

Natural Language Processing with Deep Learning

RUHR
UNIVERSITÄT
BOCHUM

RUB

Lecture 7 — BERT as encoder-only transformer

Prof. Dr. Ivan Habernal

December 4, 2025

www.trusthlt.org

Trustworthy Human Language Technologies Group (TrustHLT)

Ruhr University Bochum & Research Center Trustworthy Data Science and Security



CENTER FOR TRUSTWORTHY
DATA SCIENCE AND SECURITY

Motivation

- 1 Motivation
- 2 BERT — Encoder architecture in detail
- 3 Input and pre-training
- 4 Pre-training
- 5 Downstream tasks and fine-tuning

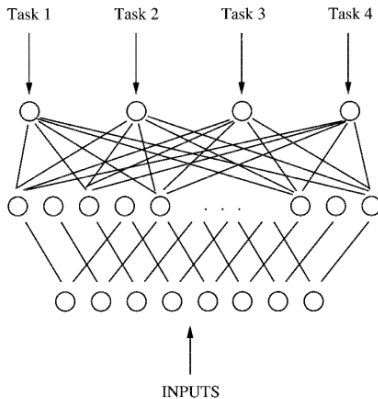
Motivation

Multi-task learning

Multi-task Learning

Approach to inductive transfer that improves generalization

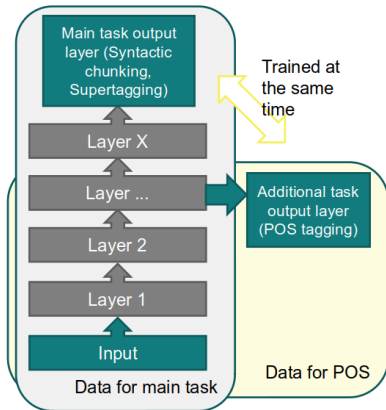
By learning tasks in parallel while using a shared representation



R. Caruana (1997). **"Multi-task Learning"**. In: *Machine Learning* 28.1, pp. 41–75

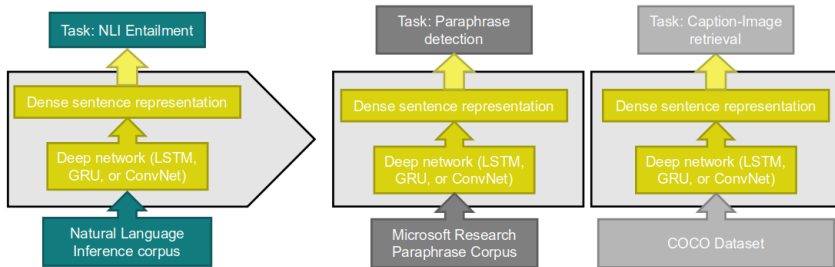
Multi-task learning in NLP

"In case we suspect the existence of a hierarchy between the different tasks, we show that it is worth-while to incorporate this knowledge in the MTL architecture's design, by making lower level tasks affect the lower levels of the representation."



A. Søgaard and Y. Goldberg (2016).
"Deep multi-task learning with low level tasks supervised at lower layers". In: *Proceedings of ACL*. Berlin, Germany: Association for Computational Linguistics, pp. 231–235

Learn a sentence representation on a different task



A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes (2017). **"Supervised Learning of Universal Sentence Representations from Natural Language Inference Data"**. In: *Proceedings of EMNLP*. Copenhagen, Denmark, pp. 670–680

"Models learned on NLI can perform better than models trained in unsupervised conditions or on other supervised tasks."

Bottlenecks of RNN for representation learning

Inherently **sequential** nature

- No parallelization
- Long-range dependencies modeling: Distance plays a role!

...but when the goal is to learn a good representation of the input sequence, we might have better/faster architectures

Also recall disadvantages of **static word embeddings**

Today: Transformers and the BERT architecture

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

BERT was a game-changer in NLP:

“BERT is conceptually simple and empirically powerful. It obtains new **state-of-the-art results on eleven natural language processing tasks**, including pushing the GLUE score to 80.5% (**7.7% point absolute improvement**), MultiNLI accuracy to 86.7% (**4.6% absolute improvement**), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (**5.1 point absolute improvement**).”

After this lecture you should be able to build BERT

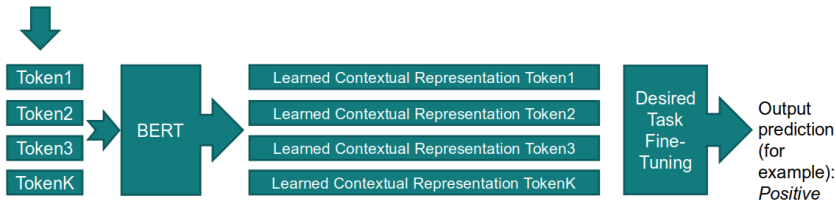
J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova (2019). “**BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**”. In: *Proceedings of NAACL*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186

BERT — Encoder architecture in detail

- 1 Motivation
- 2 BERT — Encoder architecture in detail**
- 3 Input and pre-training
- 4 Pre-training
- 5 Downstream tasks and fine-tuning

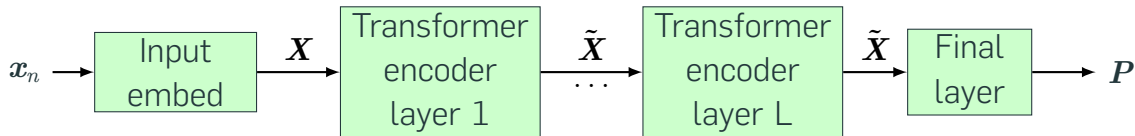
BERT: Very abstract view

Input text: *Lorem ipsum dolor*



- BERT produces contextualized token embeddings
- BERT can learn them in a 'clever' way
- BERT can be applied to many downstream tasks

Transformer encoder (BERT)



As usual, green boxes are functions with trainable parameters

\tilde{X} is just a placeholder for **updated** token embeddings matrix X

Some details (Notation)

Notation and formal description of algorithms adopted from M. Phuong and M. Hutter (2022). **Formal Algorithms for Transformers**. arXiv: 2207.09238 Note that they use column-vector notation while here (and in all lectures) we use row-vector notation.

