

AI CORE course capstone project

BERT implementation (kind of)

by Tanya Z

What is BERT?

BERT is a method of pretraining language representations and named after its pre-trained unsupervised natural language processing model

One can either use these models to extract high quality language features from text data, or can fine-tune these models on a specific task with your own data to produce state of the art predictions.

What BERT can do?

- Token Classification
 - Named entity recognition
 - Question answering.
- Text Classification
 - Text classification
 - Sentiment analysis
- Text-Pair Classification
 - Next sentence prediction
 - Automatic summarization

What BERT cannot do?

Perform Text Generation (e.g. translate text from one language to another or generate a written response to a question), as it is based on Encoder part of Transformer architecture, but does not have Decoder.

Problem description

As BERT can be fine-tuned for the custom task **sentiment analysis** seems to be an interesting application of the method

Dataset

WikiText-2 Dataset- a small preprocessed version of Wikipedia, containing 200M tokens (Merity et al., 2016)

<https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/>

Each file contains `wiki.train.tokens`, `wiki.valid.tokens`, and `wiki.test.tokens`. No processing is needed other than replacing newlines with `<eos>` tokens.

Example:

= Gold dollar =

The gold dollar or gold one @-@ dollar piece was a coin struck as a regular issue by the United States Bureau of the Mint from 1849 to 1889 . The coin had three types over its lifetime , all designed by Mint Chief Engraver James B. Longacre . The Type 1 issue had the smallest diameter of any United States coin ever minted .

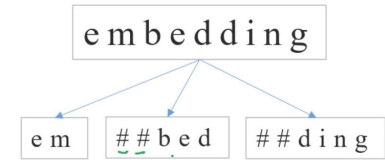
A gold dollar had been proposed several times in the 1830s and 1840s , but was not initially adopted . Congress was finally galvanized into action by the increased supply of bullion caused by the California gold rush , and in 1849 authorized a gold dollar . In its early years , silver coins were being hoarded or exported , and the gold dollar found a ready place in commerce . Silver again circulated after Congress in 1853 required that new coins of that metal be made lighter , and the gold dollar became a rarity in commerce even before federal coins vanished from circulation because of the economic disruption caused by the American Civil War .

Gold did not again circulate in most of the nation until 1879 ; once it did , the gold dollar did not regain its place . In its final years , it was struck in small numbers , causing speculation by hoarders . It was also in demand to be mounted in jewelry . The regular issue gold dollar was last struck in 1889 ; the following year , Congress ended the series .

Pre-processing data (1)

A pre-trained BERT model has an in-built BERT Tokenizer which breaks a raw text string into suitable tokens and looks up the embeddings for these **tokens**:

- As BERT ingests from 3 to 512 input tokens one needs to set a persistent max_length and truncate extra data
- [CLS] token - the beginning
- [SEP] token - special token which is placed in between the sentences
- If model encounters an unknown word, it breaks it down²:
- It discards the whitespace when tokenizing text
- It has tokens for punctuation characters



BERT Tokenizer is based on WordPiece subword tokenization algorithm

The gold dollar or gold one @-@ dollar piece was a coin struck as a regular issue by the United States Bureau of the Mint from 1849 to 1889 . *31 initial tokens*

1) Tokenize the sentence

2) Add special tokens to each sentence: CLS - to the start & SEP - to the end:

['[CLS]', 'the', 'gold', 'dollar', 'or', 'gold', 'one', '@', '-', '@', 'dollar', 'piece', 'was', 'a', 'coin', 'struck', 'as', 'a', 'regular', 'issue', 'by', 'the', 'united', 'states', 'bureau', 'of', 'the', 'mint', 'from', '1849', 'to', '1889', '[SEP]'] *(33 tokens)*

3) Map tokens to their IDs in the pretrained BERT dictionary data (101 for CLS, 102 for SEP):

[101, 1996, 2751, 7922, 2030, 2751, 2028, 1030, 1011, 1030, 7922, 3538, 2001, 1037, 9226, 4930, 2004, 1037, 3180, 3277, 2011, 1996, 2142, 2163, 4879, 1997, 1996, 12927, 2013, 9037, 2000, 6478, 102]

4) Pad (add the zeros at the end of the sequence to make the samples in the same size) or truncate(drop the last words in the sentence) the sentence to `max_length` (up to 512)

[101, 1996, 2751, 7922, 2030, 2751, 2028, 1030, 1011, 1030, 7922, 3538, 2001, 1037, 9226, 4930, 2004, 1037, 3180, 3277, 2011, 1996, 2142, 2163, 4879, 1997, 1996, 12927, 2013, 9037, 2000, 6478, 102, 0, 0, 0, 0, 0, 0]

(if we had 33 tokens while max_length was e.g. 40, we would add 7 [PAD]s to fill in the gaps)

5) Create attention masks for [PAD] tokens (to explicitly differentiate real tokens from padding tokens), so that model is instructed which tokens it should be attended to, and which should not.

