

Machine Learning

Lecture 09: Transformer and BERT

Nevin L. Zhang
lzhang@cse.ust.hk

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on internet resources and references listed at the end.

Outline

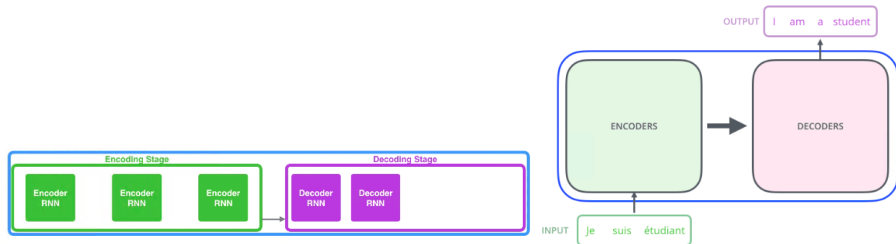
1 Transformer

- Self-Attention Layer
- The Encoder
- The Decoder
- Vision Transformer (ViT)

2 BERT

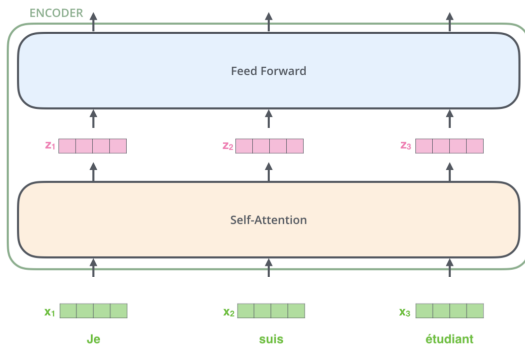
- Overview
- Pre-training BERT
- Fine-tuning BERT

Two Seq2Seq Models



- **RNN** is sequential. It precludes parallelization within training examples
http://jalammar.github.io/images/seq2seq_6.mp4
- **Transformer** allows significantly more parallelization:
http://jalammar.github.io/images/t/transformer_decoding_1.gif
 - Based solely on attention, dispensing with recurrence.
 - Requires less time to train and achieves better results than Seq2Seq.

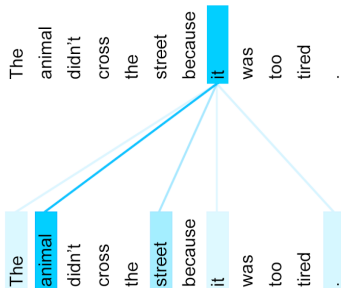
Self-Attention Layer



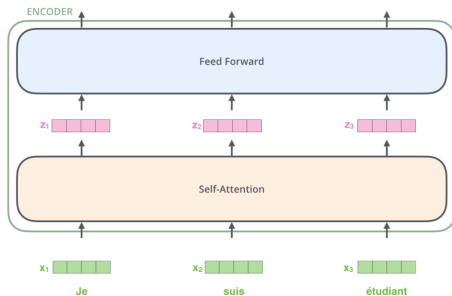
- The input to the encoder of Transformer are the embedding vectors of tokens in input sequence.
- The **self-attention layer** processes the embedding vectors in parallel to obtain new representations of the tokens.

Self-Attention Layer

- The purpose of the self-attention layer is to “improve” the representation of each token by combining information from other tokens.
- Consider sentence: The animal didn’t cross the street because it was too tired.
- What does “it” refer to? The street or the animal?
- Self-attention is able to associate “it” with “animal” when obtaining a new representation for “it”



Self-Attention Layer



- Let x_1, \dots, x_n be the current representations of input tokens. They are all row vectors of dimension $d_m (= 512)$.
- Consider obtaining a new representation z_i for token i . We need to decide:
 - How much attention to pay to each x_j ?
 - How to combine the x_j 's into z_i ?