Simplify the set notation

$\{1, 2, \dots, N\}$ is a set of integers $1, 2, \dots, N - 1, N$

simplify to $[N]$

For example $t \in [N] \equiv t \in \{1, 2, \dots, N\}$

BERT (encoding-only transformer, forward pass)

- 1: **function** ETransformer(x ; \mathcal{W})
- 2: ...

Input:

x — $x \in V^*$, a sequence of token IDs

\mathcal{W} — all trainable parameters

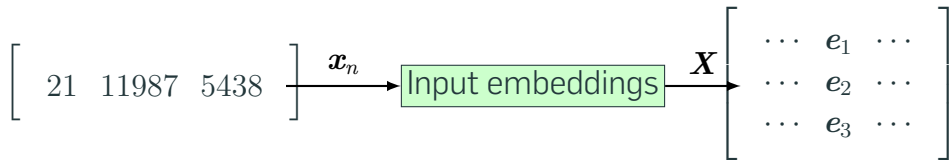
Output:

Typically an embedding vector for each input token

Or: $\mathbf{P} \in (0, 1)^{\ell_x \times N_V}$, where each row of \mathbf{P} is a distribution over the vocabulary

Input embeddings

The cat sat $\mathbf{x}_n = (21 \quad 11987 \quad 5438)$



Positional embeddings

For each input position t , we learn (train) an embedding vector $\mathbf{W}_p[t]$, for example

$$\mathbf{W}_p = \begin{pmatrix} \mathbf{W}_p[1] \\ \mathbf{W}_p[2] \\ \vdots \\ \mathbf{W}_p[\ell] \end{pmatrix} = \begin{pmatrix} 1.12 & -78.6 & \cdots \\ \vdots & & \vdots \\ \vdots & & \vdots \\ -0.1 & 799.7 & \cdots \end{pmatrix}$$

The model knows with which part of the input/output is dealing with

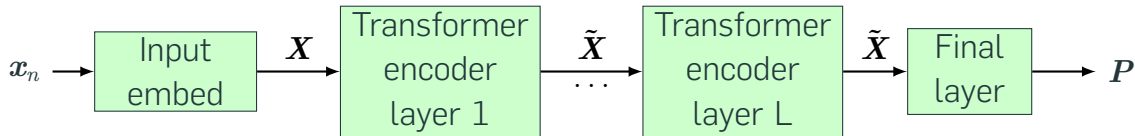
Originally proposed for CNNs for MT, state-of-the-art results and **9.3–21.3× faster** than LSTMs on GPU

J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin (2017). “**Convolutional Sequence to Sequence Learning**”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by D. Precup and Y. W. Teh. Sydney, Australia: PMLR, pp. 1243–1252

BERT (encoding-only transformer, forward pass)

- 1: **function** ETransformer($\mathbf{x}; \mathcal{W}$)
- 2: $\ell \leftarrow \text{length}(\mathbf{x})$
- 3: for $t \in [\ell] : \mathbf{e}_t \leftarrow \mathbf{W}_e[\mathbf{x}[t], :] + \mathbf{W}_p[t, :]$ ▷ Token emb. + positional emb.
- 4: $\mathbf{X} \leftarrow \text{Stack row-wise}[\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_\ell]$
- 5: ...

Transformer encoder (BERT)

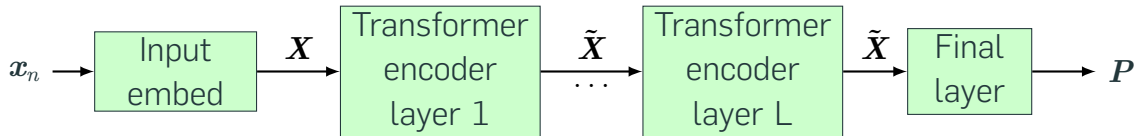


The transformer encoder layer is repeated L -times (each with **different** parameters)

BERT (encoding-only transformer, forward pass)

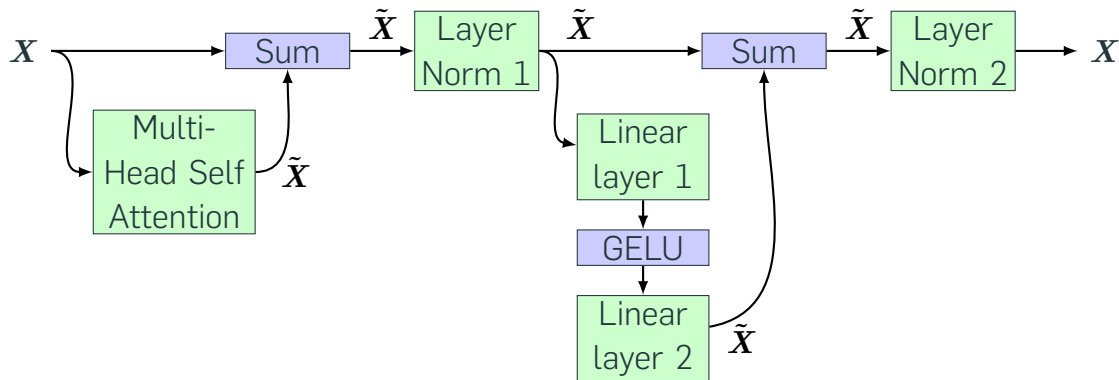
```
1: function ETransformer( $\mathbf{x}$ ;  $\mathcal{W}$ )
2:    $\ell \leftarrow \text{length}(\mathbf{x})$ 
3:   for  $t \in [\ell]$  :  $\mathbf{e}_t \leftarrow \mathbf{W}_e[\mathbf{x}[t], :] + \mathbf{W}_p[t, :]$        $\triangleright$  Token emb. + positional emb.
4:    $\mathbf{X} \leftarrow \text{Stack row-wise}[\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_\ell]$ 
5:   for  $l = 1, 2, \dots, L$  do
6:     ...
```