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

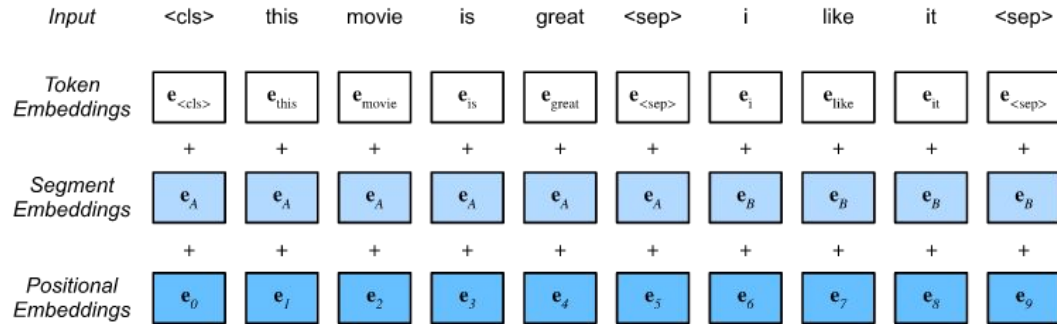
(we have 33 meaningful tokens and 7 [PAD]s here)

Pre-processing data (2)

*This is happening for the first BERT layer!

For words to be processed by BERT, they need some form of numeric representation, so each token is embedded into a vector (embedding).

BERT paper: 'For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings'.

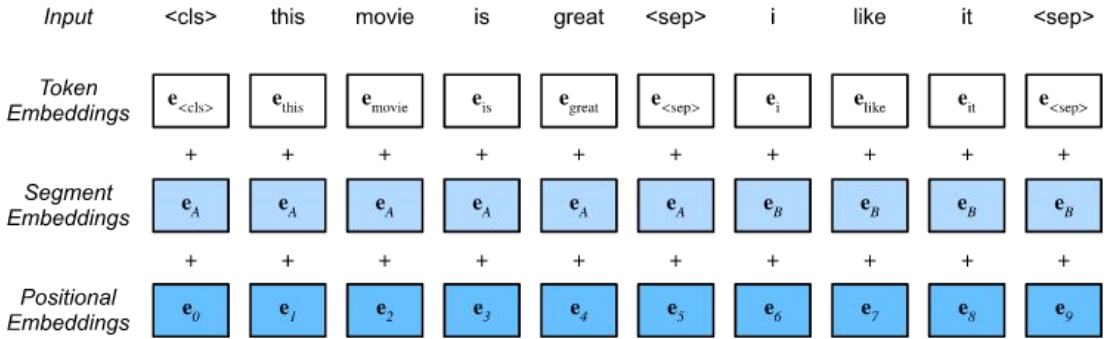


BERT's knowledge about language is based on the embeddings that it was pre-trained with, so users cannot use their own embeddings. These embeddings become BERT input sequence which may include either one text sequence or two text sequences.

The Token Embeddings layer will convert each token (corresponding to the original word) into a 768-dimensional vector representation. 768 is the final embedding dimension from the pre-trained base BERT architecture.

The Segment Embeddings 768 dimensional vector just maps the token to the 1st or 2nd text sequences within BERT input sequence. If BERT input sequence contains only one text sequences index 0 is assigned to all tokens in the input.

Positional embeddings are 768 dimensional learned vectors for every possible position between 0 and 512-1, they allow BERT to have some information about the order of the input, otherwise output would be permutation-invariant.



These 3 representations are then summed element-wise to produce a single representation with shape (1, n, 768).

Model (1)

The original paper provides two [BERT structures](#):

- **BERTBase**, consists of 12-layer bidirectional Transformer encoder blocks with hidden size 768, 12 self-attention heads, 110M parameters;
- BERT-Large includes 24- layer bidirectional Transformer encoder blocks with hidden size 1024, 16 self-attention heads, 340M parameters.

The weights (embeddings) are trained on BooksCorpus (11,038 books) and the English Wikipedia.

Model (2)

BERT enables **Transfer Learning = Pre-Training + Fine-Tuning**

A technique of adding a small task-specific component to the end of a large pre-trained model, and fine-tuning the result

Pre-Training: Masked Language Model (MLM) and Next Sentence Prediction (NSP).

1. MLM - Predict the crossed out word. (Answer is “simple”).
2. NSP - Was sentence B found immediately after sentence A, or from somewhere else? (Answer is that they are consecutive).

