AI CORE course capstone project

BERT implementation

by Tanya Z

What is BERT?

BERT is a method of pretraining language representations and named after it 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.

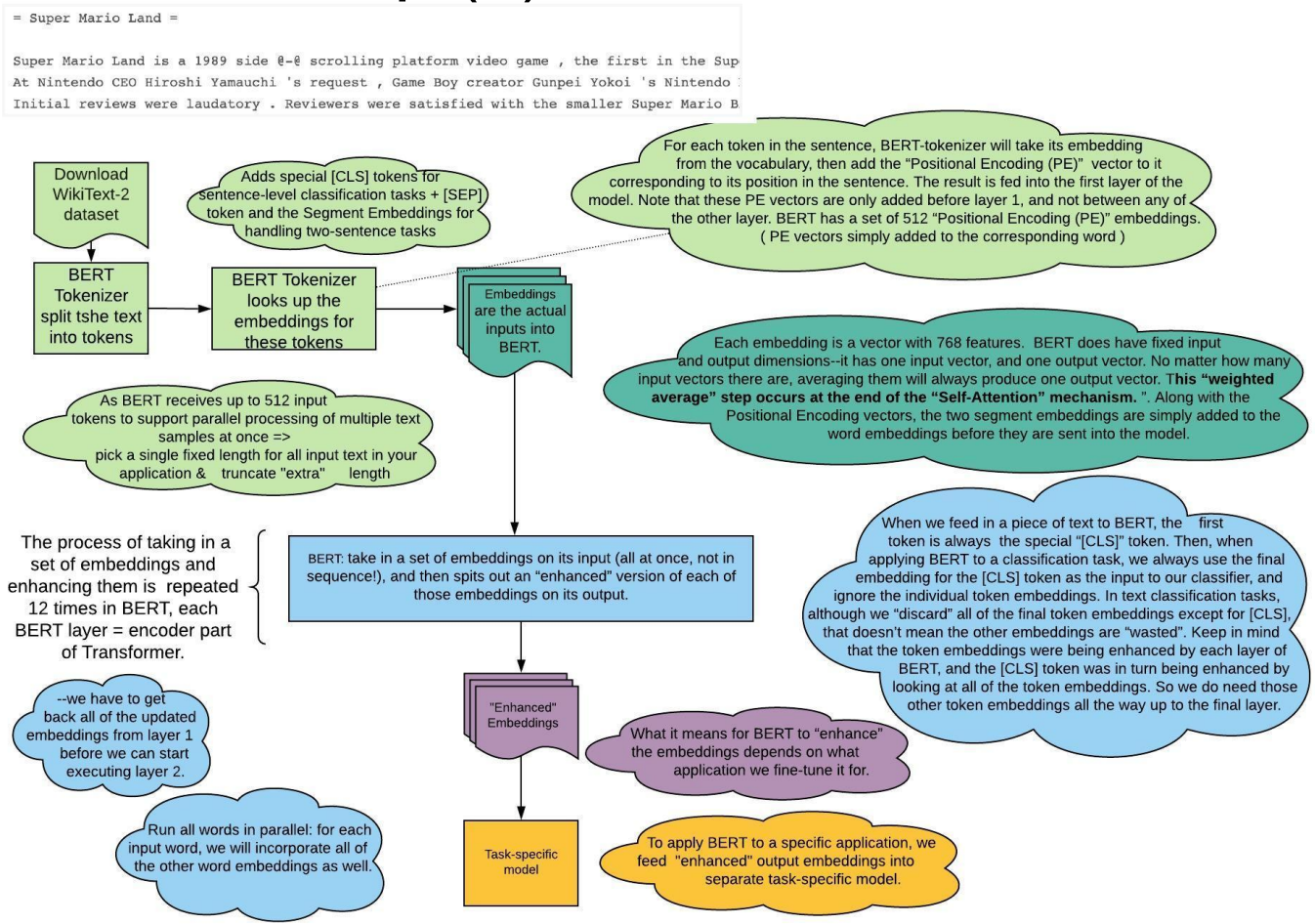
Problem description

To be described

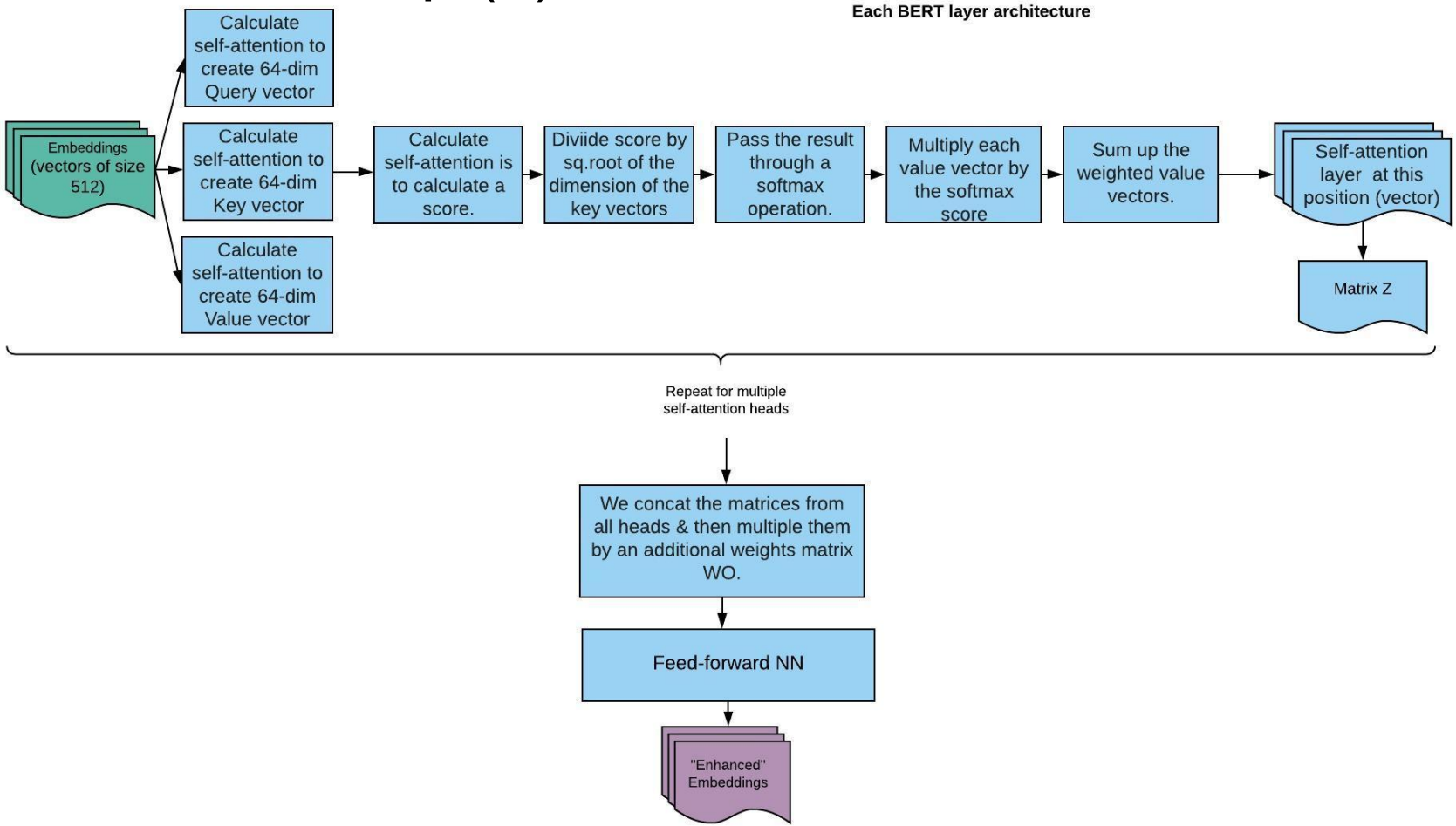
Implementation plan

To be described

Implementation map (1)



Implementation map (2)

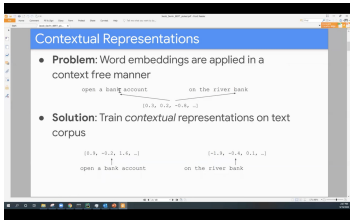


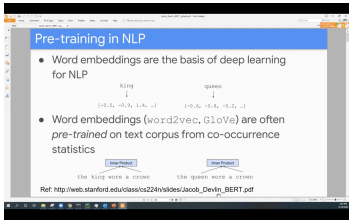
Dataset

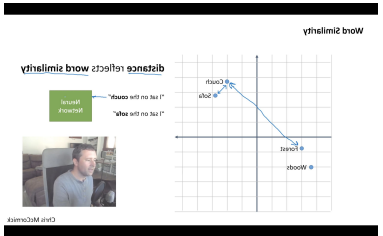
WikiText-2 Dataset- a small preprocessed version of Wikipedia, containing 200M tokens (Merity et al., 2016)
https://blog.einstein.ai/the-wikitext-long-term-dependency-language-modeling-dataset/

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

Download dataset



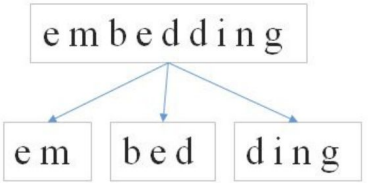




Pre-processing data (1)

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 up to up to 512 input tokens one needs to set a persistent 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 down[2]:
- 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 .