Transformer encoder (BERT)



Let's look at a single transformer encoder layer

Transformer encoder layer (BERT)

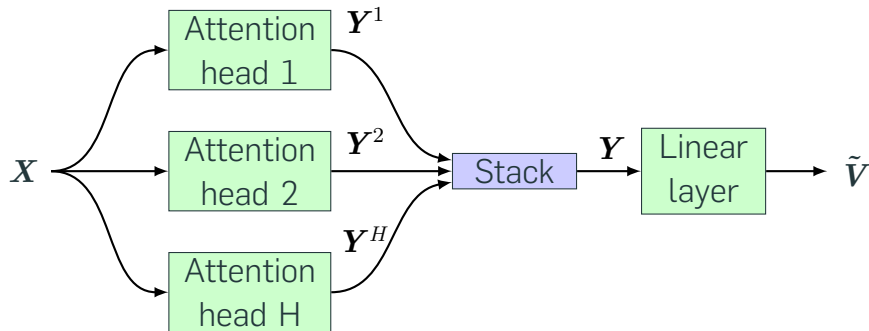


Let's focus on Multi-Head Self Attention

BERT (encoding-only transformer, forward pass)

```
1: function ETransformer( $\mathbf{x}$ ;  $\mathcal{W}$ )
2:    $\ell \leftarrow \text{length}(\mathbf{x})$ 
3:   for  $t \in [\ell]$  :  $\mathbf{e}_t \leftarrow \mathbf{W}_e[\mathbf{x}[t], :] + \mathbf{W}_p[t, :]$            ▷ Token emb. + positional emb.
4:    $\mathbf{X} \leftarrow \text{Stack row-wise}[\mathbf{e}_1, \mathbf{e}_2, \dots \mathbf{e}_\ell]$ 
5:   for  $l = 1, 2, \dots, L$  do
6:      $\mathbf{X} \leftarrow \mathbf{X} + \text{MHAttention}(\mathbf{X} | \mathcal{W}_l)$            ▷ Multi-head att., residual conn
7:     ...
```

Multi-head unmasked self-attention



Some notation details

Concatenate matrices of the same dimensions along rows

$$\mathbf{Y} = [\mathbf{X}^1; \mathbf{X}^2; \dots; \mathbf{X}^H] \quad \mathbf{X}^i \in \mathbb{R}^{m \times n} \quad \mathbf{Y} \in \mathbb{R}^{m \times H \cdot n}$$

Example

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \mathbf{B} = \begin{pmatrix} 11 & 12 \\ 13 & 14 \\ 15 & 16 \end{pmatrix} \quad \mathbf{Y} = [\mathbf{A}; \mathbf{B}] = \begin{pmatrix} 1 & 2 & 11 & 12 \\ 3 & 4 & 13 & 14 \\ 5 & 6 & 15 & 16 \end{pmatrix}$$

Multi-head bidirectional unmasked self-attention

Input: $\mathbf{X} \in \mathbb{R}^{\ell_x \times d_x}$, vector representations of the sequence of length ℓ_x

Output: $\tilde{\mathbf{V}} \in \mathbb{R}^{\ell_x \times d_{\text{out}}}$, updated vector representations of tokens in \mathbf{X}

Hyper-param: H , number of attention heads

Params for each $h \in [H]$: \mathcal{W}_{qkv}^h :

■ $\mathbf{W}_q^h, \mathbf{W}_k^h \in \mathbb{R}^{d_x \times d_{\text{attn}}}, \mathbf{b}_q^h, \mathbf{b}_k^h \in \mathbb{R}^{d_{\text{attn}}}, \mathbf{W}_v \in \mathbb{R}^{d_x \times d_{\text{mid}}}, \mathbf{b}_v \in \mathbb{R}^{d_{\text{mid}}}$

■ $\mathbf{W}_o \in \mathbb{R}^{H \cdot d_{\text{mid}} \times d_{\text{out}}}, \mathbf{b}_o \in \mathbb{R}^{d_{\text{out}}}$

1: **function** MHAttention($\mathbf{X}; \mathcal{W}$)

2: **for** $h \in [H]$ **do**

3: $\mathbf{Y}^h \leftarrow \text{Attention}(\mathbf{X}; \mathcal{W}_{qkv}^h)$

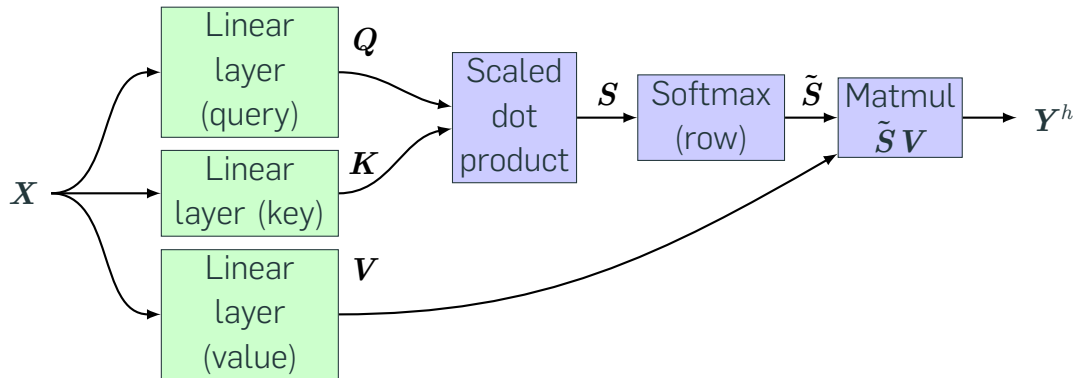
▷ $\mathbf{Y}^h \in \mathbb{R}^{\ell_x \times d_{\text{mid}}}$