Self-Attention Layer

- Moreover, we want the model to learn how to answer those questions from data.
- So, we introduce three matrices of **learnable parameters** (aka projection matrices):
 - W^Q : a $d_m \times d_k$ matrix ($d_k = 64$)
 - W^K : a $d_m \times d_k$ matrix
 - W^V : a $d_m \times d_v$ matrix ($d_v = 64$)

Self-Attention Layer

- Using the three matrices of parameters, we compute z_i as follows:
 - Project x_i and x_j to get query, key, and value:
 - $q_i = x_i W^Q$: **query** vector of dimension d_k .
 - $k_j = x_j W^K$: **key** vector of dimension d_k .
 - $v_j = x_j W^V$: **value** vector of dimension d_v .
 - Compute attention weights:
 - Dot-product attention: $\alpha_{i,j} \leftarrow q_i k_j^T$.
 - **Scaled dot-product attention**: $\alpha_{i,j} \leftarrow \frac{\alpha_{i,j}}{\sqrt{d_k}}$.
 - Apply softmax: $\alpha_{i,j} \leftarrow \frac{e^{\alpha_{i,j}}}{\sum_j e^{\alpha_{i,j}}}$
 - Obtain z_i (a vector of dimension d_v) by:

$$z_i = \sum_j \alpha_{i,j} v_j$$

Self-Attention Layer: Example

Input

Embedding

Queries

Keys

Values

Score

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

Softmax


Softmax

X

Value

Sum

Thinking

 x_1  q_1  k_1  v_1  $q_1 \cdot k_1 = 112$

14

0.88

 v_1  z_1 

Machines

 x_2  q_2  k_2  v_2  $q_1 \cdot k_2 = 96$

12

0.12

 v_2  z_2 

Self-Attention Layer: Matrix Notation

- Let X be the matrix with x_j 's as row vectors.
- $Q = XW^Q$, $K = XW^K$, $V = XW^V$
- Let Z be the matrix with z_i 's as row vectors.
- Then,

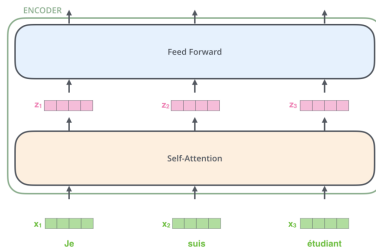
$$Z = \text{softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) V$$

$$X \times W^Q = Q$$

$$X \times W^K = K$$

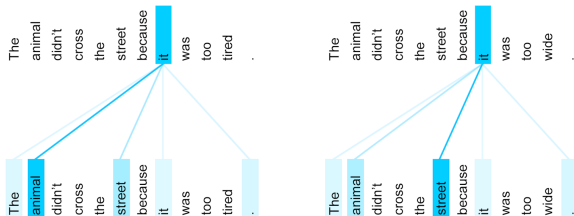
$$X \times W^V = V$$

Multi-Head Attention



- So far, we have been talking about obtaining **one** new representation z_i for taken i .
- It combines information from x_j 's in one way.

Multi-Head Attention



- It is sometimes useful to consider multiple ways to combine information from x_j 's, or multiple attentions.
- To do so, introduce multiple sets of projection matrices: W_i^Q, W_i^K, W_i^V ($i = 1, \dots, h$), each of which is called an **attention head**

Multi-Head Attention

- For each head i , let $Q_i = XW_i^Q$, $K_i = XW_i^K$, $V_i = XW_i^V$, and we get attention output:

$$Z_i = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right) V_i$$

- Then we concatenate those matrices to get the overall output $Z = \text{Concat}(Z_1, \dots, Z_h)$ with hd_v columns.
- To ensure the new embedding of each token is also of dimension d_m , introduce another $hd_v \times d_m$ matrix W^O of learnable parameter and project:

$$Z \leftarrow ZW^O.$$

- In Transformer, $d_m = 512$, $h = 8$, $d_v = 64$.

Self-Attention Layer: Summary

1) This is our input sentence*

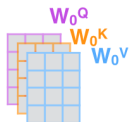
2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting $Q/K/V$ matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

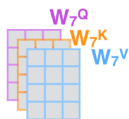
Thinking
Machines



...

...

...



W^O



Z

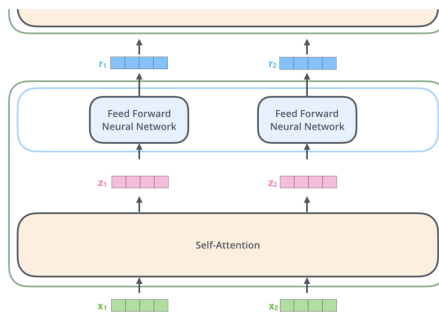


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



Encoder Block

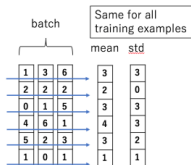
- Each output vector of the self-attention layer is fed to a feedforward network.
- The FNN's at different positions share parameters and function independently.
- The self-attention layer and the FNN layer make up one **encoder layer** (aka encoder block).
- The self-attention layer and FNN layer are hence called **sub-layers**.



