Warta or not?

Mikołaj Grzywacz

NLP

1. How to represent text?

NLP

1. How to represent text?
2. Vectorization
   a. Bag of Words

Bag of words

Warta wprowadziła nową ofertą ubezpieczeń k
Warta stworzyła ofertę gwarancji środowiskowej

[[0 1 1 0 0 1 1 1 0]
 [1 0 0 1 1 0 1 0 1]]

{'warta': 6,
'wprowadziła': 7,
'nową': 1,
'ofertą': 2,
'ubezpieczeń': 5,
'stworzyła': 4,
'ofertę': 3,
'gwarancji': 0,
'środowiskowej': 8}

Note:
 Information is lost -> sequence

NLP

1. How to represent text?
2. Vectorization
   a. Bag of Words
   b. TF-IDF

TF-IDF

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term *x* within document *y*

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

TF-IDF

Looks at how important the word is to the document (scraped website) in a collection of documents.

Still we lose information about ordering.

NLP

1. How to represent text?
2. Vectorization
   a. Bag of Words
   b. TF-IDF
3. Text normalization techniques

Text normalization

- Context-independent normalization: removing non-alphanumeric text symbols
- All lowercase
- Removing stop words
- Canonicalization: convert data to "standard", "normal", or canonical form.
  - Stemming: extracts the word's root.
  - Lemmatization: transforms word to its lemma.

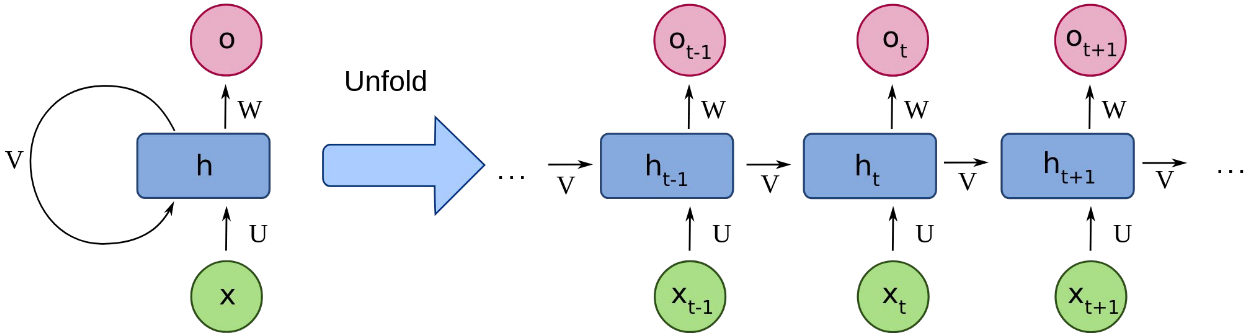
NLP

1. How to represent text?
2. Vectorization
   a. Bag of Words
   b. TF-IDF
3. Preprocessing techniques
4. Making predictions, for example:
   a. SVM
   b. XGBoost

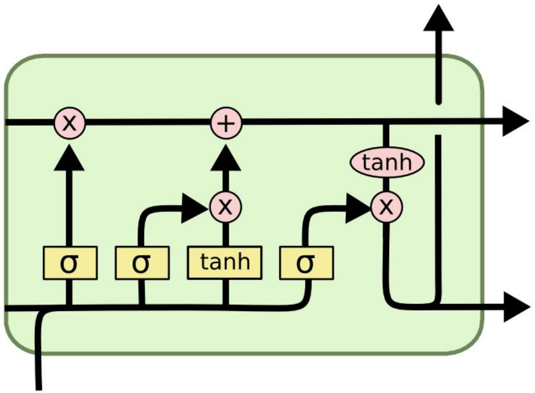
NLP

What if we need information about word ordering?

RNN

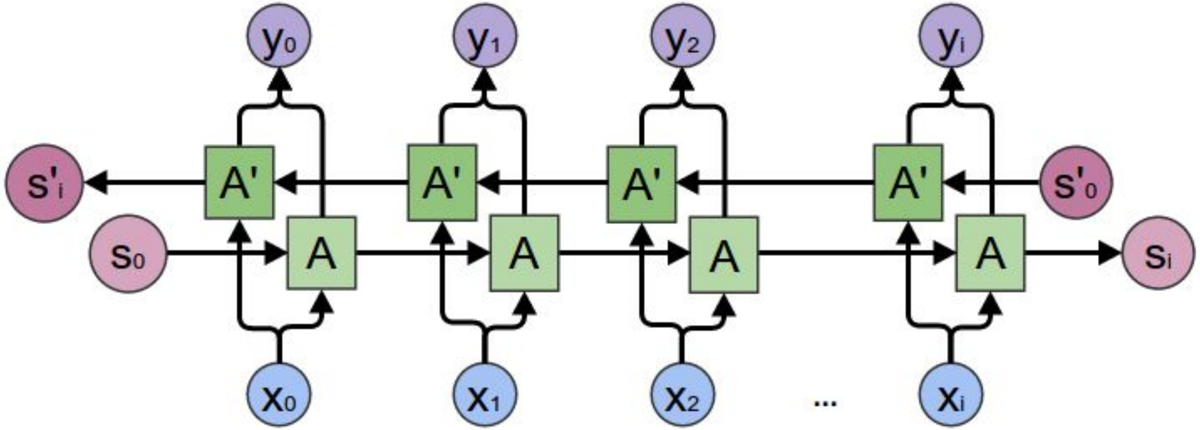

o
W
V
h
U
x
Unfold
...
V
o_{t-1}
h_{t-1}
x_{t-1}
o_t
h_t
x_t
o_{t+1}
h_{t+1}
x_{t+1}
...