4: $\mathbf{Y} \leftarrow [\mathbf{Y}^1; \mathbf{Y}^2; \dots; \mathbf{Y}^H]$

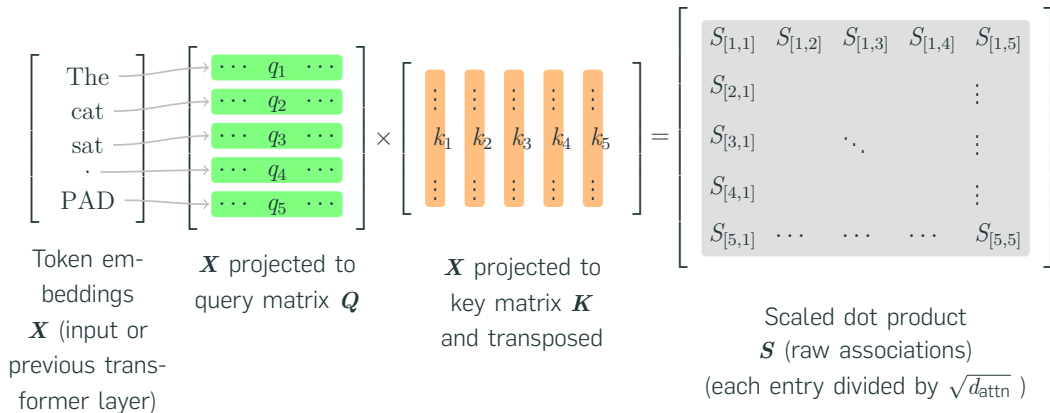
▷ $\mathbf{Y} \in \mathbb{R}^{\ell_x \times H \cdot d_{\text{mid}}}$

5: **return** $\tilde{\mathbf{V}} = \mathbf{Y}\mathbf{W}_o + \mathbf{b}_o$

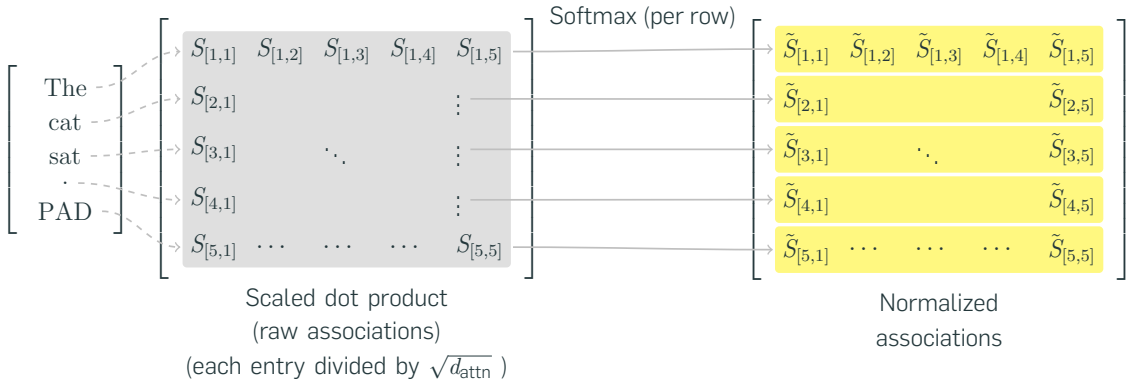
Single unmasked self-attention head



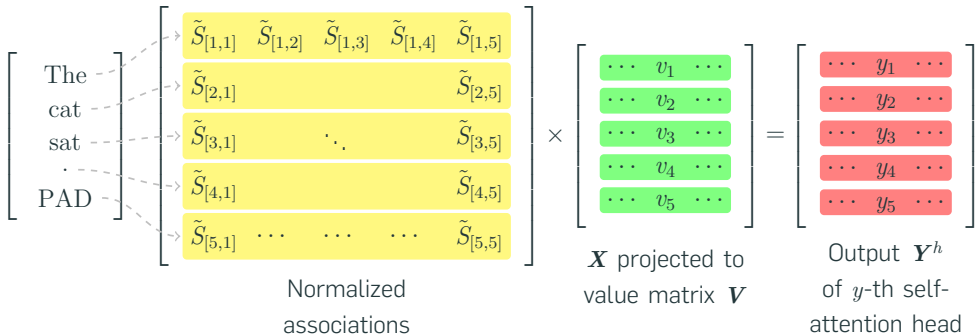
Self-attention in detail — query, key, scaled dot product



Self-attention in detail — softmax over scaled dot product



Self-Attention in detail — head output by weighting the value



Some notation details

How to add a single vector \mathbf{b} to each row in a matrix \mathbf{W}

$$(\mathbf{W} \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^n)$$

We want $\mathbf{Z} = \mathbf{X} +_{(\text{rows})} \mathbf{b}$

Let $\mathbf{1}^m = (1, 1, \dots, 1_m)$, then $\mathbf{Z} = \mathbf{X} +_{(\text{rows})} \mathbf{b} = \mathbf{X} + (\mathbf{b}^\top \mathbf{1}^m)^\top$

Example

$$\mathbf{X} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \mathbf{b} = (10 \quad 20)$$

$$\mathbf{b}^\top \mathbf{1}^m = \begin{pmatrix} 10 \\ 20 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 10 & 10 & 10 \\ 20 & 20 & 20 \end{pmatrix} \quad (\mathbf{b}^\top \mathbf{1}^m)^\top = \begin{pmatrix} 10 & 20 \\ 10 & 20 \\ 10 & 20 \end{pmatrix}$$

Some notation details

Soft-max for matrices row-wise, $\mathbf{A} \in \mathbb{R}^{m \times n}$

$$\text{softmax}_{\text{row}} : \mathbb{R}^{m \times n} \mapsto \mathbb{R}^{m \times n}$$

$$\text{softmax}_{\text{row}}(\mathbf{A})[i, j] = \frac{\exp(\mathbf{A}[i, j])}{\sum_{k=1}^n \exp(\mathbf{A}[i, k])}$$