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

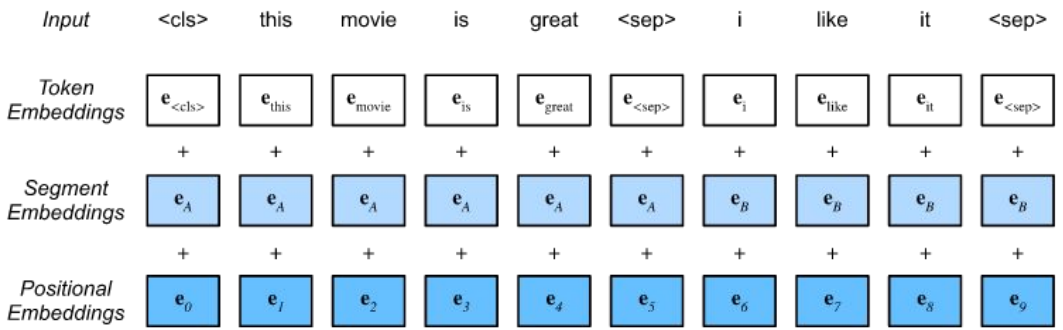
   ['[CLS]', 'the', 'gold', 'dollar', 'or', 'gold', 'one', '@', '-', '@', 'dollar', 'piece', 'was', 'a', 'coin', 'struck', 'as', 'a', 'regular', 'issue', 'by', 'the', 'united', 'states', 'bureau', 'of', 'the', 'mint', 'from', '1849', 'to', '1889', '[SEP]']

3) Map tokens to their IDs: *['[CLS]', 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, '[SEP]']*

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)
5) Create attention masks for [PAD] tokens (to explicitly differentiate real tokens from padding tokens)

Pre-processing data (2)

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’.



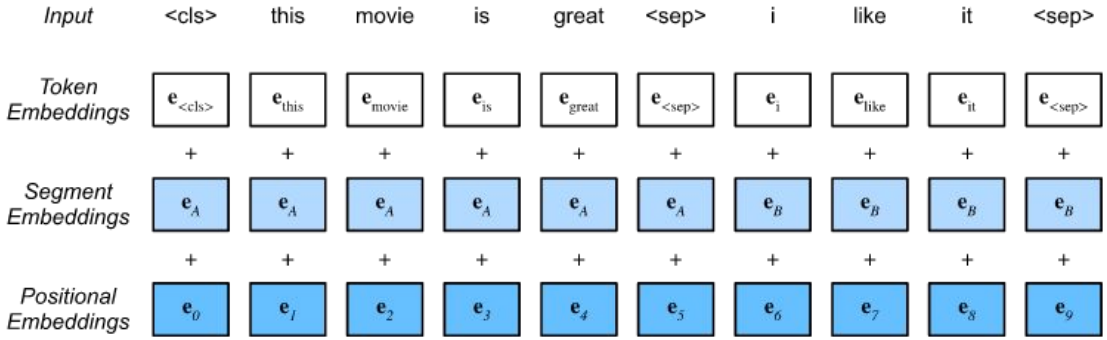
BERT’s knowledge about language is based on the embeddings that it was pre-trained with, so users cannot use your own embeddings.

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 this input 1.

Positional embeddings are learned 768 dimensional 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) for tokenized input sequence of length.

Model

Transformers:
The Transformer was designed to perform Machine Translation (translating text from one language to another). It consisted of two major parts--an “Encoder” and a “Decoder”. The Encoder would look at the whole input sentence in one language and encode it into embeddings, and then the Decoder would use those embeddings to generate the output sentence in a different language.BERT is literally just the Encoder portion of that model, scaled up a bit, and trained on MLM and NSP instead of translation.The result is an incredibly powerful Encoder (BERT), but with no Decoder portion. The loss of the Decoder means that BERT is not applicable to tasks where we want to generate text (such as translation).

Encoder¶

We have three layers in the encoder: an embedding layer (with dropout), a RNN layer, and a linear layer. As we have known from the word representation session, we can apply a embedding layer and distributed word representation is trained jointly with the model.If we want to have a bidirectional encoder to encode both forward and backward contexts in the input, the hidden dimension of the RNN layer is doubled. Therefore, the linear layer is here to keep the same dimensionality between the encoder output and decoder input.

Decoder

The decoder has four layers: an embedding layer (with dropout), a unidirectional RNN layer and two linear layers. The decoder is always unidirectional in that we only generate the outputs from left to right.