LSTM


x
+
tanh
x
x
σ
σ
tanh
σ

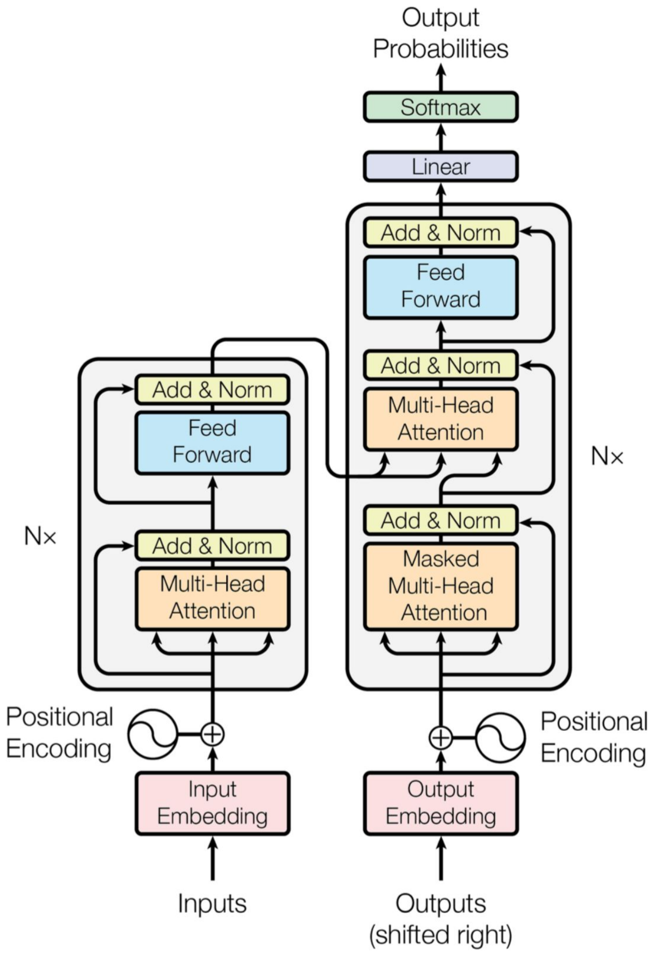
Transformers

1. Why LSTMs are not that great?
    a. LSTMs are very slow
    b. Bidirectional LSTMs are not “really bidirectional”



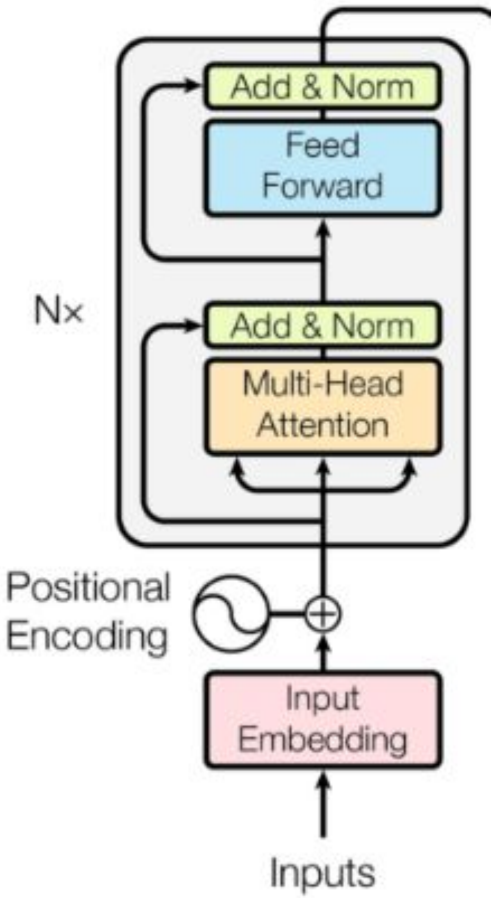
Transformer architecture

1. Encoder
2. Decoder

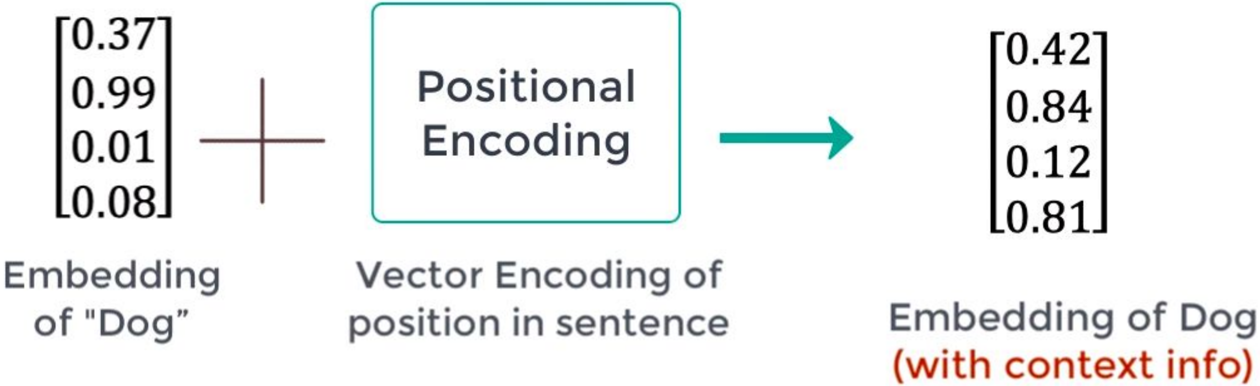


Encoder

1. Input Embedding
2. Positional encoding

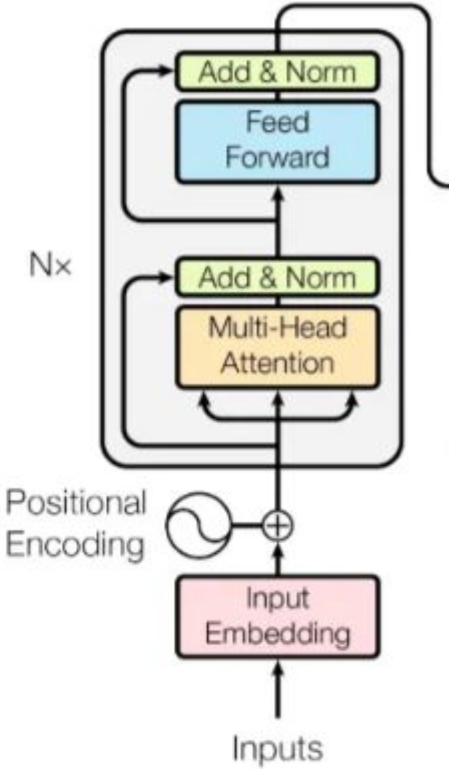


Encoder - input encoding

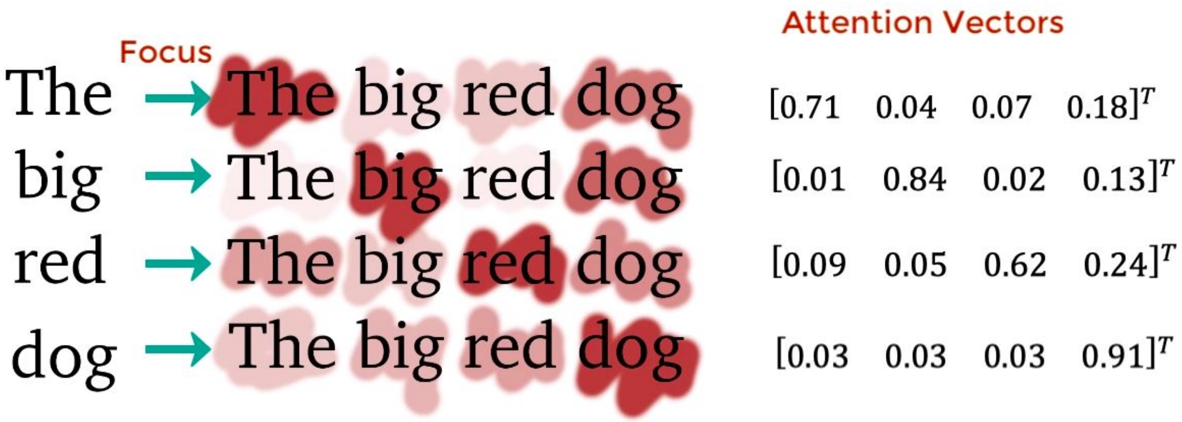


Encoder

1. Input Embedding
2. Positional encoding
3. Attention block

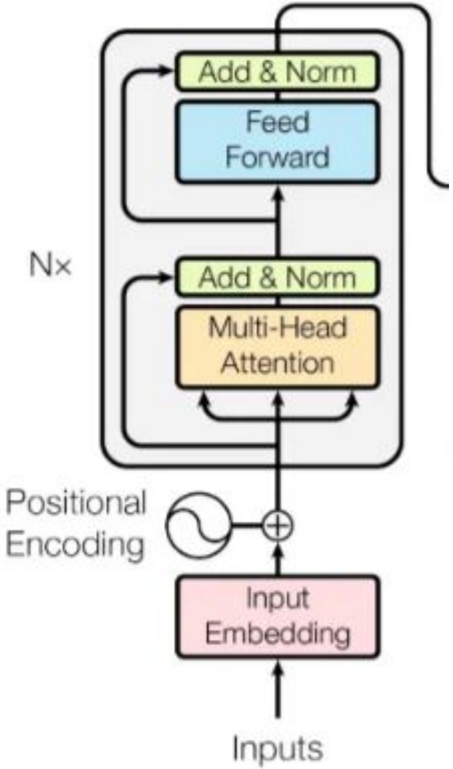


Encoder - attention

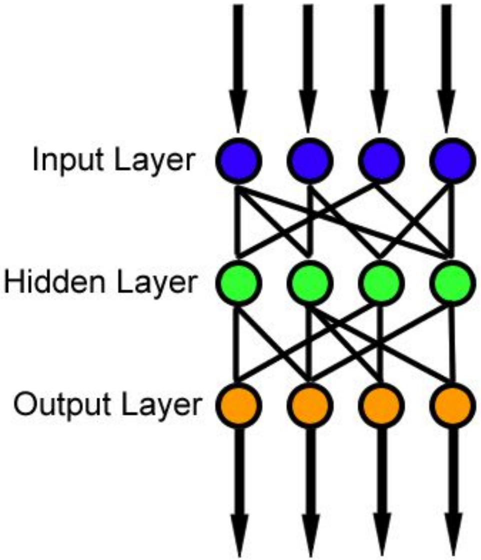


Encoder

1. Input Embedding
2. Positional encoding
3. Attention block
4. Feed forward

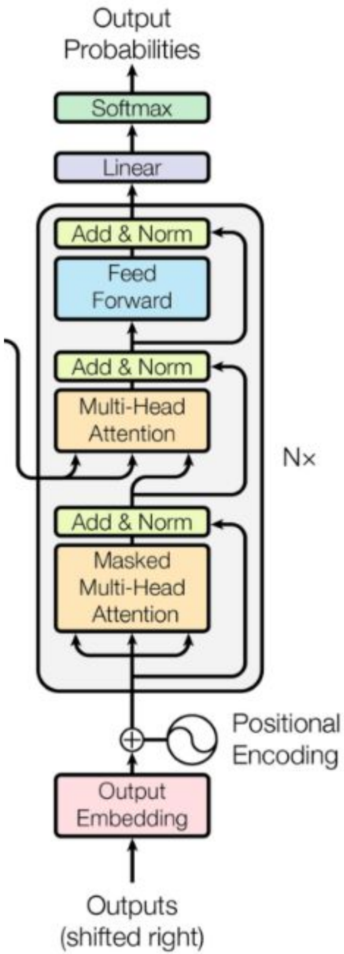


Encoder - feed forward