Bidirectional unmasked self-attention precisely

Input: $\mathbf{X} \in \mathbb{R}^{\ell_x \times d_x}$, vector representations of the sequence of length ℓ_x

Output: $\tilde{\mathbf{V}} \in \mathbb{R}^{\ell_x \times d_{\text{out}}}$, updated vector representations of tokens in \mathbf{X}

Params \mathcal{W}_{qkv} : $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{d_x \times d_{\text{attn}}}$, $\mathbf{b}_q, \mathbf{b}_k \in \mathbb{R}^{d_{\text{attn}}}$, $\mathbf{W}_v \in \mathbb{R}^{d_x \times d_{\text{out}}}$, $\mathbf{b}_v \in \mathbb{R}^{d_{\text{out}}}$

1: **function** Attention($\mathbf{X}; \mathcal{W}_{qkv}$)

2: $\mathbf{Q} \leftarrow \mathbf{X} \mathbf{W}_q +_{(\text{rows})} \mathbf{b}_q$

▷ Query $\in \mathbb{R}^{\ell_x \times d_{\text{attn}}}$

3: $\mathbf{K} \leftarrow \mathbf{X} \mathbf{W}_k +_{(\text{rows})} \mathbf{b}_k$

▷ Key $\in \mathbb{R}^{\ell_x \times d_{\text{attn}}}$

4: $\mathbf{V} \leftarrow \mathbf{X} \mathbf{W}_v +_{(\text{rows})} \mathbf{b}_v$

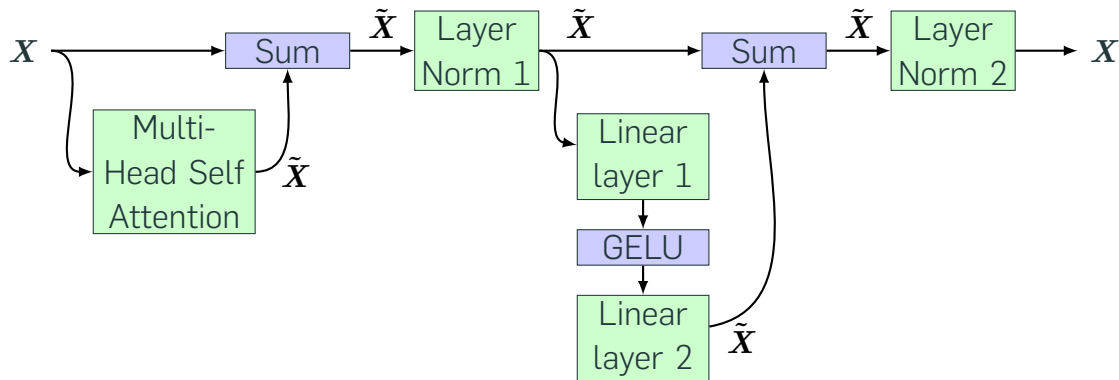
▷ Value $\in \mathbb{R}^{\ell_x \times d_{\text{out}}}$

5: $\mathbf{S} \leftarrow \frac{1}{\sqrt{d_{\text{attn}}}} (\mathbf{Q} \mathbf{K}^\top)$

▷ Scaled score $\in \mathbb{R}^{\ell_x \times \ell_x}$

6: **return** $\tilde{\mathbf{V}} = \text{softmax}_{\text{row}}(\mathbf{S}) \mathbf{V}$

Transformer encoder layer (BERT)



Let's look at Layer Normalization and GELU

Layer normalization

Input: $\mathbf{e} \in \mathbb{R}^d$, output of a layer

Input: $\hat{\mathbf{e}} \in \mathbb{R}^d$, normalized output of a layer

Parameters: $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^d$, element-wise scale and offset

- 1: **function** LayerNorm($\mathbf{e}; \boldsymbol{\gamma}, \boldsymbol{\beta}$)
- 2: $m \leftarrow \frac{1}{d} \sum_{i=1}^d \mathbf{e}[i]$ ▷ 'Sample mean' of \mathbf{e}
- 3: $v \leftarrow \frac{1}{d} \sum_{i=1}^d (\mathbf{e}[i] - m)^2$ ▷ 'Sample variance' of \mathbf{e}
- 4: **return** $\hat{\mathbf{e}} = \frac{\mathbf{e} - m}{\sqrt{v}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$ ▷ \odot element-wise product

(some transformers use $m = \boldsymbol{\beta} = 0$)

Simplifying notation: Perform LayerNorm on each row

```
1: function LayerNormEachRow( $\mathbf{X} \in \mathbb{R}^{m \times n} | \gamma, \beta$ )  
2:   for  $t \in [m]$  do  
3:      $\mathbf{X}[t, :] \leftarrow \text{LayerNorm}(\mathbf{X}[t, :] | \gamma, \beta)$   
4:   return  $\mathbf{X}$ 
```

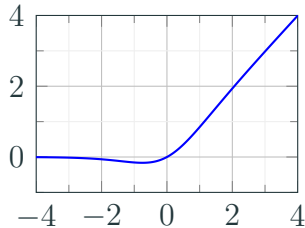
GELU — Gaussian Error Linear Units

Recall: CDF $\Phi(x)$ of standard normal $X \sim \mathcal{N}(0; 1)$

$$\Phi(x) = \Pr(X \leq x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp\left(-\frac{t^2}{2}\right) dt$$

$$\text{GELU}(x) = x \cdot \Phi(x)$$

$$\approx x \cdot \sigma(1.702x) \quad (\text{if speed} > \text{exactness})$$



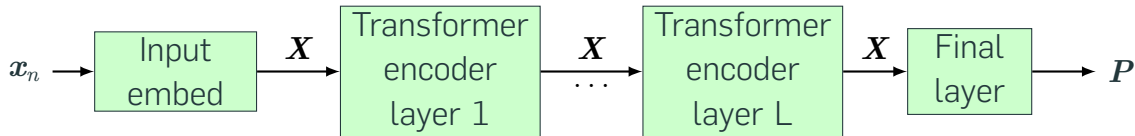
D. Hendrycks and K. Gimpel (2016).
Gaussian Error Linear Units (GELUs).
arXiv: 1606.08415