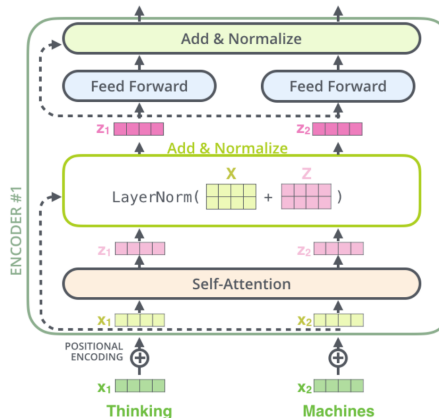
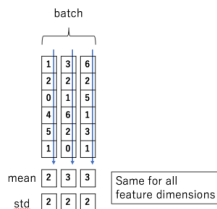
Residual Connection

- A **residual connection** is added to each sub-layer, followed by **layer normalization**.
- This enables us to deep models with many layers.

Batch Normalization

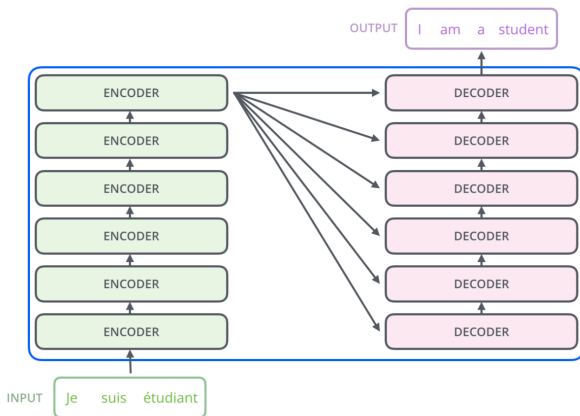


Layer Normalization



The Encoder

The encoder is composed of a stack of $N = 6$ encoder blocks with the same structure, but **different parameters (i.e., no weight sharing across different encoder blocks)**.



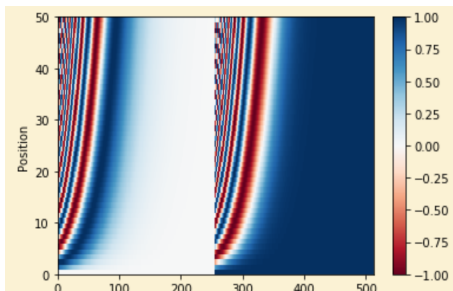
Positional Encoding

- Transformer contains no recurrence, and hence needs to inject information about token positions in order to make use of order of sequence. **Positional encoding** is therefore introduced.
- For a token at position pos , its PE is a vector of d_m dimensions:

$$PE(pos, 2i) = \sin(pos/1000^{2i/d_m})$$

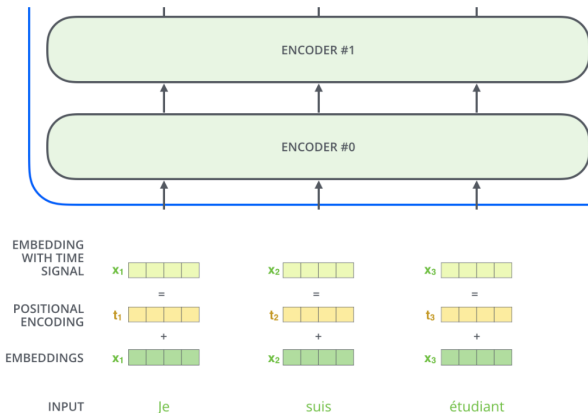
$$PE(pos, 2i + 1) = \cos(pos/1000^{2i/d_m})$$

