Natural Language Processing with Deep Learning

Lecture 8 — BERT as encoder-only transformer

Prof. Dr. Ivan Habernal

December 12, 2024

www.trusthlt.org
Trustworthy Human Language Technologies Group (TrustHLT)
Ruhr University Bochum & Research Center 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

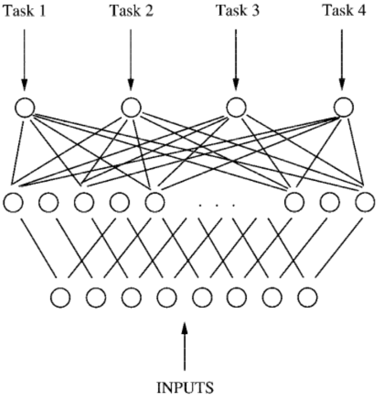
Motivation

Multi-task learning

Multi-task Learning

Approach to inductive transfer that improves generalization

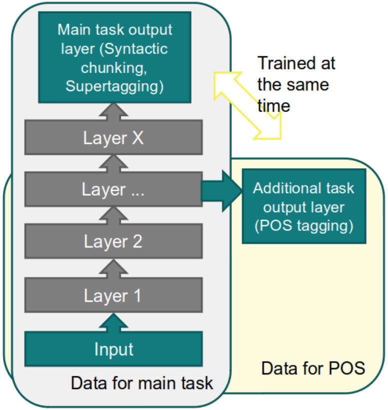
By learning tasks in parallel while using a shared representation



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

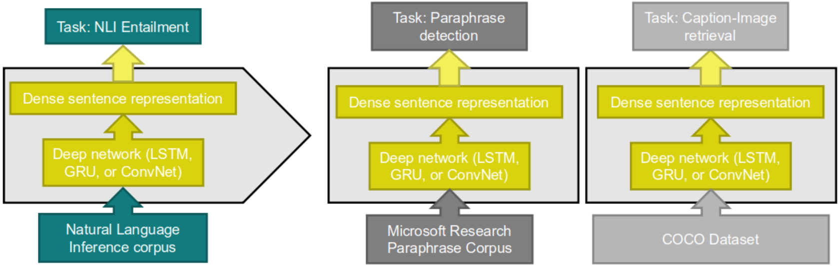
Multi-task learning in NLP

"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**)."

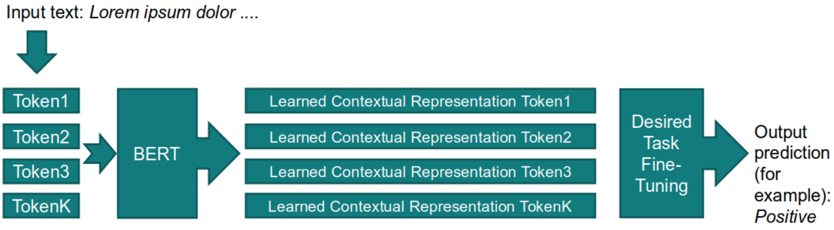
After this lecture you should be able to build BERT

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

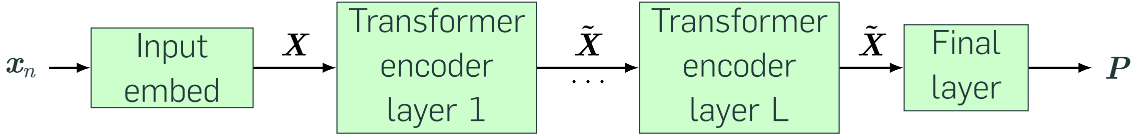
1 Motivation
2 BERT — Encoder architecture in detail
3 Input and pre-training
4 Pre-training
5 Downstream tasks and fine-tuning

BERT: Very abstract view



- 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

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

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

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

simplify to $[N]$

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

BERT (encoding-only transformer, forward pass)

function ETransformer($\boldsymbol{x}; \boldsymbol{\mathcal{W}}$)
    ...