Decoder

1. What is the input?
2. Input Embedding
3. Positional encoding
4. Attention block 1 (decoder)
5. Attention block 2 (encoder-decoder)
6. Feed forward
7. Linear + Softmax



Decoder - attention blocks

Takes encoder outputs
Takes attention last layer

Outputs attention between all words

Output - next word!

$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad \begin{bmatrix}0.1\\0.9\\0\\0\end{bmatrix} \quad \begin{bmatrix}0.05\\0.40\\0.55\\0\end{bmatrix} \quad \begin{bmatrix}0.16\\0.09\\0.15\\0.66\end{bmatrix}$$

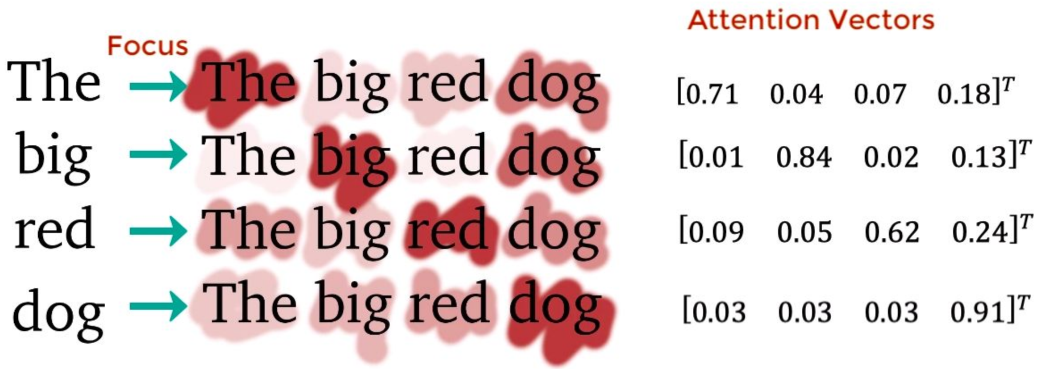
Ten	duży	czerwony	pies

$$\begin{bmatrix}0.71\\0.04\\0.07\\0.18\end{bmatrix} \quad \begin{bmatrix}0.01\\0.84\\0.02\\0.13\end{bmatrix} \quad \begin{bmatrix}0.09\\0.05\\0.62\\0.24\end{bmatrix} \quad \begin{bmatrix}0.03\\0.03\\0.03\\0.91\end{bmatrix}$$