where $0 \leq i \leq 255$

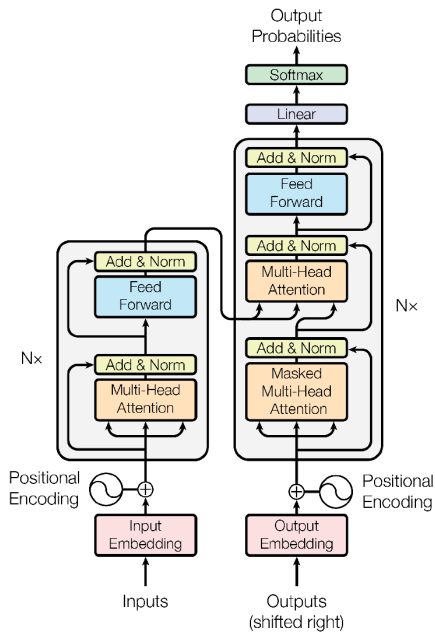


Positional Encoding

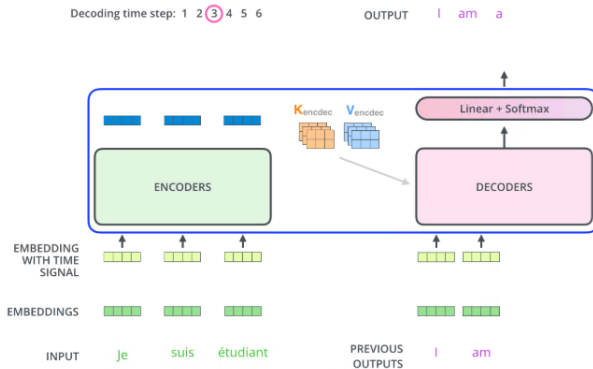
- Positional encodings are added to input embeddings at the bottom.



Complete structure of the encoder is shown on the next slide.



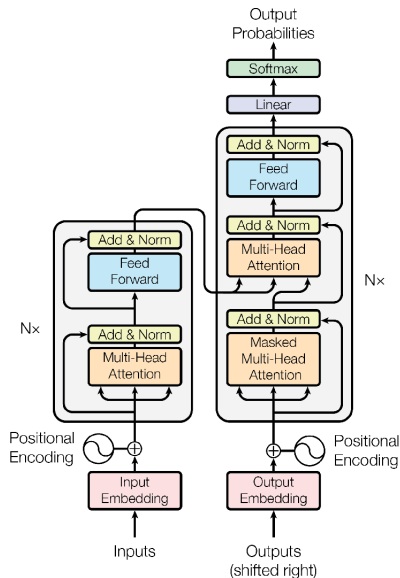
The Decoder



- The decoder generates one word at a time. At a given step, its inputs consists of all the words generated already. The representation of those words are added with positional encodings, and the results are fed to be decoder.

The Decoder

- The decoder has a stack of $N = 6$ identical decoder blocks.
- A decoder block is the same as an encoder block, except that it has an additional **decoder-encoder attention layer** between self-attention and FNN sub-layers

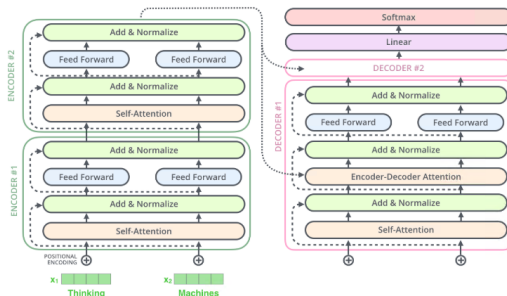


Decoder Block

- The **decoder-encoder attention layer** performs multi-head attention where
 - The queries Q come from the previous decoder layer.
 - The keys K and values V come from the output of encoder.

$$Z_i = \text{softmax}\left(\frac{Q_i K_i^\top}{\sqrt{d_k}}\right) V_i.$$

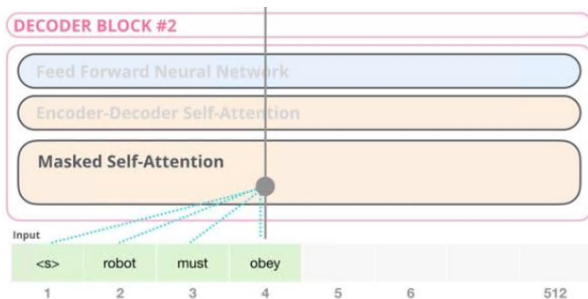
where $Q_i = X_{\text{decoder}} W_i^Q$, $K_i = X_{\text{encoder}} W_i^K$, $V_i = X_{\text{encoder}} W_i^V$.
 Similar to RNN. (Illustration with $N = 2$)