Input:

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

$\boldsymbol{\mathcal{W}}$ — all trainable parameters

Output:

Typically an embedding vector for each input token

Or: $\boldsymbol{P} \in (0,1)^{\ell_\mathsf{x} \times N_\mathsf{V}}$, where each row of $\boldsymbol{P}$ is a distribution over the vocabulary

Input embeddings

The cat sat $\boldsymbol{x}_n = \begin{pmatrix} 21 & 11987 & 5438 \end{pmatrix}$

$$\begin{bmatrix} 21 & 11987 & 5438 \end{bmatrix} \xrightarrow{\boldsymbol{x}_n} \boxed{\text{Input embeddings}} \xrightarrow{\boldsymbol{X}} \begin{bmatrix} \cdots & \boldsymbol{e}_1 & \cdots \\ \cdots & \boldsymbol{e}_2 & \cdots \\ \cdots & \boldsymbol{e}_3 & \cdots \end{bmatrix}$$

Positional embeddings

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

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

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

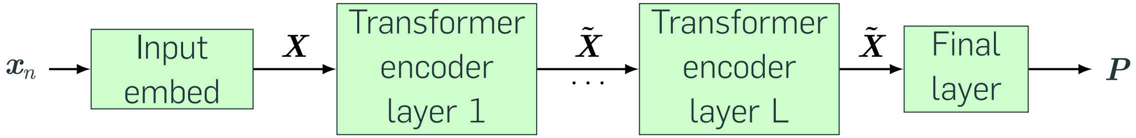
Originally proposed for CNNs for MT, state-of-the-art results and **9.3–21.3**$\times$ **faster** than LSTMs on GPU

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)

function ETransformer$(\boldsymbol{x}; \boldsymbol{\mathcal{W}})$
    $\ell \leftarrow \text{length}(\boldsymbol{x})$
    for $t \in [\ell]$: $\boldsymbol{e}_t \leftarrow \boldsymbol{W_e}[x[t], :] + \boldsymbol{W_p}[t, :]$ ▷ Token emb. + positional emb.
    $\boldsymbol{X} \leftarrow \text{Stack row-wise}[\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots \boldsymbol{e}_\ell]$
    $\ldots$

Transformer encoder (BERT)



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

BERT (encoding-only transformer, forward pass)

function ETransformer($\boldsymbol{x}; \boldsymbol{\mathcal{W}}$)

  $\ell \leftarrow$ length($\boldsymbol{x}$)

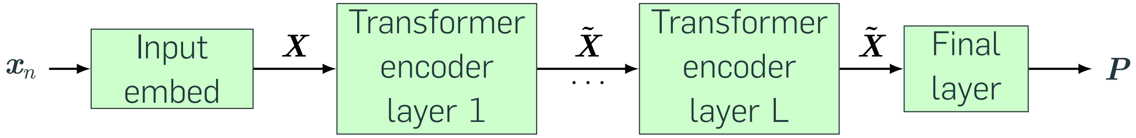
  for $t \in [\ell]$: $\boldsymbol{e}_t \leftarrow \boldsymbol{W_e}[x[t], :] + \boldsymbol{W_p}[t, :]$ $\triangleright$ Token emb. + positional emb.

  $\boldsymbol{X} \leftarrow$ Stack row-wise$[\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots \boldsymbol{e}_\ell]$

  for $l = 1, 2, \ldots, L$ **do**

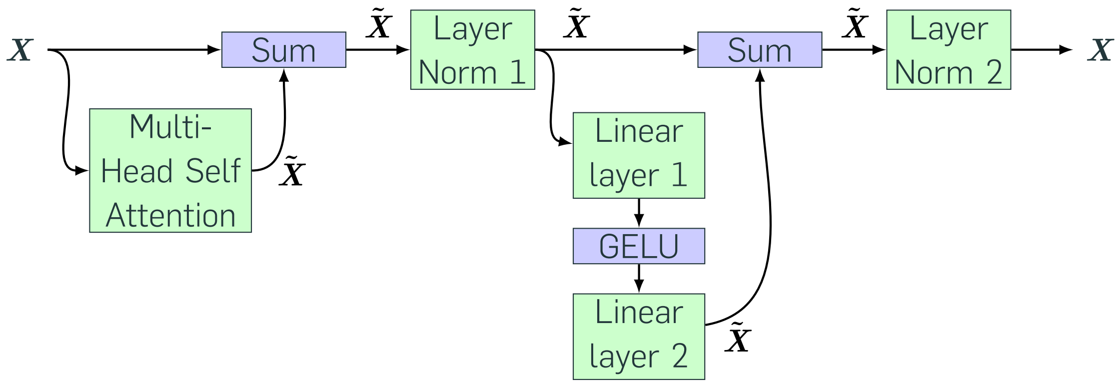
    $\ldots$

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)

function ETransformer($\boldsymbol{x}; \boldsymbol{\mathcal{W}}$)

    $\ell \leftarrow \text{length}(\boldsymbol{x})$

    for $t \in [\ell] : \boldsymbol{e}_t \leftarrow \boldsymbol{W_e}[x[t], :] + \boldsymbol{W_p}[t, :]$ $\quad\triangleright$ Token emb. + positional emb.