This	big	red	dog

Transformers - attention (multi - head)

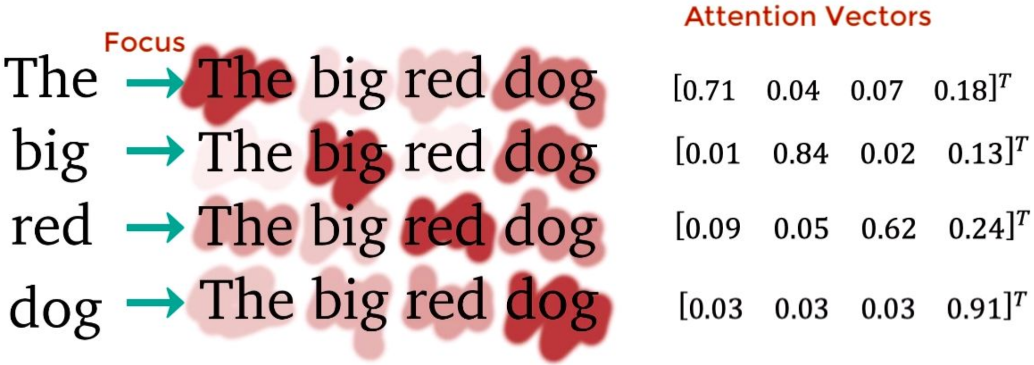
What is multi - head?



Transformers - attention (multi - head)

What is multi?
Runs through attention generation mechanism in parallel several times.

Why?

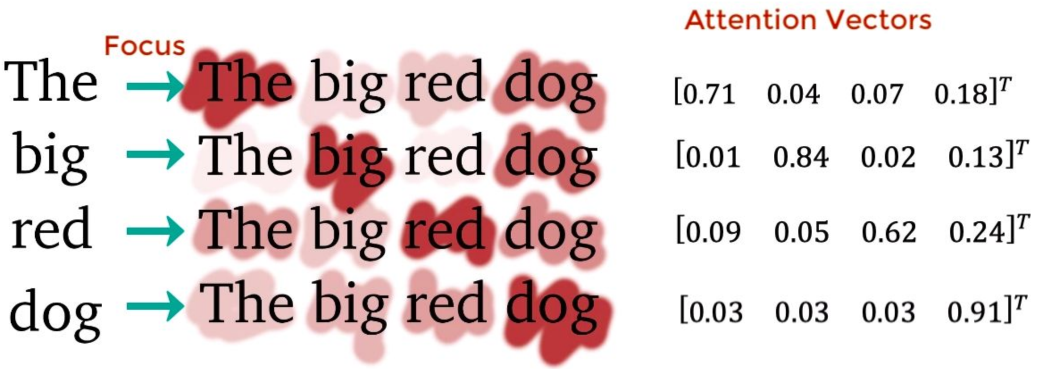


Transformers - attention (multi - head)

What is multi?
Runs through attention generation mechanism in parallel several times.

Why?
Intuitively it helps model to attend to different part of sentence differently
Long term vs short term dependencies

Then concatenate results

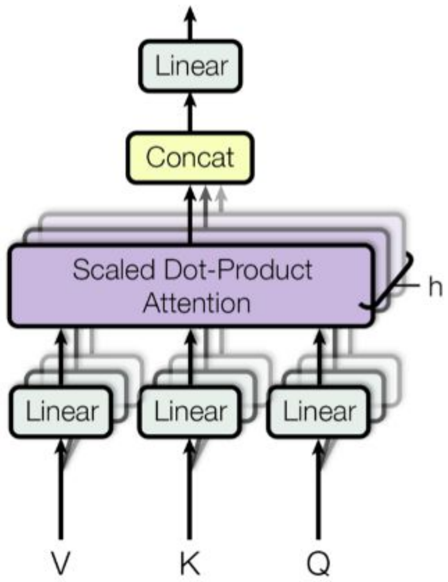


Transformers - attention (multi - head)

V K Q - different input vectors for word.

Concat is needed to bring multiple attention vectors to a single vector.

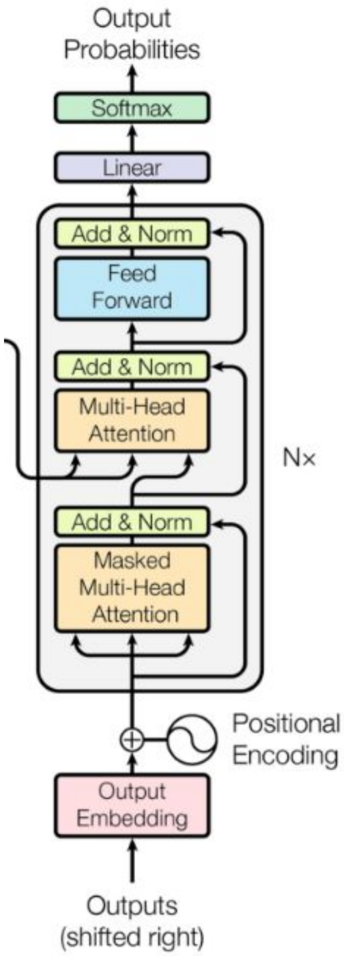
$$Z = softmax\left(\frac{Q.K^T}{\sqrt{Dimension\ of\ vector\ Q, K\ or\ V}}\right).V$$



Decoder - attention (masked?)

This
big
red
dog

Ten duży czerwony pies



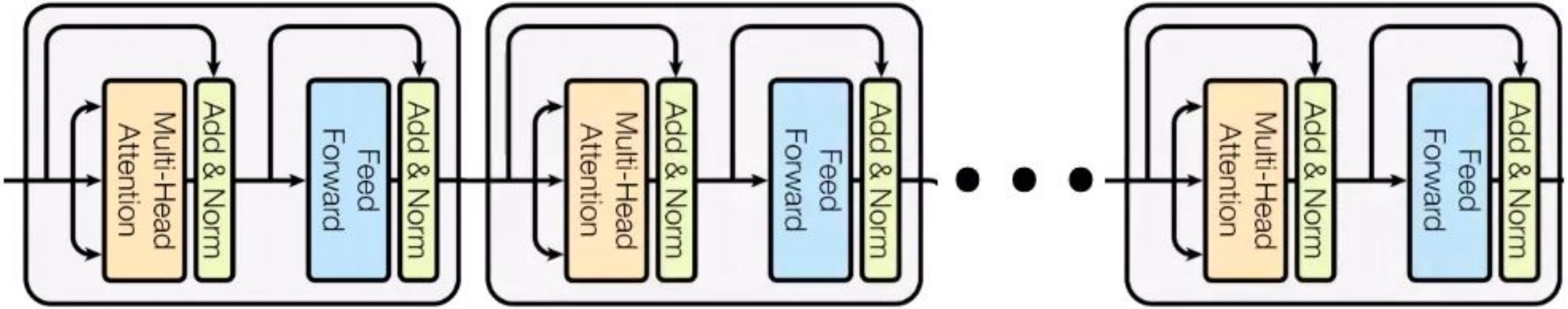
BERT

What is BERT then?