Teacher forcing

Teacher forcing is used in training to reduce error propagation and accelerate training. During training, the next input to the decoder can be either the ground truth token or the output by the decoder, determined by a teacher_force_ratio.

Attention

https://arxiv.org/pdf/1409.0473.pdf https://arxiv.org/pdf/1508.04025.pdf The attention mechanism allows the decoder to search the source sentence and concentrate on the more relevant information (based on previous decoder hidden state), at each decoding step. At the heart of the Transformer is self-attention. Self-attention is a mechanism that allows each input in a sequence to look at the whole sequence to compute a representation of the sequence...what? Ok. Let's look at the following sentence: *the animal didn't cross the street because it was too tired.* What does the it refer to in this sentence? The street or the animal? This is trivial for us as humans to answer but not for a machine. Wouldn't it be nice if we could have some way of the computer understanding what it referred to?

BERT to use how to pre-train encoders from transformer???

This technique of adding a small task-specifc component to the end of a large pre-trained model, and fine-tuning the result, is known as **Transfer Learning** = Pre-Training + Fine-Tuning

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

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

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

BERT was trained to perform these two tasks purely as a way to force it to develop a sophisticated understanding of language. Once the pre-training is done, we actually discard the (tiny) portion of the model that's specific to these tasks, and replace it with something more useful Benefits: We don't need any labeled data. We can simply train with raw text, such as all of Wikipedia, or a crawl of the interne + The tasks are challenging, requiring BERT to develop strong language understanding skills.

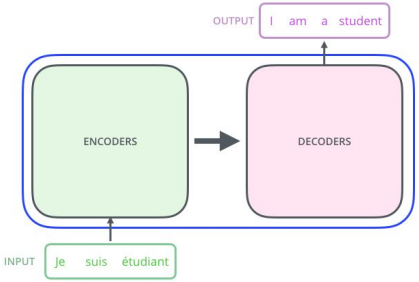
The main portion of BERT is a very large 12-layer neural network that processes text. MLM and NSP each add a small, single-layer classifier to the output of BERT to perform their respective tasks. **The magic of BERT is that you can replace this final classifier with your own task-specific model, and leverage all of BERT's knowledge!**

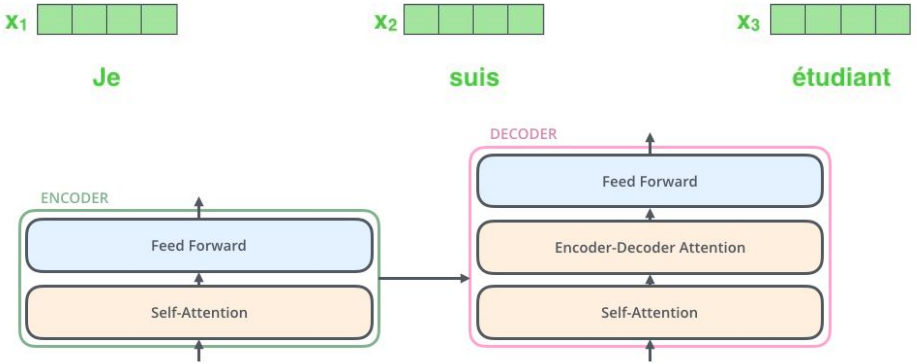
For example, if you want to apply BERT to a particular text classification task, you would take the pre-trained BERT model, add an untrained layer of neurons on the end, and train the new, combined model on your own dataset and task. This **final training step is referred to as "fine-tuning",** because the amount of training required to adapt BERT to your own task is very small compared to what it took Google to pre-train BERT.

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

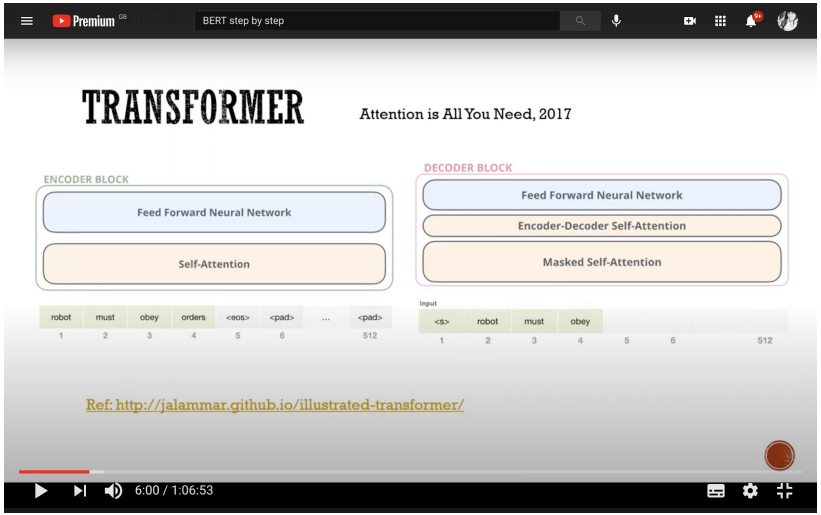
Each position outputs a vector of size *hidden_size* (768 in BERT Base).