For vectors $\mathbf{x} \in \mathbb{R}^n$, $\text{GELU}(\mathbf{x})$ is
applied element-wise

BERT (encoding-only transformer, forward pass)

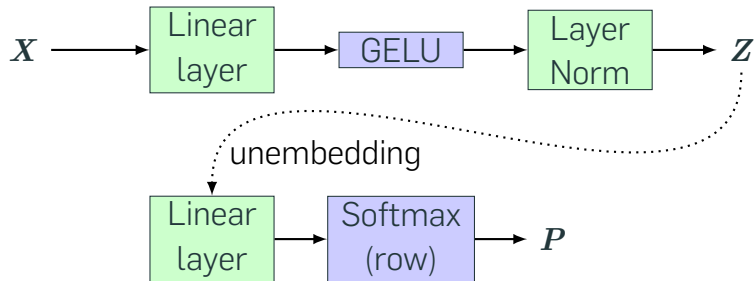
```
1: function ETransformer( $\mathbf{x}$ ;  $\mathcal{W}$ )
2:    $\ell \leftarrow \text{length}(\mathbf{x})$ 
3:   for  $t \in [\ell]$  :  $\mathbf{e}_t \leftarrow \mathbf{W}_e[\mathbf{x}[t], :] + \mathbf{W}_p[t, :]$  ▷ Token emb. + positional emb.
4:    $\mathbf{X} \leftarrow \text{Stack row-wise}[\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_\ell]$ 
5:   for  $l = 1, 2, \dots, L$  do
6:      $\mathbf{X} \leftarrow \mathbf{X} + \text{MHAttention}(\mathbf{X} | \mathcal{W}_l)$  ▷ Multi-head att., residual conn
7:      $\mathbf{X} \leftarrow \text{LayerNormPerRow}(\mathbf{X} | \gamma_l^1, \beta_l^1)$ 
8:      $\mathbf{X} \leftarrow \mathbf{X} + \left( \text{GELU}(\mathbf{X} \mathbf{W}_l^{\text{mlp1}} +_{(\text{row})} \mathbf{b}_l^{\text{mlp1}}) \mathbf{W}_l^{\text{mlp2}} +_{(\text{row})} \mathbf{b}_l^{\text{mlp2}} \right)$  ▷ MLP
9:      $\mathbf{X} \leftarrow \text{LayerNormPerRow}(\mathbf{X} | \gamma_l^2, \beta_l^2)$ 
10:  ...
```

Transformer encoder (BERT)



Let's look at the final layers

Final layer (BERT)



BERT (encoding-only transformer, forward pass)

```
1: function ETransformer( $\mathbf{x}; \mathcal{W}$ )
2:    $\ell \leftarrow \text{length}(\mathbf{x})$ 
3:   for  $t \in [\ell] : \mathbf{e}_t \leftarrow \mathbf{W}_e[x[t], :] + \mathbf{W}_p[t, :]$  ▷ Token emb. + positional emb.
4:    $\mathbf{X} \leftarrow \text{Stack row-wise}[\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_\ell]$ 
5:   for  $l = 1, 2, \dots, L$  do
6:      $\mathbf{X} \leftarrow \mathbf{X} + \text{MHAttention}(\mathbf{X} | \mathcal{W}_l)$  ▷ Multi-head att., residual conn
7:      $\mathbf{X} \leftarrow \text{LayerNormPerRow}(\mathbf{X} | \gamma_l^1, \beta_l^1)$ 
8:      $\mathbf{X} \leftarrow \mathbf{X} + \left( \text{GELU}(\mathbf{X} \mathbf{W}_l^{\text{mlp1}} +_{(\text{row})} \mathbf{b}_l^{\text{mlp1}}) \mathbf{W}_l^{\text{mlp2}} +_{(\text{row})} \mathbf{b}_l^{\text{mlp2}} \right)$  ▷ MLP
9:      $\mathbf{X} \leftarrow \text{LayerNormPerRow}(\mathbf{X} | \gamma_l^2, \beta_l^2)$ 
10:     $\mathbf{X} \leftarrow \text{GELU}(\mathbf{X} \mathbf{W}_f +_{(\text{row})} \mathbf{b}_f)$ 
11:     $\mathbf{X} \leftarrow \text{LayerNormPerRow}(\mathbf{X} | \gamma_l, \beta_l)$ 
12:    return  $\mathbf{P} = \text{softmax}(\mathbf{X} \mathbf{W}_u)$  ▷ Project to vocab., probabilities
```

BERT parameters and hyperparameters

Hyperparameters: $\ell_{\max}, L, H, d_e, d_{\text{mlp}}, d_f \in \mathbb{N}$

Parameters:

$\mathbf{W}_e \in \mathbb{R}^{N_V \times d_e}$, $\mathbf{W}_p \in \mathbb{R}^{\ell_{\max} \times d_e}$, the token and positional embedding matrices

For $l \in [L]$: \mathcal{W}_l , multi-head attention parameters for layer l :

- $\gamma_l^1, \beta_l^1, \gamma_l^2, \beta_l^2$, two sets of layer-norm parameters

- $\mathbf{W}_l^{\text{mlp1}} \in \mathbb{R}^{d_e \times d_{\text{mlp}}}$, $\mathbf{b}_l^{\text{mlp1}} \in \mathbb{R}^{d_{\text{mlp}}}$

- $\mathbf{W}_l^{\text{mlp2}} \in \mathbb{R}^{d_{\text{mlp}} \times d_e}$, $\mathbf{b}_l^{\text{mlp2}} \in \mathbb{R}^{d_e}$

$\mathbf{W}_f \in \mathbb{R}^{d_e \times d_f}$, $\mathbf{b}_f \in \mathbb{R}^{d_f}$, $\gamma, \beta \in \mathbb{R}^{d_f}$, the final linear projection and layer-norm parameters.