BERT

Bidirectional **E**ncoder **R**epresentation from **T**ransformers

Stack encoders!!!



BERT

Problems to solve in NLP:

1. Translation
2. Sentiment analysis
3. Question answering
4. Text summarization

BERT

Problems to solve in NLP:

1. Translation
2. Sentiment analysis
3. Question answering
4. Text summarization

How to solve them:

- Pre train BERT to understand language
- Fine Tune BERT for specific task

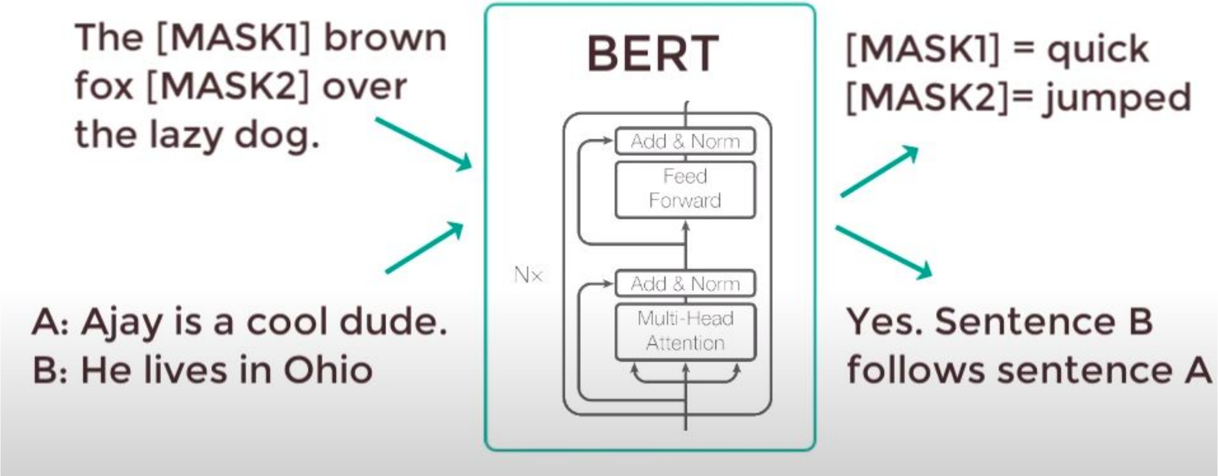
BERT - pretraining

Tries to learn: What is language? What is context?

BERT - pretraining (1)

How to learn - what is language?
Masked **L**anguage **M**odeling - MLM

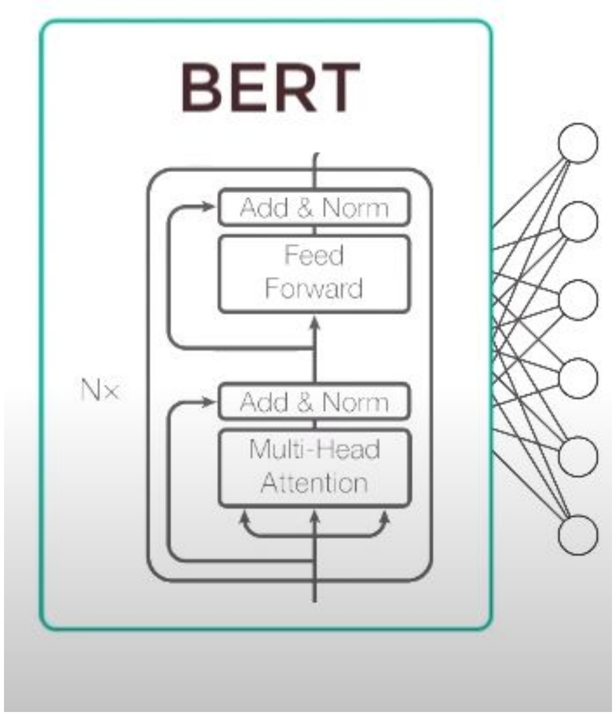
Next **S**entence **P**rediction - NSP



BERT - fine tuning (1)