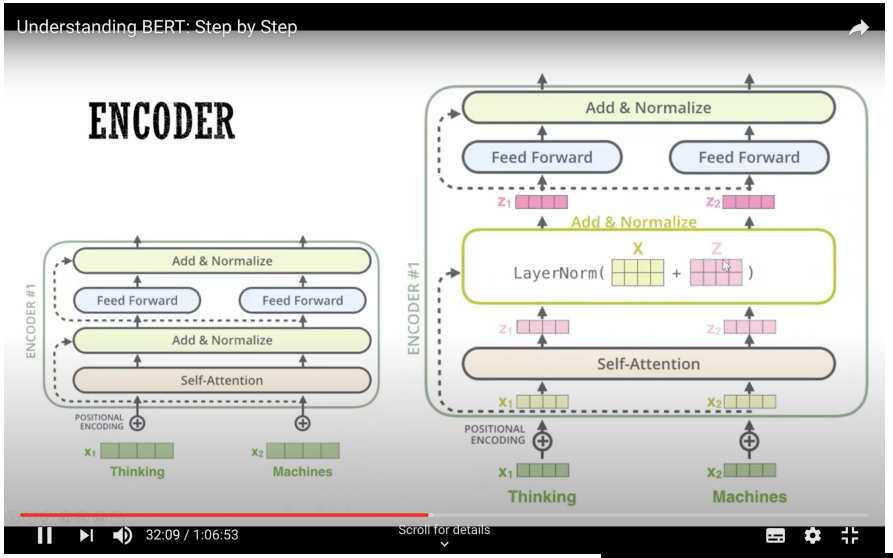
The biggest benefit, however, comes from how The Transformer lends itself to parallelization.
The embedding only happens in the bottom-most encoder.
Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

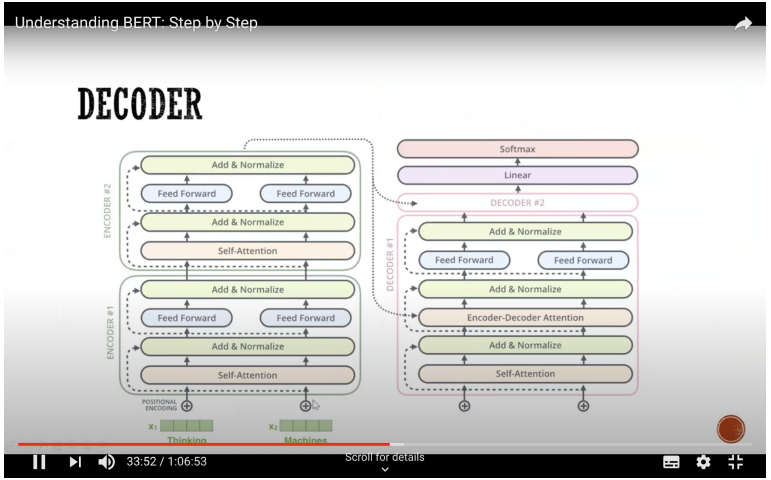


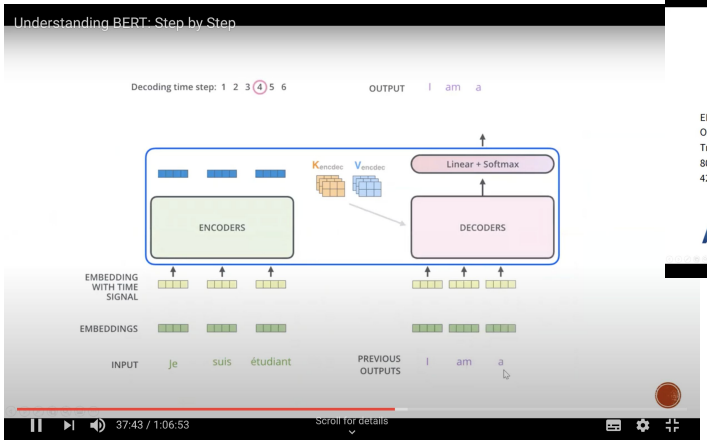


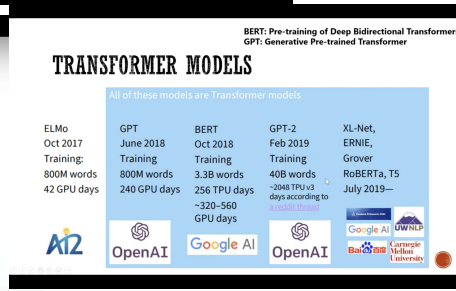
When producing the enhanced embedding for the word “he”, self-attention will take a weighted-average of the embeddings of the other context words. The weight self-attention assigns to each context word is, you could say, how much attention to give to that context word.