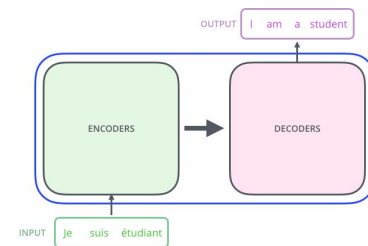
“BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks...”

The main part of BERT is a large 12-layer neural network that processes text. MLM and NSP each add a small, single-layer classifier to the output of BERT to perform their respective tasks.

Also you can replace this final classifier with your own task-specific model (e.g. sentiment classification)

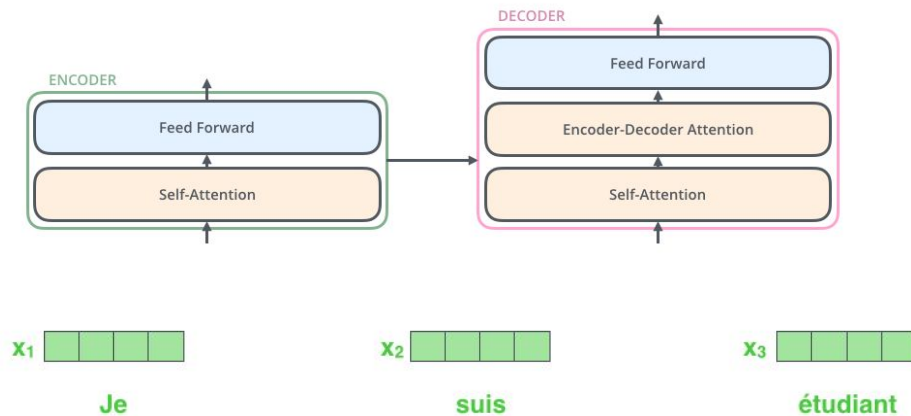
Transformers

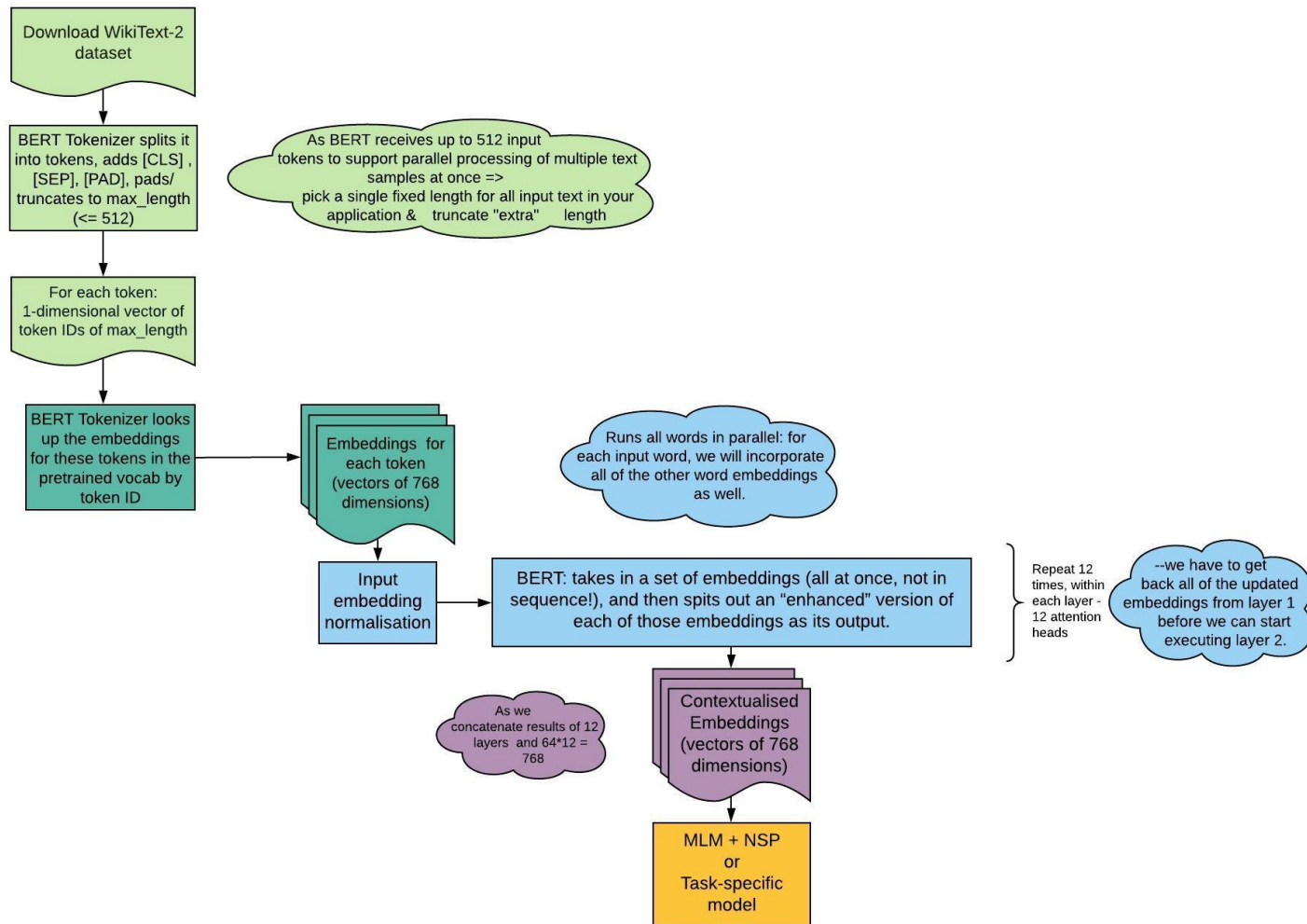
Authors of BERT did not come up with its architecture - it is taken directly from an earlier model called the **Transformer**.