$\mathbf{W}_u \in \mathbb{R}^{d_e \times N_V}$, the unembedding matrix

Input and pre-training

- 1 Motivation
- 2 BERT — Encoder architecture in detail
- 3 Input and pre-training**
- 4 Pre-training
- 5 Downstream tasks and fine-tuning

BERT: Tokenization

Tokenizing into a multilingual WordPiece inventory

- Recall that WordPiece units are sub-word units
- 30,000 WordPiece units (newer models 110k units, 100 languages)

Implications: BERT can "consume" any language

BERT: Input representation

- Each WordPiece token from the input is represented by a **WordPiece embedding** (randomly initialized)
- Each position from the input is associated with a **positional embedding** (also randomly initialized)
- Input length limited to **512** WordPiece tokens, using `<PAD>`ding
- Special tokens
 - The first token is always a special token **[CLS]**
 - If the task involves two sentences (e.g., NLI), these two sentences are separated by a special token **[SEP]**; also special two **segment position embeddings**

BERT: Input representation summary

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{\# \# ing}$	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

Pre-training

- 1 Motivation
- 2 BERT — Encoder architecture in detail
- 3 Input and pre-training
- 4 Pre-training**
- 5 Downstream tasks and fine-tuning

BERT: Self-supervised multi-task pre-training

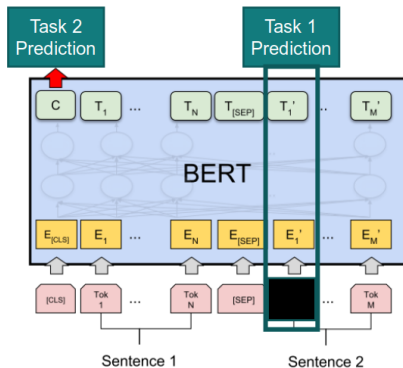
Prepare two auxiliary tasks that need no labeled data

Task 1: Cloze-test task

- Predict the masked WordPiece unit (multi-class, 30k classes)

Task 2: Consecutive segment prediction

- Did the second text segment appeared after the first segment? (binary)



BERT: Pre-training data generation

Take the entire Wikipedia (in 100 languages; 2,5 billion words)

To generate a single training instance, sample two segments (max combined length 512 WordPiece tokens)

- For Task 2, replace the second segment randomly in 50% (negative samples)
- For Task 1, choose random 15% of the tokens, and in 80% replace with a [MASK]

BERT: Pre-training data – Simplified example

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

- <PAD>ding is missing
- The actual segments are longer and not necessarily sentences (just spans)
- The WordPiece tokens match full words here

BERT: pre-training by masked language modeling

```
1: function ETraining( $\{\mathbf{x}_n\}_{n=1}^{N_{\text{data}}}$  seqs,  $\boldsymbol{\theta}$  init. params;  $p_{\text{mask}} \in (0, 1)$ ,  $N_{\text{epochs}}$ ,  $\eta$ )
2:   for  $i \in [N_{\text{epochs}}]$  do
3:     for  $n \in [N_{\text{data}}]$  do
4:        $\ell \leftarrow \text{length}(\mathbf{x}_n)$ 
5:       for  $t \in [\ell]$  do
6:          $\tilde{\mathbf{x}}_n[t] \leftarrow \text{<mask\_token> with prob. } p_{\text{mask}}, \text{ otherwise } \mathbf{x}_n[t]$ 
7:          $\tilde{T} \leftarrow \{t \in [\ell] : \tilde{\mathbf{x}}_n[t] = \text{<mask\_token>}\}$  ▷ Indices of masked
tokens
8:          $\mathbf{P}_{\boldsymbol{\theta}} \leftarrow \text{ETransformer}(\tilde{\mathbf{x}}_n | \boldsymbol{\theta})$ 
9:          $\text{loss}_{\boldsymbol{\theta}} \leftarrow - \sum_{t \in \tilde{T}} \log \mathbf{P}_{\boldsymbol{\theta}}[t, \mathbf{x}_n[t]]$ 
10:         $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla \text{loss}_{\boldsymbol{\theta}}$ 
11:   return  $\boldsymbol{\theta}$ 
```

Simple example explaining lines 6–7 (masking)

$$\begin{pmatrix} \text{The} & \text{cat} & \text{sat} \end{pmatrix} \rightarrow \mathbf{x}_n = \begin{pmatrix} 21 & 11987 & 5438 \end{pmatrix} \quad (\text{Indices in } V)$$

Random masking (index of `<mask_token>` = 50001):

- 1 For $t = 1$, the random outcome is "mask"
- 2 For $t = 2$, the random outcome is "keep"
- 3 For $t = 3$, the random outcome is "mask"

$$\tilde{\mathbf{x}}_n = \begin{pmatrix} 50001 & 11987 & 50001 \end{pmatrix}, \tilde{T} = \{1, 3\}$$

Explaining line 9 (negative log likelihood)

$$\left(\text{The} \quad \text{cat} \quad \text{sat}\right) \rightarrow \mathbf{x}_n = \begin{pmatrix} 21 & 11987 & 5438 \end{pmatrix}, \tilde{\mathbf{x}}_n = \begin{pmatrix} 50001 & 11987 & 50001 \end{pmatrix}, \tilde{T} = \{1, 3\}$$

$$\mathbf{P}_\theta \leftarrow \text{ETransformer}(\tilde{\mathbf{x}}_n | \theta)$$

$$\mathbf{P}_\theta = \begin{pmatrix} 0.001 & 0.0007 & \dots & 0.0003 \\ 0.0013 & 0.0065 & \dots & 0.0001 \\ 0.079 & 0.015 & \dots & 0.0001 \end{pmatrix}$$

$\mathbf{P}_\theta \in (0, 1)^{\ell_x \times N_V}$, where each row of \mathbf{P} is a distribution over the vocabulary