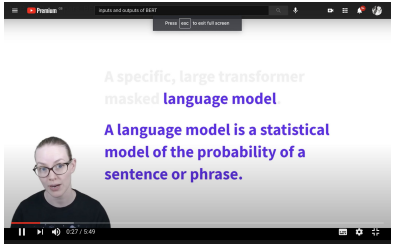

BERT step by step
TRANSFORMER
Attention is All You Need, 2017
ENCODER BLOCK
Feed Forward Neural Network
Self-Attention
DECODER BLOCK
Feed Forward Neural Network
Encoder-Decoder Self-Attention
Masked Self-Attention
robot must obey orders <eos> <pad> ... <pad>
Input
<s> robot must obey
Ref: http://jalammar.github.io/illustrated-transformer/
6:00 / 1:06:53

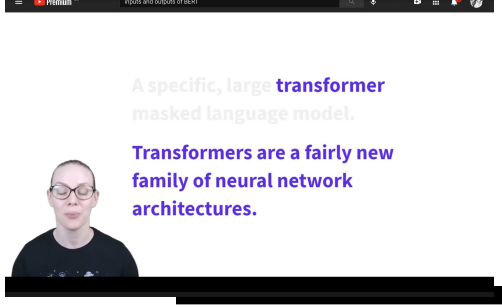

Understanding BERT: Step by Step
ENCODER
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
POSITIONAL ENCODING
Thinking
Machines
LayerNorm(X + Z)
32:09 / 1:06:53
Scroll for details

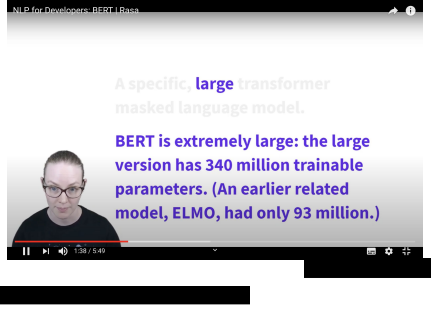

Understanding BERT: Step by Step
DECODER
Softmax
Linear
DECODER #2
Encoder-Decoder Attention
ENCODER #2
ENCODER #1
33:52 / 1:06:53
Scroll for details

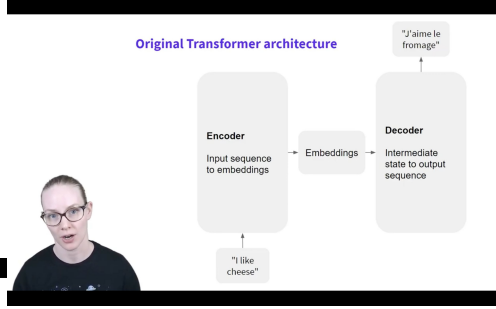

Understanding BERT: Step by Step
Decoding time step: 1 2 3 4 5 6
OUTPUT I am a
ENCODERS
DECODERS
Linear + Softmax
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT je suis étudiant
PREVIOUS OUTPUTS I am a
37:43 / 1:06:53
Scroll for details

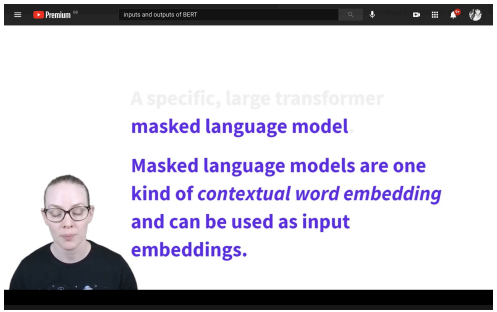

BERT: Pre-training of Deep Bidirectional Transformer
GPT: Generative Pre-trained Transformer
TRANSFORMER MODELS
All of these models are Transformer models
ELMo Oct 2017 Training: 800M words 42 GPU days
GPT June 2018 Training 800M words 240 GPU days
BERT Oct 2018 Training 3.3B words 256 TPU days ~320–560 GPU days
GPT-2 Feb 2019 Training 40B words ~2048 TPU v3 days according to a reddit thread
XL-Net, ERNIE, Grover RoBERTa, T5 July 2019—
OpenAI
Google AI
OpenAI