What made the difference was an architecture introduced by Google based on a self-attention mechanism that is believed to be well suited for language understanding.

BERT inherited the biggest benefit of Transformer - parallelisation.





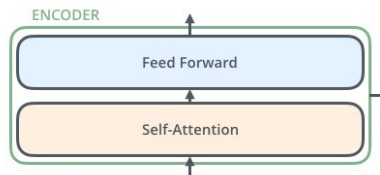
BERT layer = Encoder

BERT has fixed input and output dimensions - it has one input vector, and one output vector (both 768 dimensional).

Regardless of the number of input vectors, averaging them will always produce one output vector. This “weighted average” step occurs at the end of the “**Self-Attention**” mechanism. ”.

Neural network architecture introduced by Google based on a self-attention mechanism that is believed to be well suited for language understanding.

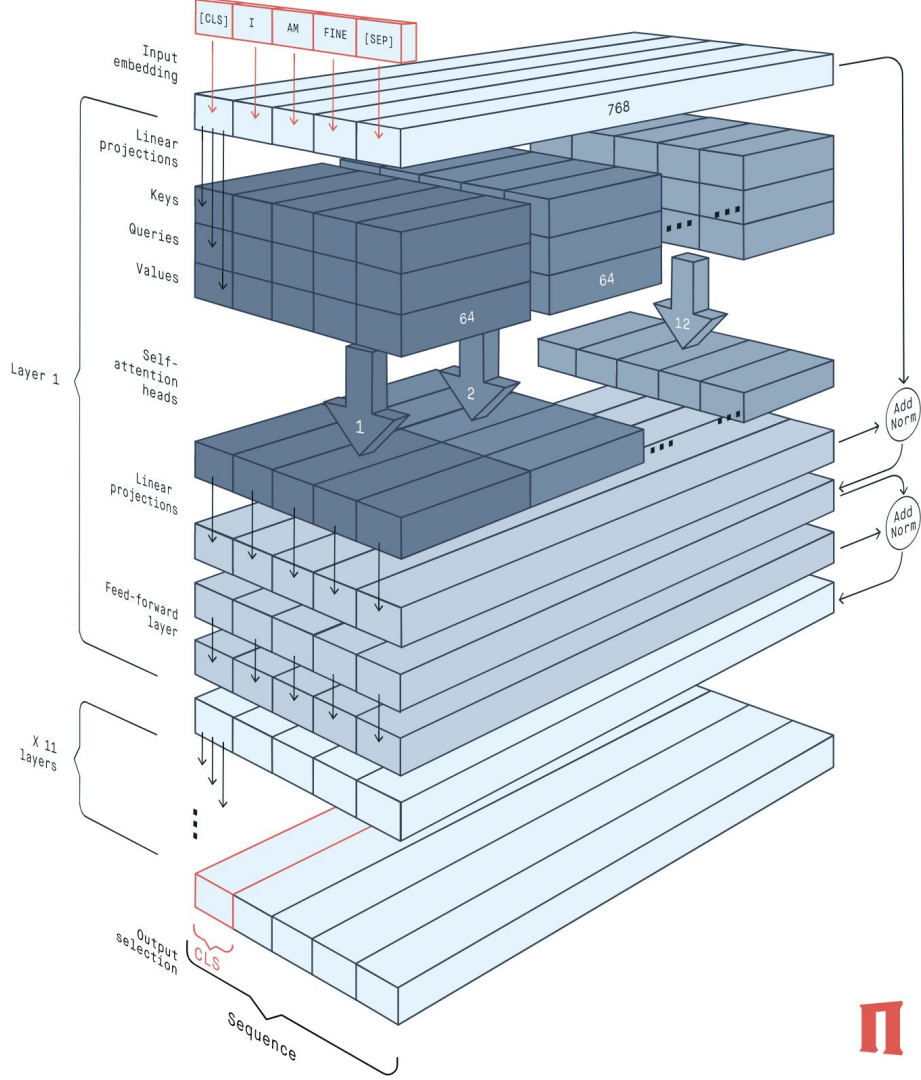
The biggest benefit - parallelisation.