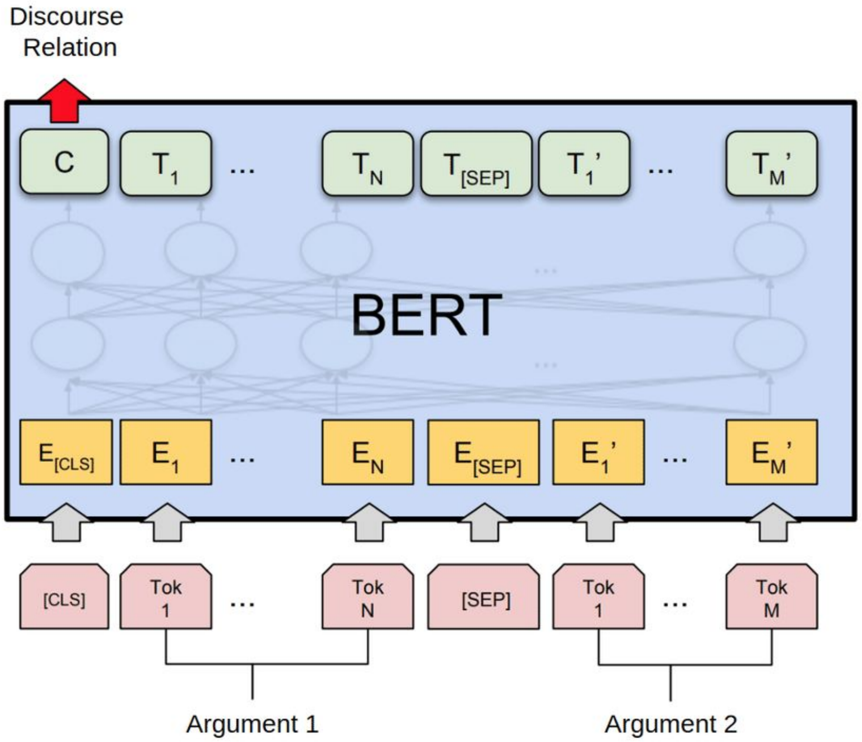
Supervised training to specific task

1. Two approaches
2. Either way, only needs to learn new weights



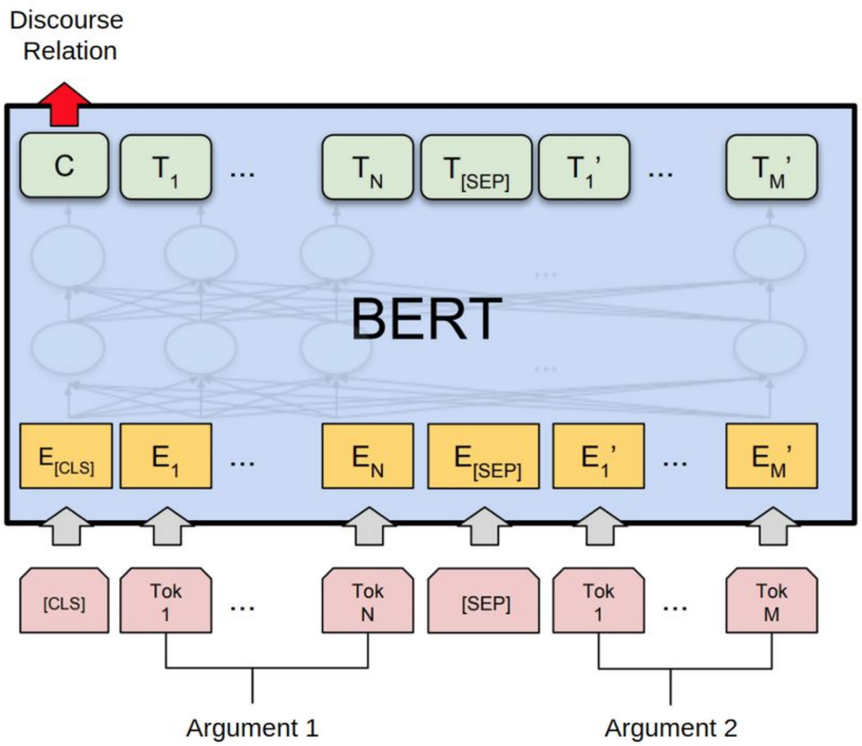
BERT - pretraining (2)

1. Word token inputs



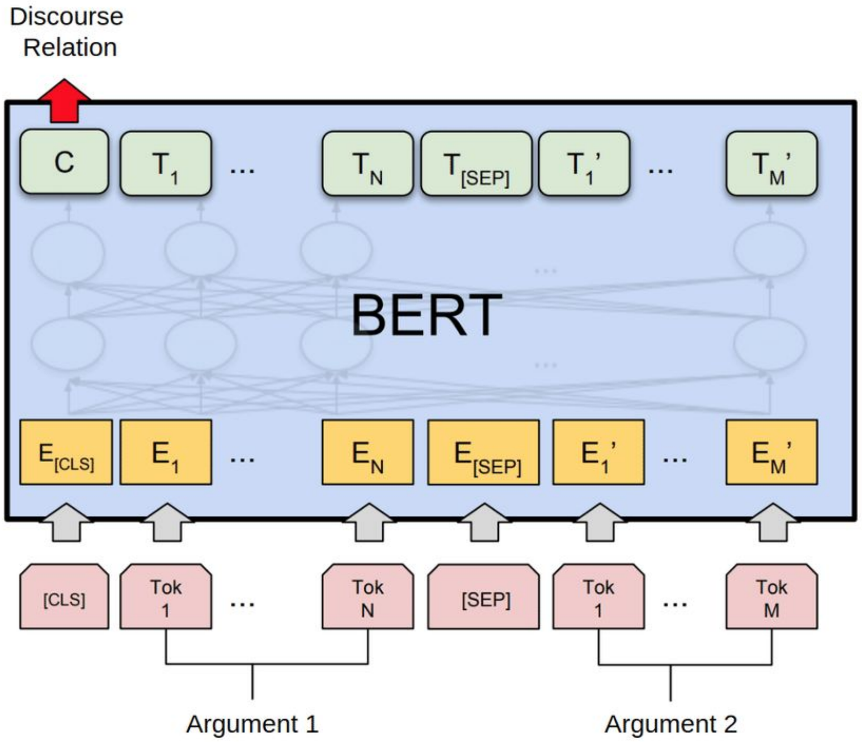
BERT - pretraining (2)

1. Word token inputs
2. Embeddings



BERT - pretraining (2)

1. Word token inputs
2. Embeddings
3. Output (C, T)



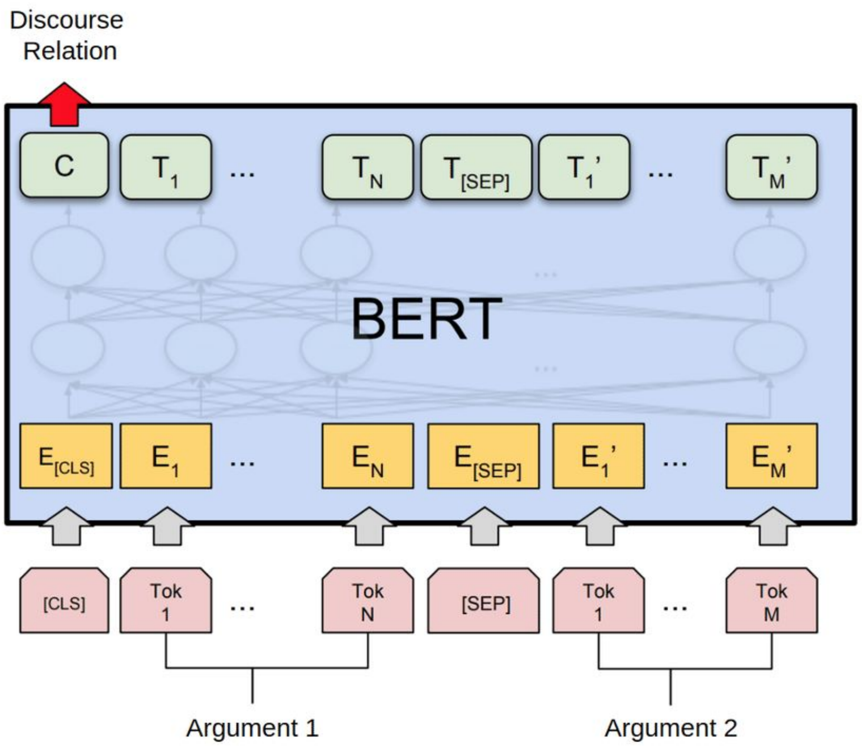
BERT - pretraining (2)