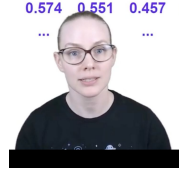




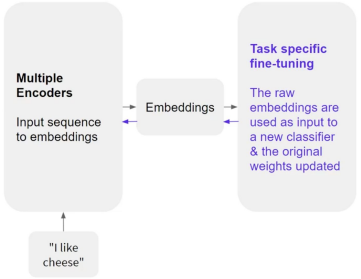




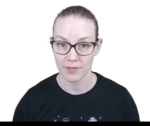
[CLS]	This	is	an	input	string	[SEP]	So	is	this	[SEP]
0.364	0.421	0.907	0.029	0.038	0.097	0.623	0.644	0.582	0.743	0.007
0.439	0.844	0.503	0.129	0.058	0.78	0.983	0.099	0.413	0.999	0.175
0.315	0.969	0.842	0.667	0.199	0.114	0.353	0.212	0.563	0.623	0.79
0.433	0.595	0.739	0.432	0.132	0.03	0.832	0.409	0.713	0.825	0.17
0.574	0.551	0.457	0.516	0.22	0.166	0.162	0.667	0.941	0.778	0.357
...	...	...	...	...	...	...	...	...	...	...



The BERT architecture uses a stack of either 12 (BASE) or 24 (LARGE) Encoders

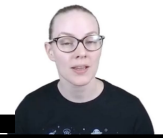


How is it used?



- To convert text input into a numeric representation (i.e. to replace word embeddings)
- As a general-purpose pre-trained model that's fine tuned for specific tasks

What are the benefits?

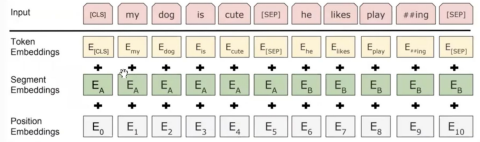


- Transferable model: can be used as input to smaller, task specific models
- With successful fine tuning, can have very good accuracy
- Pretrained BERT models available in 100+ languages

Common Errors

- Not using WordPiece tokenizer
- While fine tuning, some runs produce degenerate results (seems to be task specific & not clear beforehand which tasks this will happen to)
- Training new models instead of using pre-trained ones

Input Representation

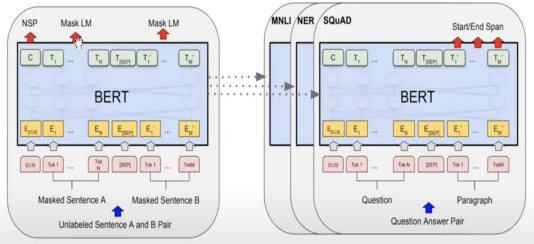


- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
- Single sequence is much more efficient.

Model Details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

Fine-Tuning Procedure



Masked LM in BERT

- Problem: Mask token never seen at fine-tuning
- Solution: 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
- 80% of the time, replace with [MASK]
  went to the store → went to the [MASK]
- 10% of the time, replace random word
  went to the store → went to the running
- 10% of the time, keep same
  store → went to the store

Next Sentence Prediction

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The
Sentence B = Pengu
Label = NotNextSen

Masked LM in BERT

- **Solution**: Mask out *k*% of the input words, and then predict the masked words
  - We always use *k* = 15%

- Too little masking: Too expensive to train
- Too much masking: Not enough context

GLUE Results

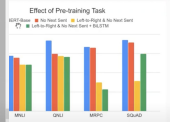
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT$_{BASE}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{LARGE}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

MultiNLI
Premise: Hills and mountains are especially sanctified in Jainism.
Hypothesis: Jainism hates nature.
Label: Contradiction

CoLA
Sentence: The wagon rumbled down the road.
Label: Acceptable

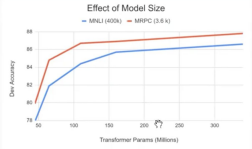
Sentence: The car honked down the road.
Label: Unacceptable

aining Task



mpared to left-to-right LM) is very important on
xt Sentence Prediction is important on other tasks.
odel does very poorly on word-level task (SQuAD),
mitigated by BiLSTM

fect of Model Size



- Big models help *a lot*
- Going from 110M -> 340M params helps even on datasets with 3,600 labeled examples
- Improvements have *not* asymptoted