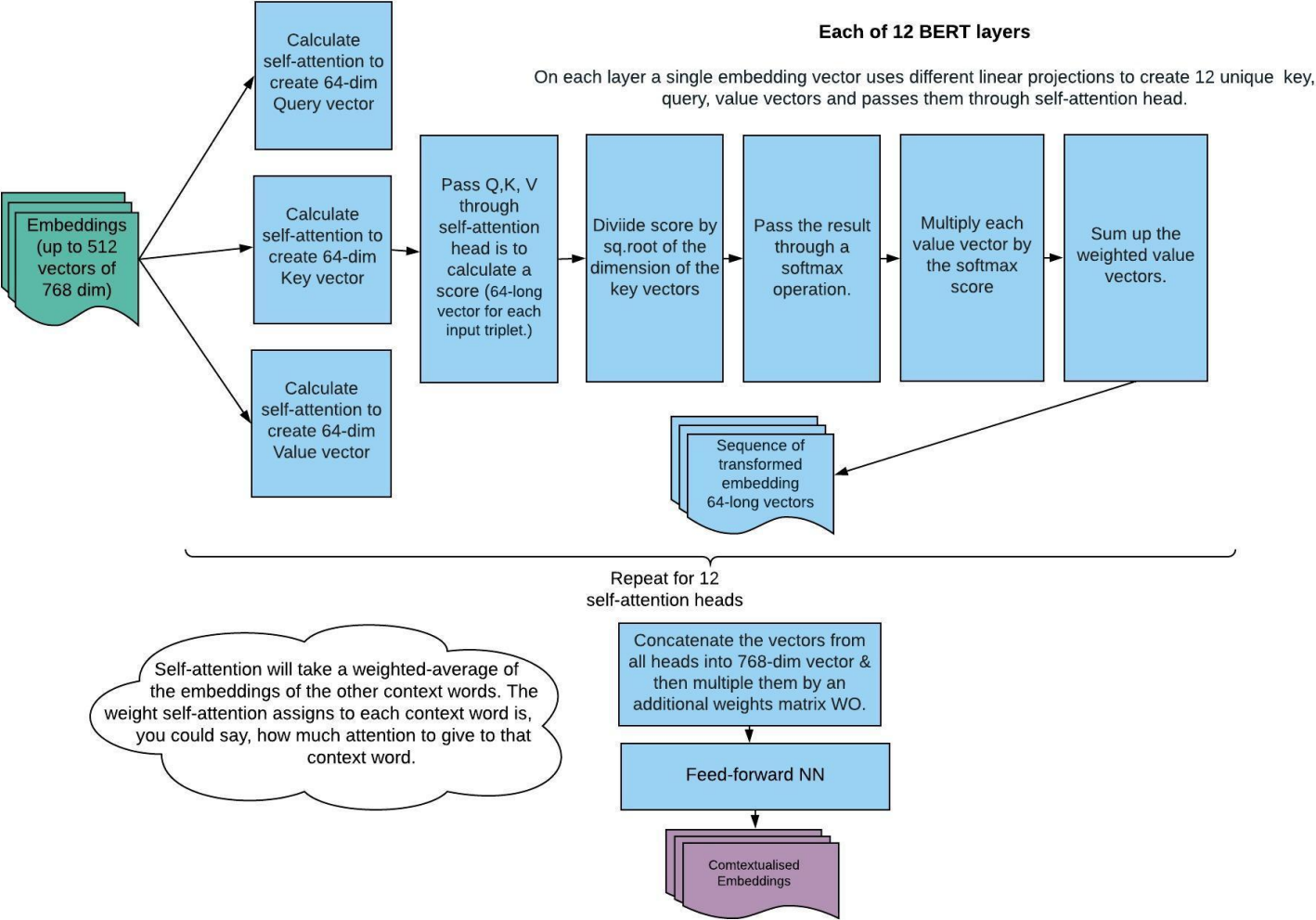
x_1 Je

x_2 suis

x_3 étudiant



BERT layer



Self-attention

Great animation:

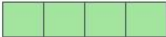
<https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/bert-encoder>

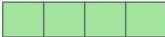
Input

Thinking

Machines

Embedding

x_1 

x_2 

Queries

q_1 

q_2 

Keys

k_1 

k_2 

Values

v_1 

v_2 

Score

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

Divide by 8 ($\sqrt{d_k}$)