NSP

- Understand relationships
- 50% / 50%

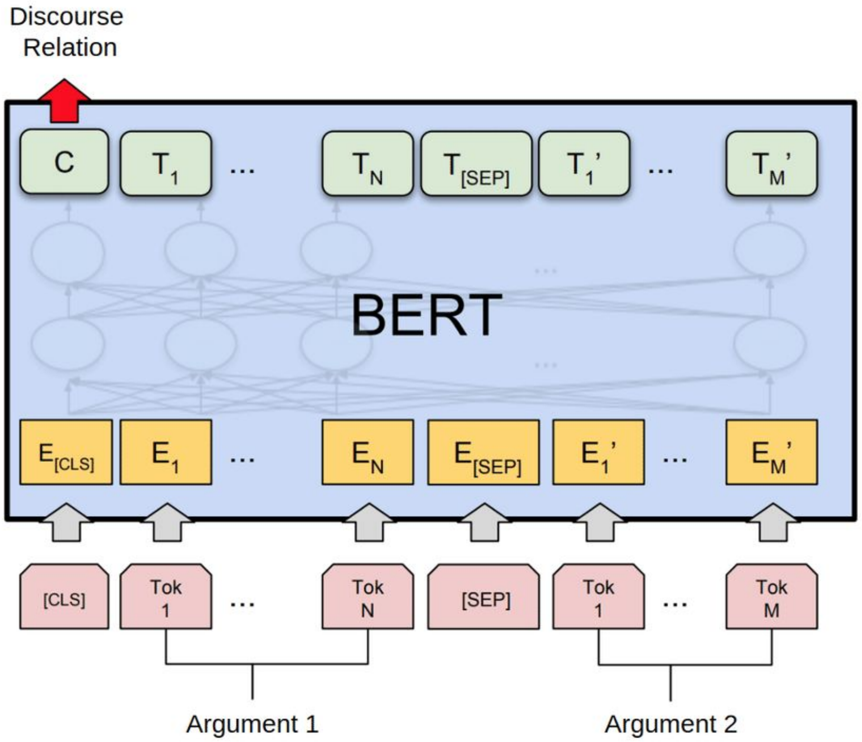
MLM

- “Deeply bidirectional”
- 15% masked
    a. 80% / 10% /10%
- Why?



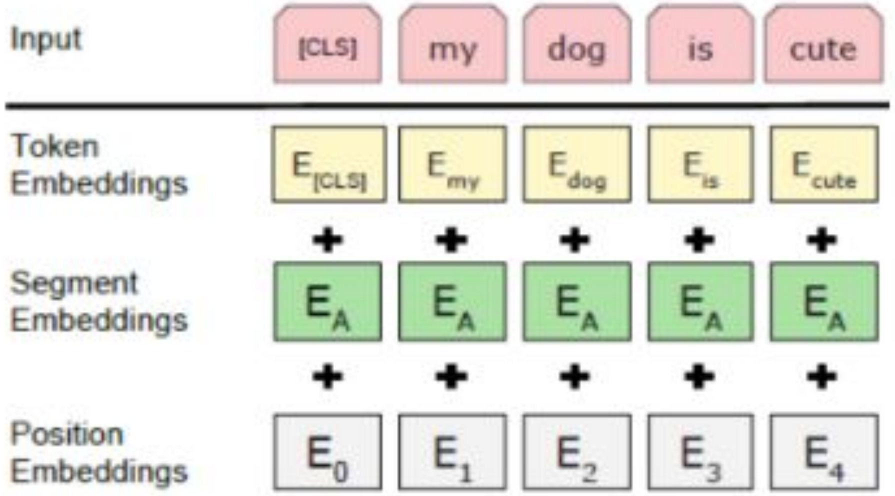
BERT - pretraining (2)

1. Word token inputs
2. Embeddings
3. Output (C, T)
4. What exactly are this embeddings?



BERT - pretraining (2)

Embeddings for BERT
1. Uses WordPiece algorithm
2. Generates around 50000 tokens



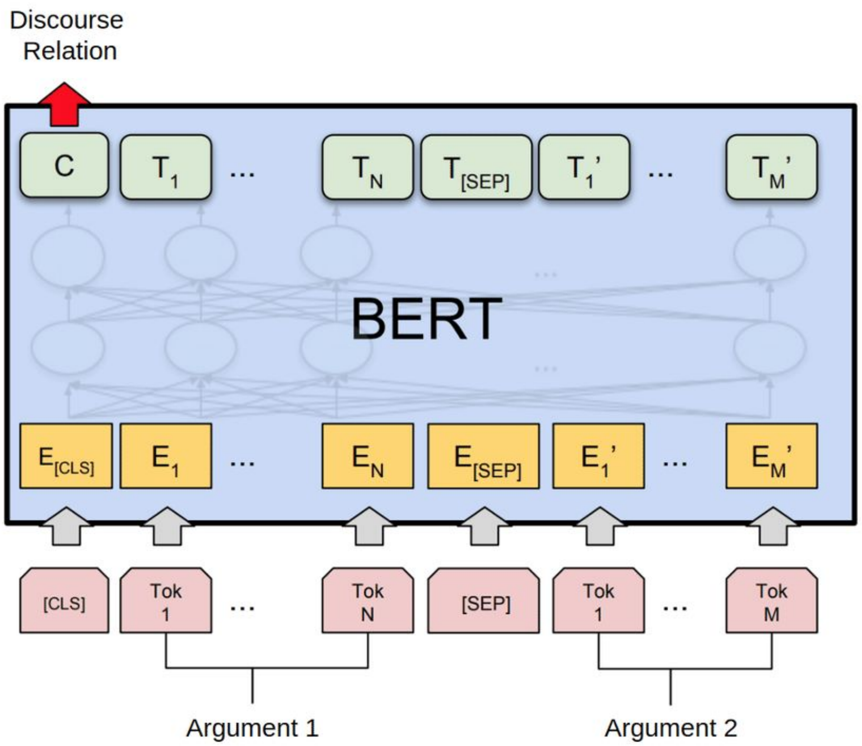
Overview
Example:

text = "Here is the sentence I want embeddings for."
marked_text = "[CLS] " + text + " [SEP]"

the tokenized sentence looks like,
['[CLS]', 'here', 'is', 'the', 'sentence', 'i', 'want', 'em', '##bed', '##ding', '##s', 'for', '.', '[SEP]']

BERT - pretraining (2)

