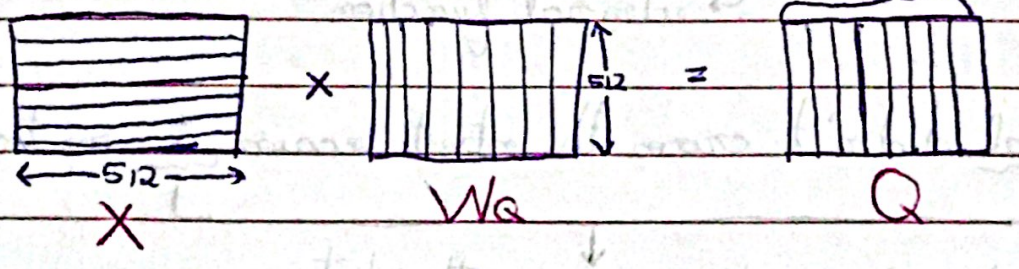
2nd to 6th steps

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

Multi-headed attention

Improves the performance of attention layer in two ways:

1. Expands the model's ability to focus on different positions
   ↳ find different local minimums
2. Multiple "representation subspaces"
   - Multiple sets of query/key/value: 8 attentions - 8 sets
   - Each randomly initialized

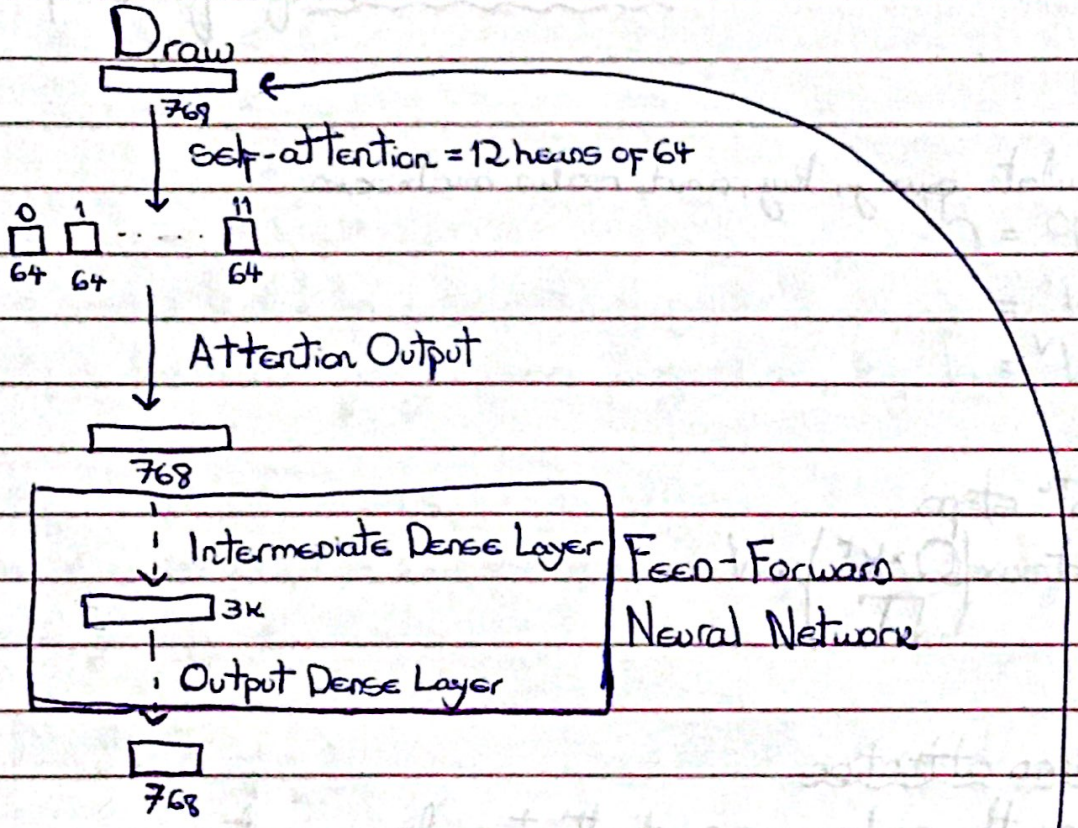


My playlist today: #

Transformer
- 8 heads len 64
- len 512 embeddings

BERT
- 12 heads len 64
- len 768 embeddings

Feed Forward Neural Network



Positional Encoding

- RNN does it naturally.

what is fed to the model

embeddings + positional encoding = encoding with signal

↳ external function

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

Positional encoding considers this distance

BERT "Pre-training" tasks

- Masked Language Model (MLM) → cloze task - very known on NLP
- Next Sentence Prediction (NSP)

The training loss is the sum of the mean MLM likelihood and the NSP likehood.

They are sample such that combined length is ≤ 512 tokens.

My playlist today: #

BERT "Pre-training" tasks

- Masked Language Model (MLM) → cloze task - very known on NLP
- Next Sentence Prediction (NSP)

The training loss is the sum of the mean MLM likelihood and the NSP likehood.

They are sample such that combined length is ≤ 512 tokens.

My playlist today: #