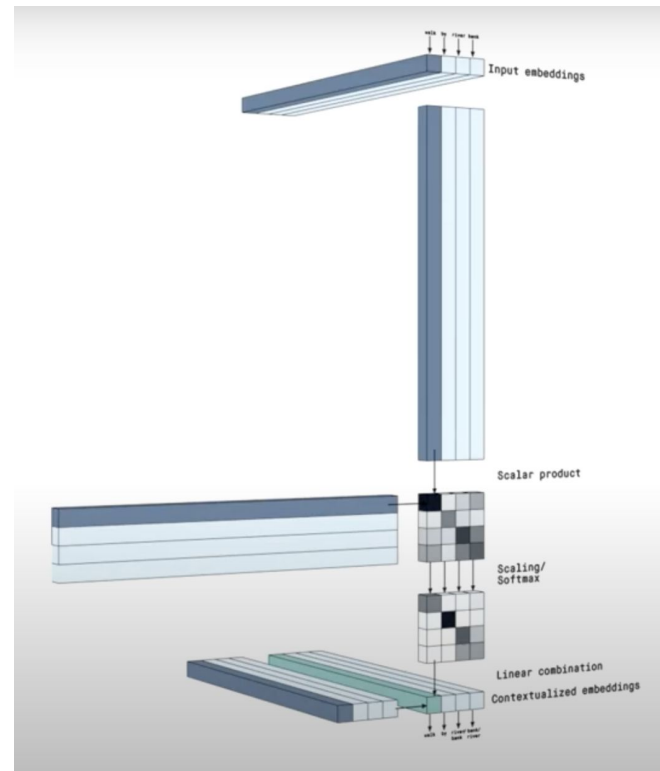
14

12

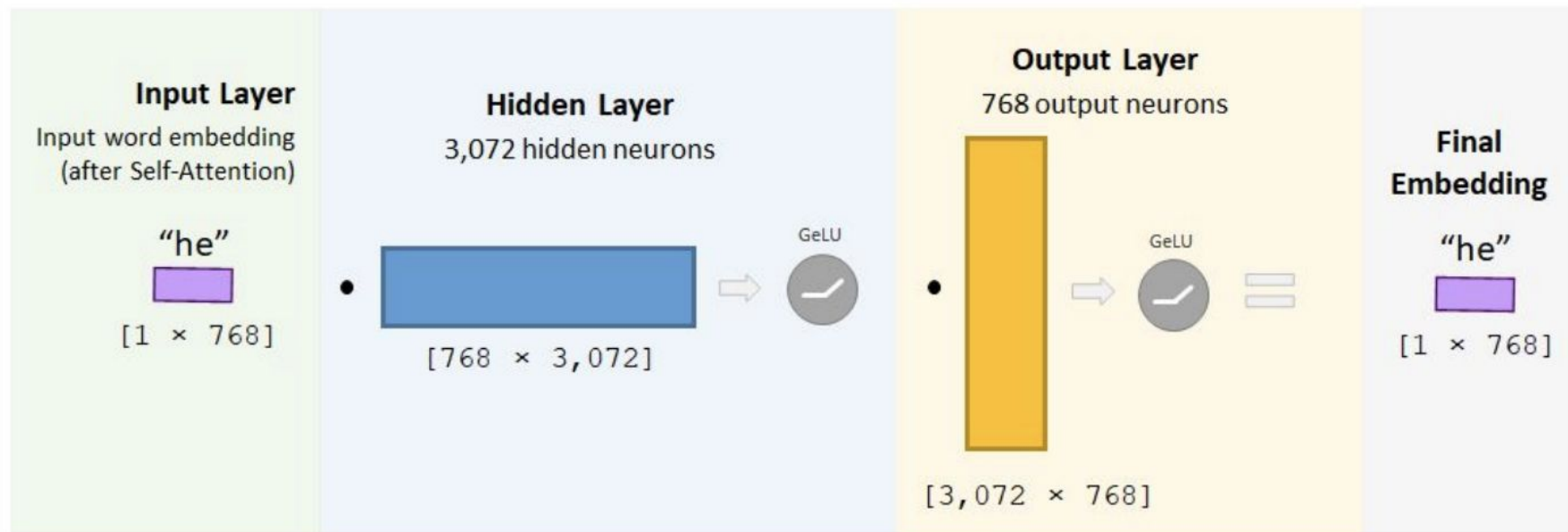
Softmax

0.88

0.12



Feed Forward Neural Network



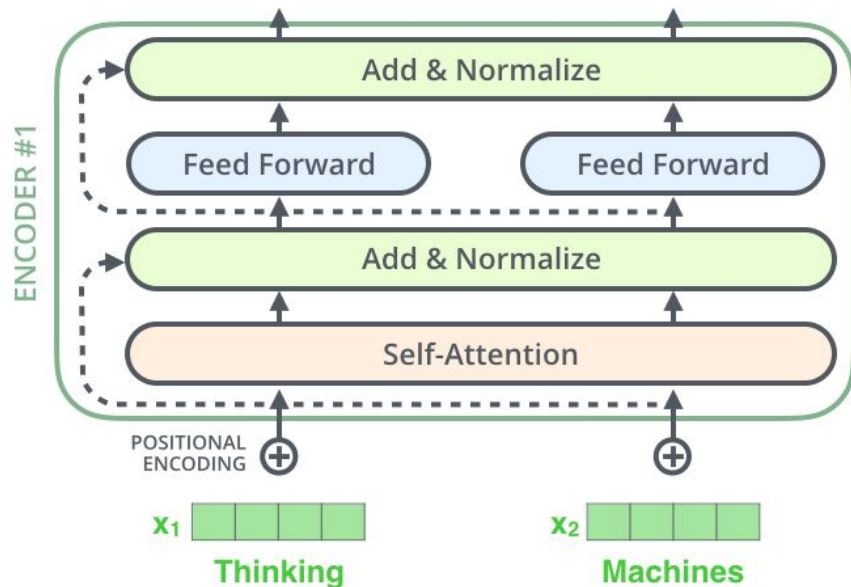
Layer normalisation

This is expressed by the standard formula for calculating a weighted average, shown below. In our context, the variables have the following meanings:

- n is the number of the words in the sentence.
- i is the position of a word in the sentence.
- x_i is the embedding for the word at position i .
- w_i is the weight value (between 0 - 1.0) given to the word at position i .

$$\bar{x} = \frac{\sum_{i=1}^n w_i x_i}{\sum_{i=1}^n w_i}$$

The result, \bar{x} , is the enhanced embedding for the current input word!



To enable deep bi-directional representations, BERT “introduced” **Masked Language Modelling (MLM)**

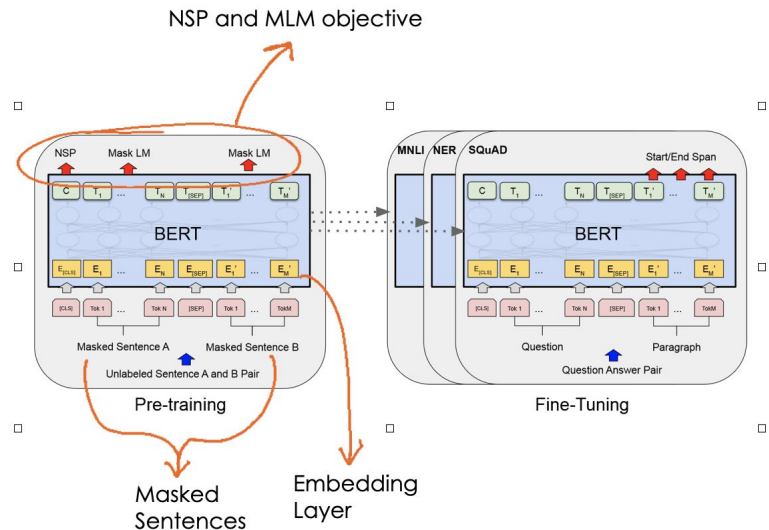