Explaining line 9 (negative log likelihood), $t = 1$

$$\mathbf{x}_n = (21, 11987, 5438), \tilde{\mathbf{x}}_n = (50001, 11987, 50001), \tilde{T} = \{1, 3\}$$

$$\mathbf{P}_\theta = \begin{pmatrix} 0.001 & \dots & 0.0041_{21} & \dots & 0.0003 \\ \vdots & & & & \end{pmatrix}$$

For $t = 1$, the model should learn to predict "The" (index 21)

$$\text{Gold: } \mathbf{y} = (0, 0, \dots, 1_{21}, \dots, 0) \in \mathbb{R}^{N_v}$$

$$\text{Pred: } \hat{\mathbf{y}} = \mathbf{P}_\theta[1, :] = (0.001, \dots, 0.0041_{21}, \dots, 0.0003) \in \mathbb{R}^{N_v}$$

Recall: Categorical cross entropy loss

$$\begin{aligned} L(\hat{\mathbf{y}}, \mathbf{y}) &:= - \sum_{k=1}^K \mathbf{y}_{[k]} \log(\hat{\mathbf{y}}_{[k]}) \\ &= -1 \cdot \log(\hat{\mathbf{y}}[21]) = -\log(\mathbf{P}_\theta[1, 21]) \\ &= -\log(\mathbf{P}_\theta[1, \mathbf{x}_n[1]]) = -\log(\mathbf{P}_\theta[t, \mathbf{x}_n[t]]) \end{aligned}$$

Explaining line 9 (negative log likelihood), $t = 3$

$$\mathbf{x}_n = (21, 11987, 5438), \tilde{\mathbf{x}}_n = (50001, 11987, 50001), \tilde{T} = \{1, 3\}$$

$$\mathbf{P}_\theta = \begin{pmatrix} \vdots & \dots & \dots \end{pmatrix}$$

For $t = 3$, the model should learn to predict "sat" (id 5438)

Categorical cross entropy loss

$$\begin{aligned} L(\hat{\mathbf{y}}, \mathbf{y}) &:= - \sum_{k=1}^K \mathbf{y}_{[k]} \log(\hat{\mathbf{y}}_{[k]}) \\ &= -1 \cdot \log(\hat{\mathbf{y}}[5438]) = -\log(\mathbf{P}_\theta[3, 5438]) = -\log(\mathbf{P}_\theta[t, \mathbf{x}_n[t]]) \end{aligned}$$

Sum over all masked token positions in \tilde{T} gives us line 9:

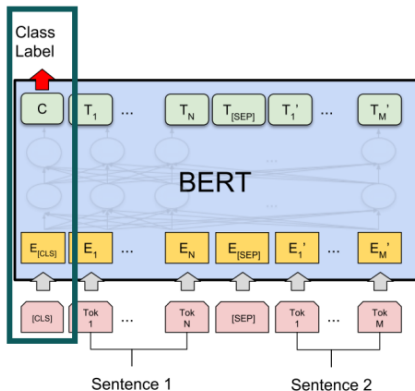
$$\text{loss}_\theta \leftarrow - \sum_{t \in \tilde{T}} \log \mathbf{P}_\theta[t, \mathbf{x}_n[t]]$$

Downstream tasks and fine-tuning

- 1 Motivation
- 2 BERT — Encoder architecture in detail
- 3 Input and pre-training
- 4 Pre-training
- 5 Downstream tasks and fine-tuning**

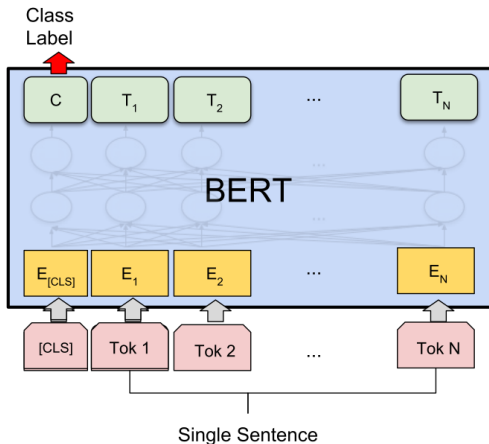
BERT: Representing various NLP tasks

That explains the special [CLS] token at sequence start



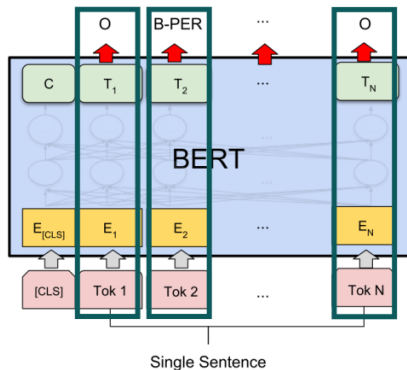
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG

BERT: Representing various NLP tasks



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

BERT: Representing various NLP tasks



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

Not conditioned on surrounding predictions

BERT pre-training time

Pretraining BERT took originally 4 days on 64 TPUs¹

Once pre-trained, transfer and "fine-tune" on your small-data task and get competitive results

P. Izsak, M. Berchansky, and O. Levy (2021). **"How to Train BERT with an Academic Budget"**. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 10644–10652

¹Can be done more efficiently, see, e.g., Izsak, Berchansky, and Levy (2021)

Recap

BERT stays on the shoulders of many clever concepts and techniques, mastered into a single model

What do we know about how BERT works?

“BERTology has clearly come a long way, but it is fair to say we still have more questions than answers about how BERT works.” — Rogers, Kovaleva, and Rumshisky (2020)²

A. Rogers, O. Kovaleva, and A. Rumshisky (2020). **“A Primer in BERTology: What We Know About How BERT Works”**. In: *Transactions of the Association for Computational Linguistics* 8, pp. 842–866

²Highly recommended reading!

License and credits

Licensed under Creative Commons
Attribution-ShareAlike 4.0 International
(CC BY-SA 4.0)



Credits

Ivan Habernal

Content from ACL Anthology papers licensed under CC-BY

<https://www.aclweb.org/anthology>