Decoder Block

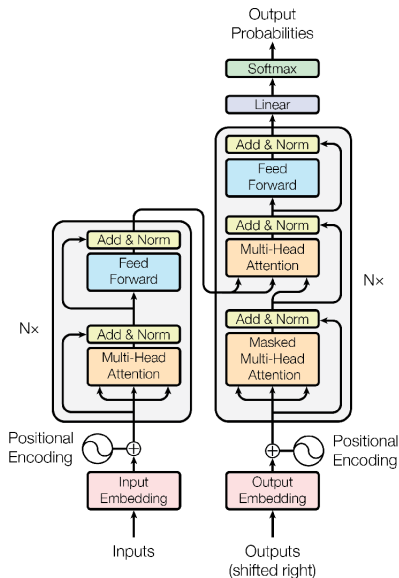
- Self-attention layer in decoder differs from that in encoder in one important way. Each position can only attend to all preceding positions and itself, because there are no inputs from future positions.

- Implementation: For $j > i$, set $\alpha_{i,j} = -\infty$. Apply softmax:
$$\alpha_{i,j} \leftarrow \frac{e^{\alpha_{i,j}}}{\sum_j e^{\alpha_{i,j}}}$$
 Then $\alpha_{i,j} = 0$ for all $j > i$.



The Decoder

- Finally, the decoder has a softmax layer that defines a distribution over the vocabulary, and a word is sampled from the distribution as the next output.
- The loss function is defined in the same way as in RNN.
- All parameters of the model are optimized by minimizing the loss function using the Adam optimizer.



Transformer in Action

http://jalammar.github.io/images/t/transformer_decoding_1.gif

http://jalammar.github.io/images/t/transformer_decoding_2.gif

Empirical Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

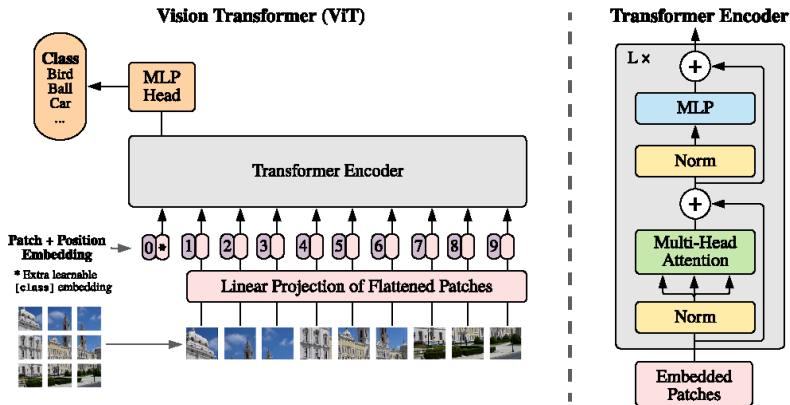
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.0	$2.3 \cdot 10^{19}$	

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	65
big	6	1024	4096	16			0.3		300K	213

Conclusions on Transformer

- Transformer is first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.
- For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers.
- On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art.

Vision Transformer (ViT)¹



- We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder.
- In order to perform classification, we use the standard approach of adding an extra learnable classification token to the sequence.

¹Dosovitskiy *et al* 2021, AN IMAGE IS WORTH 16X16 WORDS ...

Vision Transformer (ViT)

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	88.55 ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	90.72 ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	99.50 ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	—
CIFAR-100	94.55 ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	—
Oxford-IIIT Pets	97.56 ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	—
Oxford Flowers-102	99.68 ± 0.02	99.74 ± 0.00	99.61 ± 0.02	99.63 ± 0.03	—
VTAB (19 tasks)	77.63 ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	—
TPUV3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Table 2: Comparison with state of the art on popular image classification benchmarks. We report mean and standard deviation of the accuracies, averaged over three fine-tuning runs. Vision Transformer models pre-trained on the JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substantially less computational resources to pre-train. ViT pre-trained on the smaller public ImageNet-21k dataset performs well too. *Slightly improved 88.5% result reported in [Touvron et al. \(2020\)](#).