- Before tokens are fed into the model, 15% of them are masked with a [MASK] tag.
- The model’s objective is to predict the masked tokens,
- There’s a discrepancy however – the [MASK] token will not be present during fine-tuning.
 - To mitigate this, the masked words are not always replaced with a [MASK] token
 - 10% of the time, the masked token will actually be a random word from our vocabulary
 - 10% of the time, the masked token will actually be the original word
 - 80% of the time, the masked token will be [MASK]

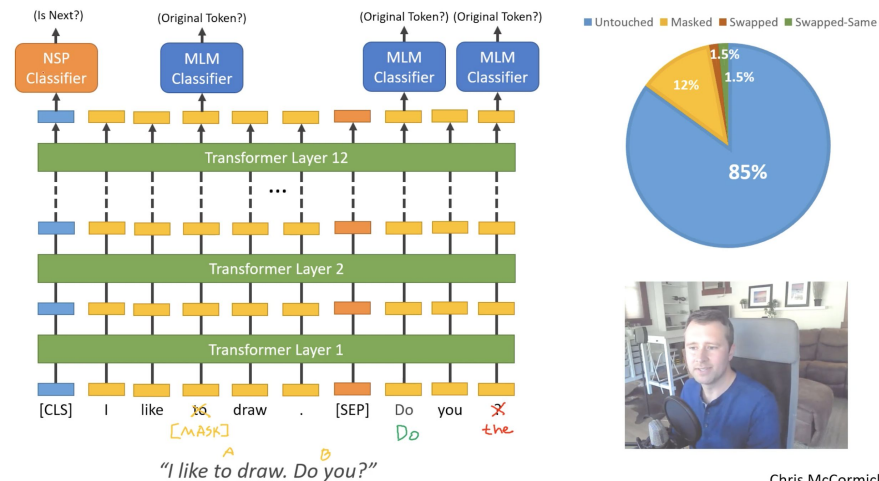
In order to train a model that understands sentence relationships, BERT pre-train for a binarized **next sentence prediction (NSP)** task that can be trivially generated from any monolingual corpus. Specifically, when choosing the sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext).