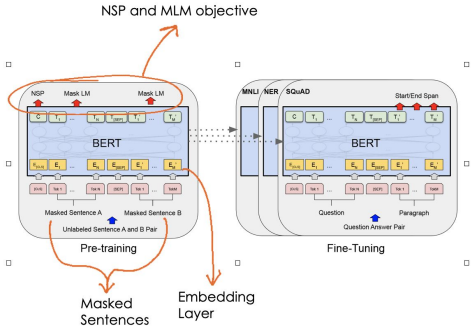
Open Source Release

- One reason for BERT's success was the open source release
  - Minimal release (not part of a larger codebase)
  - No dependencies but TensorFlow (or PyTorch)
  - Abstracted so people could including a single file to use model
  - End-to-end push-button examples to train SOTA models
  - Thorough README
  - Idiomatic code
  - Well-documented code
  - Good support (for the first few months)

- 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]

BERT is a multi-layer bidirectional Transformer encoder (Devlin et al., 2018). The original paper provides two BERT structures: BERTBase, consists of 12-layer bidirectional Transformer encoder block with hidden size 768 and 12 self-attention heads; BERT-Large includes 24- layer bidirectional Transformer encoder blocks with hidden size 1024 and 16 self-attention heads. The weights are trained on BooksCorpus and the English Wikipedia. Unless stated otherwise, we mean BERT Base when mentioning BERT.

Fig. 14.8.2



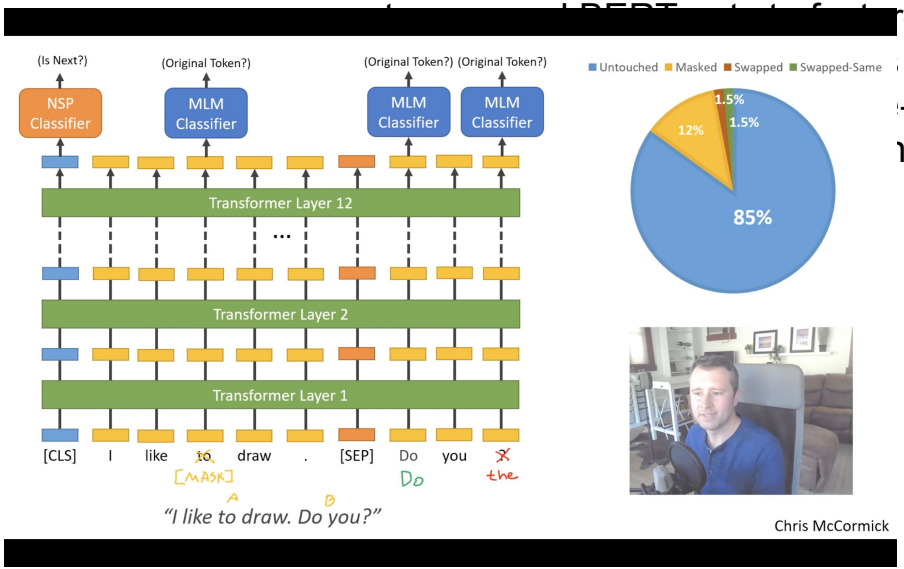
Architecture

- Apart from output layers, architecture between pre-training and fine-tuning is the same
- Special [CLS] token which is given as first input
- Special [SEP] token which separates the two input sentences

0) We apply magic on data: mask 15% of the text, highlight another 15%, leave other text as is

1) We pass all our data through 12 transformer layers
2) The masked data goes into MLM classifier which predicts what word most probably was masked
3)

In MLM and NSP, you can give BERT the entire input -in all of the the --it stands nt.



The dot product of two vectors is the sum of the products of elements with regards to position. The first element of the first vector is multiplied by the first element of the second vector and so on. The sum of these products is the dot product

```
import numpy as np
```

```
a = np.array([1,2,3])
b = np.array([2,4,6])
```

```
np.dot(a,b)
```

```
28
```

```
a[0]*b[0] + a[1]*b[1] + a[2]*b[2]
```

```
28
```

multiplication of two matrices involves many dot product operations of vectors. A 2x2 matrix has 2 rows and 2 columns.

A

```
array([[4, 2],
       [0, 3]])
```

B

```
array([[4, 0],
       [1, 4]])
```

So the resulting matrix, C, will have a (4*4) + (2*1) at the first row and first column.
C[0,0] = 18.

```
C = np.dot(A,B)
C
```

```
array([[18,  8],
       [ 3, 12]])
```

1)

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!).

Resources (1)

1) https://www.youtube.com/watch?v=g253pI3Kp9o - 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://colab.research.google.com/drive/1ywsvwO6thOVOrfagjjfuxEf6xVRxbUNO#scrollTo=BJR6t_gCQe_x
8) https://d2l.ai/chapter_natural-language-processing-applications/natural-language-inference-bert.html

Resources (2)

9)