Outline

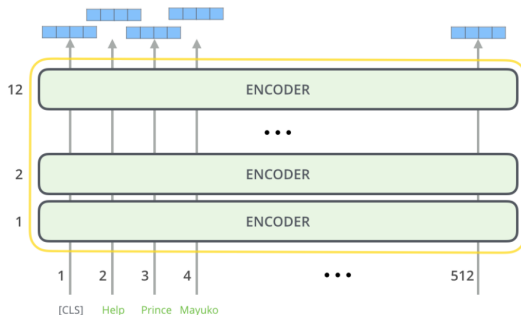
1 Transformer

- Self-Attention Layer
- The Encoder
- The Decoder
- Vision Transformer (ViT)

2 BERT

- Overview
- Pre-training BERT
- Fine-tuning BERT

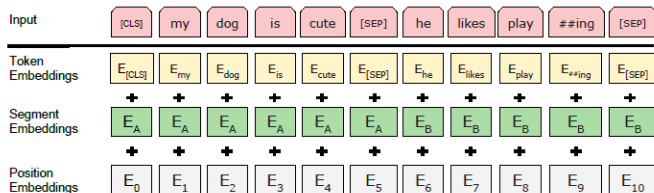
BERT



BERT: Bidirectional Encoder Representations from Transformers

- BERT is a language representation model that **maps a token sequence into a sequence of vectors**. It consists of a stack of Transformer encoder blocks:
 - BERT_BASE: $N = 12$ **Transformer encoder blocks**, hidden size $d_m = 768$, # of heads $h = 12$; 110M parameters.
 - BERT_LARGE: $N = 24$ **Transformer encoder blocks**, hidden size $d_m = 1024$, # of heads $h = 16$; 340M parameters.

Input of BERT during training



- The input BERT is a pair of “sentences” (A, B).
- It starts with a special classification token [CLS], and the two sentences are separated by a special token [SEP].
- Each token has a **token embedding**, a **segment embedding**, and a **positional embedding**. Their sum is feed to BERT.
- Segment embedding: E_A is vector of 0's and E_B is vector of 1's.

Training Examples

- BERT is pre-trained on unsupervised tasks.
- Training examples are “sentence” pairs (A, B) sampled from BooksCorpus and Wikipedia:
 - 50% of the time, B actually follows A, and 50% of the time it is a random sentence.
 - The combined length ≤ 512 tokens.
- Some of the tokens are randomly masked.
- Examples:

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

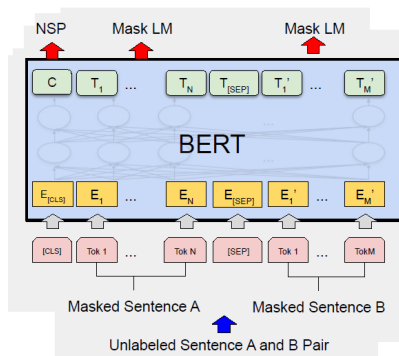
Label = IsNext

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

Label = NotNext

The Loss Function

- BERT is trained using two unsupervised tasks:
 - **Masked Language Model (MLM):**
Predict the masked words.
 - **Next Sentence Prediction (NSP):**
Predict whether B follows A.
- The training loss is the sum of the losses of those two tasks.



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

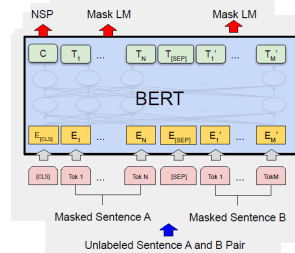
Label = IsNext

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

Label = NotNext

Masked Language Model (MLM) (aka the Cloze task)

- For each training example, 15% of the tokens are chosen for prediction.
- Suppose the i -th token is chosen. It is,
 - 1 With 80% chance, replaced with [MASK],
 - 2 With 10% chance, with a random token,
 - 3 With 10% chance, unchanged.
- Then, the output vector T_i is used to predict the original token with the cross entropy loss.
- Recall: In standard LM, the task is to predict the next word given all the previous words.



MLM ensures that T_i is a “good” contextual representation of the i -th token of the input sequence in relation to ALL other tokens in the sequence. This is why it is bidirectional.

Masking Strategy Motivation

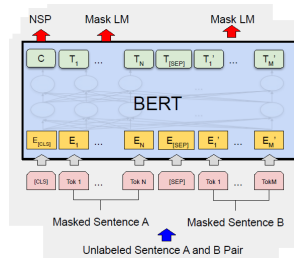
- During pre-training, we have the [MASK] token.
 - [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]
 - Label = IsNext.
- During fine-tuning, we don't:
 - [CLS] the man went to the store [SEP] he bought a gallon of milk [SEP]
 - Question: IsNext?
- It is to mitigate the mismatch that items 2 and 3 of the masking strategy are introduced. This will help with fine-tuning.