In MLM and NSP, you can give BERT the entire input sequence at once, and BERT gets to factor-in all of the words in the text into its predictions. This is the significance behind the ‘B’ in BERT’s name--it stands for “Bidirectional”, as opposed to left-to-right.

Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext





Token Substitutions

this is not creative . those are the dictionary definitions of the
 this is not creative . those are [MASK] dictionary definitions of the

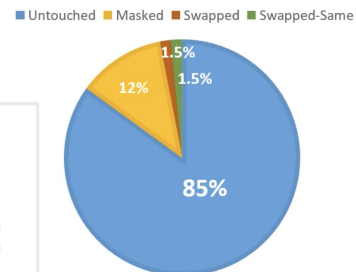
terms insurance and en ##sur ##ance as properly applied to destruction
 terms [MASK] and en ##sur ##ance as [MASK] applied to destruction

. if you don ' t understand that , fine , legitimate criticism
 . if you don [MASK] [MASK] understand [MASK] , fine want [MASK] criticism

, i ' ll write up three man cell and bounty hunter and then
 , [MASK] ' ll write up three man [MASK] and [MASK] hunter and then

it will be easy to understand why ensured and ins ##ured are different
 it will be easy understand why ensured and ins ##ured are different

- and why both differ from assured . the sentence you quote is absolutely
 - and why [MASK] differ from assured . the [MASK] you quote is absolutely



Fine-tuning

Authors recommend only 2-4 epochs of training for fine-tuning BERT on a specific NLP task (compared to the hundreds of GPU hours needed to train the original BERT model or an LSTM from scratch!).

Implementation plan

Work in progress

I am going to use the following notebooks as a benchmark:

- 1) BERT pre-training by [Dive into Deep Learning](https://colab.research.google.com/github/d2l-ai/d2l-en-colab/blob/master/chapter_natural-language-processing-pretraining/bert-pretraining.ipynb#scrollTo=2kxIKa3HuOzE):
https://colab.research.google.com/github/d2l-ai/d2l-en-colab/blob/master/chapter_natural-language-processing-pretraining/bert-pretraining.ipynb#scrollTo=2kxIKa3HuOzE
- 2) BERT Fine-Tuning Tutorial with PyTorch By Chris McCormick and Nick Ryan:
<https://colab.research.google.com/drive/1pTuQhug6DhI9XaIKB0zUGf4FIIdYFIpcX>
- 3) BERT End to End (Fine-tuning + Predicting) with Cloud TPU: Sentence and Sentence-Pair Classification Tasks by Google:
https://colab.research.google.com/github/tensorflow/tpu/blob/master/tools/colab/bert_finetuning_with_cloud_tpus.ipynb
- 4) BERT Fine-Tuning Sentence Classification.ipynb By The Corpus of Linguistic Acceptability:
https://colab.research.google.com/drive/1ywsvwO6thOVOrfagjifuxEf6xVRxbUNO#scrollTo=BJR6t_gCQe_x

Resources (1)

- 1) <https://www.youtube.com/watch?v=g253pl3Kp9o> - Understanding BERT: Step by Step by [Yan Xu](#)
- 2) [Chris McCormick](#) eBook: The Inner Workings of BERT
- 3) <http://jalammar.github.io/illustrated-transformer/>
- 4) <https://www.youtube.com/watch?v=rBCqOTefxvg> Attention is all you need; Attentional Neural Network Models | Łukasz Kaiser | Masterclass
- 5) <https://www.youtube.com/watch?v=u91645MFytY> - interview by Kaggle with BERT's primary author, Jacob Devlin
- 6) <https://arxiv.org/abs/1810.04805> - [Submitted on 11 Oct 2018 (v1), last revised 24 May 2019 (this version, v2)]BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding - Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova
- 7) https://d2l.ai/chapter_natural-language-processing-applications/natural-language-inference-bert.html

Resources (2)

- 9) https://www.alibabacloud.com/blog/natural-language-intelligence-building-a-language-bridge-for-business_596417
- 10) <https://nlp.stanford.edu/seminar/details/jdevlin.pdf>
- 11) https://nlp.stanford.edu/~johnhew//structural-probe.html?utm_source=quora&utm_medium=referral#the-structural-probe
- 12) BERT guide + visualisation
<https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/bert-encoder>