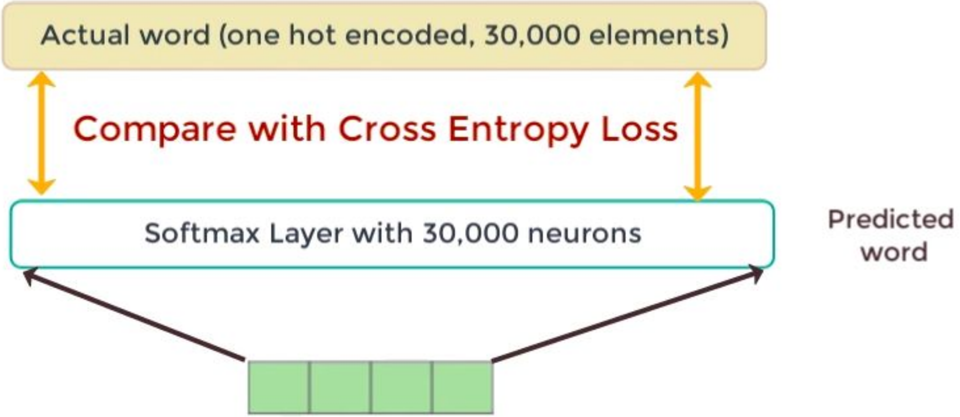
1. Word token inputs
2. Embeddings
3. Output (C, T)
4. What exactly are this embeddings?
5. Final look at result layer



BERT - pretraining (2)

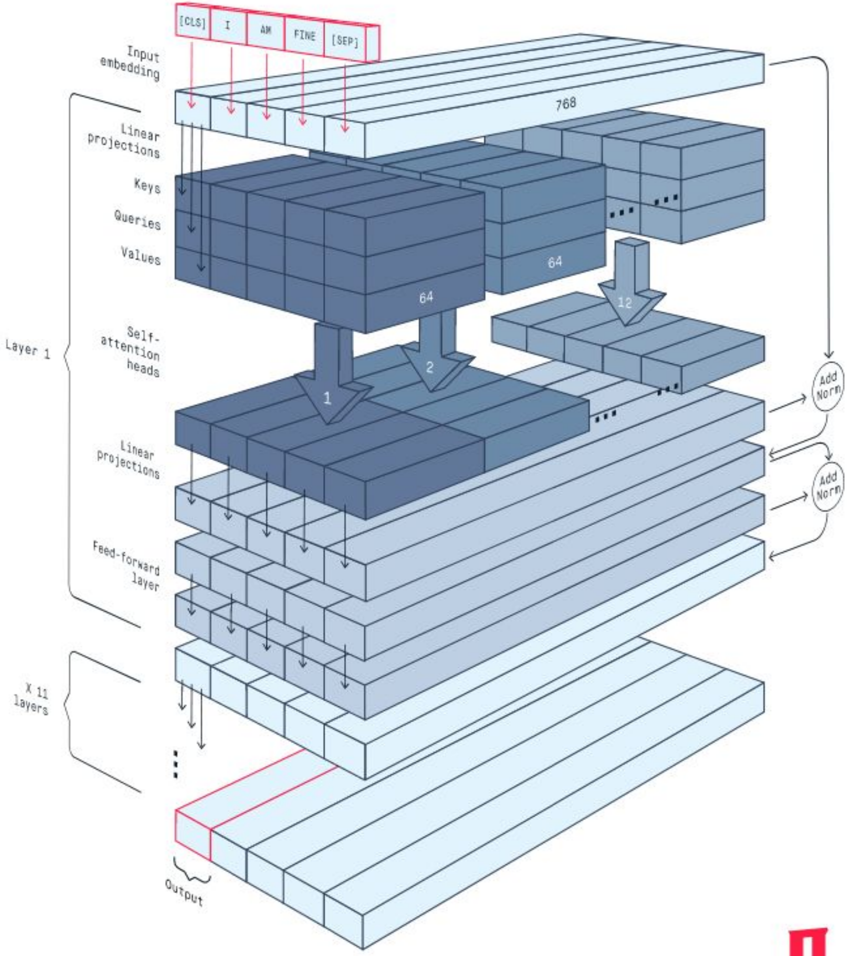
How does the single T output look like?

Calculate loss only on masked words.



BERT - overview

- 12 successive transformer layers
- Encoders only, no decoders
- 12 attention heads for each layer
- The total number of parameters is 110 million
- Outputs one vector for each xi (size 768)



ROBERTA

Robustly **O**ptimized **BERT** Pre-Training **A**pproach

Why ROBERTA is better:

- More training data (16G vs 160G)
- Uses dynamic masking pattern instead of static masking pattern
- Replacing next sentence prediction objective with full sentences without NSP
- Training on Longer Sequences

ROBERTA

More Training Data

- RoBERTa uses BookCorpus (16G), CC-NEWS (76G), OpenWebText (38G) and Stories (31G) data
- BERT uses BookCorpus (16G) as training data only

ROBERTA - dynamic masking

In bert sequences are masked once (static masking)

Same pattern is used in all training steps.

Duplicate data 10 times -> 10 different patterns (40 epochs)

Dynamic masking -> generated every time sequence is passed

ROBERTA -
Is NSP needed?

- **SEGMENT-PAIR with NSP:** A pair of segments where there could be multiple sentences. BERT uses this training objective. The number of token is less than 512.
- **SENTENCE-PAIR with NSP:** A pair of sentences from the same or different documents. It is slightly different from the original BERT approach. The number of token is significantly less than 512.
- **FULL-SENTENCES without NSP:** Inputs are packed with sentences that are sampling from one or more documents. When training data is reached the end of document, sentences from other documents will be sampled. We also add an extra separator token between the documents. The number of token is at most 512. **This is the training objective RoBERTa uses.**
- **DOC-SENTENCES without NSP:** Inputs are similar to **FULL-SENTENCES without NSP** except they do not cross document boundaries. The number of tokens may be shorter than 512.

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (K = 6)	–/81.0	85.6	93.4	66.7

ROBERTA

More Training Data

- RoBERTa uses BookCorpus (16G), CC-NEWS (76G), OpenWebText (38G) and Stories (31G) data
- BERT uses BookCorpus (16G) as training data only

ROBERTA

Training on Longer Sequences

- BERT-BASE is trained through 1 million steps with a batch size of 256 sequences
- RoBERTa trained 125K steps with 2k sequences

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

References

- https://paperswithcode.com/method/wordpiece
- https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/english-bert-encoder
- https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/tokenizer
- https://www.datasciencecentral.com/profiles/blogs/top-nlp-algorithms-amp-concepts
- https://www.youtube.com/watch?v=TQQlZhbC5ps
- https://www.kaggle.com/c/tweet-sentiment-extraction/discussion/157801