Next Sentence Prediction (NSP)

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

- The output vector C is used to predict whether B follows A with cross entropy loss.
- NSP ensures that C contains information about whether one sentence follows another.
- However, it is not a meaningful summary of the contents of the sentences. Fine-tuning is required if we want to get meaningful sentence representations.

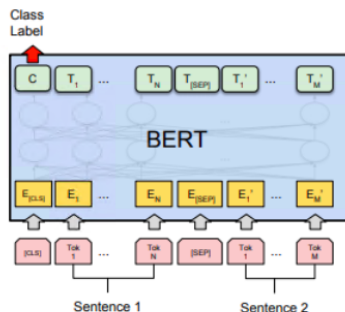


Pre-training Procedure

- We train with **batch size of 256 sequences (256 sequences * 512 tokens = 128,000 tokens/batch)** for 1,000,000 steps, which is approximately **40 epochs** over the 3.3 billion word corpus.
- Training of BERT_BASE was performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total). Training of BERT_LARGE was performed on 16 Cloud TPUs (64 TPU chips total). Each pretraining took 4 days to complete.

Applying BERT to Downstream Tasks

- Feed token representation from BERT to a task-specific output layer. Fine-tune **all** BERT parameters for that task (and learn the parameters of the output layer).
- For sentence-pair classification, the representation C of [CLS] is fed to a one-layer classifier.
- Example: MNLI – sentence pairs annotated with textual entailment. **Fine-tuning makes C more suitable for the task (entailment \neq isNext).**



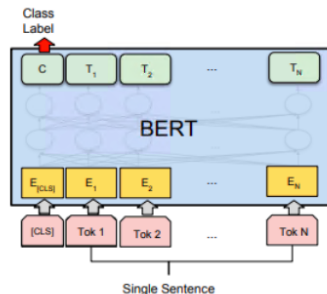
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

Premise	Label	Hypothesis
Your gift is appreciated by each and every student who will benefit from your generosity.	<i>neutral</i>	Hundreds of students will benefit from your generosity.
yes now you know if everybody like in August when everybody's on vacation or something we can dress a little more casual or	<i>contradiction</i>	August is a black out month for vacations in the company.
At the other end of Pennsylvania Avenue, people began to line up for a White House tour.	<i>entailment</i>	People formed a line at the end of Pennsylvania Avenue.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8
OpenAI GPT	82.1/81.4	70.3	87.4
BERT _{BASE}	84.6/83.4	71.2	90.5
BERT _{LARGE}	86.7/85.9	72.1	92.7

Applying BERT to Downstream Tasks

- For single sentence classification, the BERT representation C of [CLS] is fed to a one-layer classifier. (This is why the token is called the classification token.)
- After pre-training, C contains information about if one sentence follows another.
- But here, we need C be a summary of information of the in the input (single) sentence.
- Fine-tuning achieves this goal.



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

System	SST-2 67k	CoLA 8.5k
Pre-OpenAI SOTA	93.2	35.0
BiLSTM+ELMo+Attn	90.4	36.0
OpenAI GPT	91.3	45.4
BERT _{BASE}	93.5	52.1
BERT _{LARGE}	94.9	60.5

Applying BERT to Downstream Tasks

- For token-level tasks, representations of tokens are fed to an output layer.
- Example: Name Entity Recognition (NER)

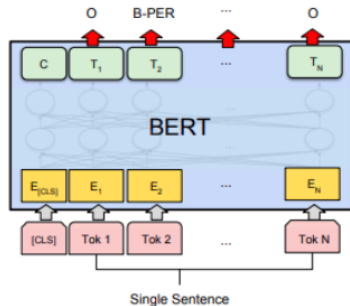
Input: Vancouver is a coastal seaport city on the mainland of British Columbia. The city's mayor is Gregor Robertson.

Location

Output: Vancouver is a coastal seaport city on the mainland of British Columbia. The city's mayor is Gregor Robertson.

Location

Person



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

Fine-tuning is inexpensive

- Compared to pre-training, fine-tuning is relatively inexpensive.
- All of the results in the paper can be replicated in at most 1 hour on a single Cloud TPU, or a few hours on a GPU, starting from the exact same pre-trained model.

Conclusions on BERT

- Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers.
- As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.
- BERT is conceptually simple and empirically powerful.
- It obtains new state-of-the-art results on eleven natural language processing tasks,

References

- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
- Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.www.deeplearningbook.org
- Jay Alammar: <http://jalammar.github.io/>
- Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.