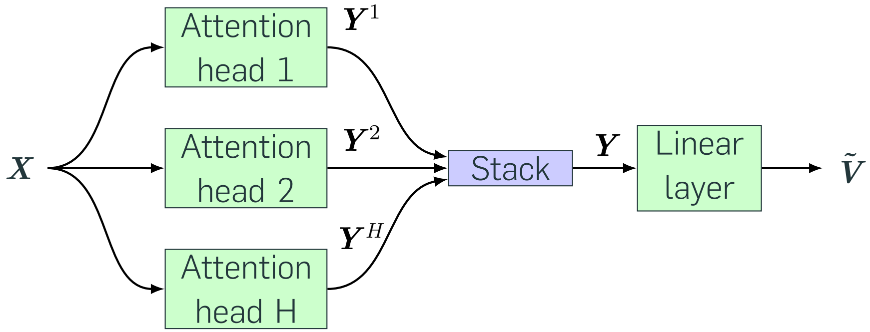
    $\boldsymbol{X} \leftarrow$ Stack row-wise$[\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots \boldsymbol{e}_\ell]$

    for $l = 1, 2, \ldots, L$ **do**

        $\boldsymbol{X} \leftarrow \boldsymbol{X} + \text{MHAttention}(\boldsymbol{X} | \boldsymbol{\mathcal{W}}_l)$ $\quad\triangleright$ Multi-head att., residual conn

        $\ldots$

Multi-head unmasked self-attention



Some notation details

Concatenate matrices of the same dimensions along rows

$$\boldsymbol{Y} = [\boldsymbol{X}^1; \boldsymbol{X}^2; \ldots; \boldsymbol{X}^H] \qquad \boldsymbol{X}^i \in \mathbb{R}^{m \times n} \qquad \boldsymbol{Y} \in \mathbb{R}^{m \times H \cdot n}$$

Example

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

Multi-head bidirectional unmasked self-attention

Input: $\boldsymbol{X} \in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{x}}$, vector representations of the sequence of length ℓ_x

Output: $\tilde{\boldsymbol{V}} \in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{out}}$, updated vector representations of tokens in $\boldsymbol{X}$

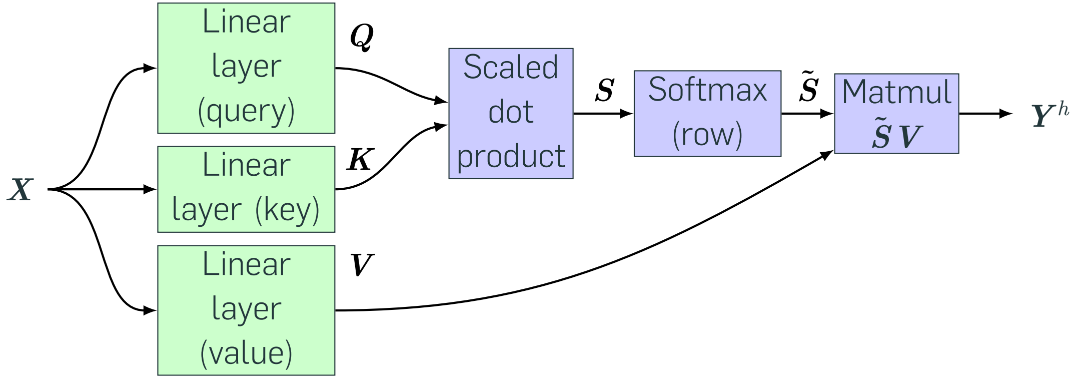
Hyper-param: H, number of attention heads

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

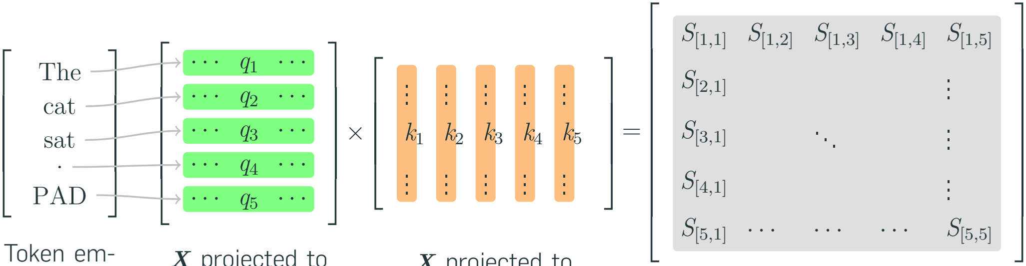
- $\boldsymbol{W}^h_{\boldsymbol{q}}, \boldsymbol{W}^h_{\boldsymbol{k}} \in \mathbb{R}^{d_\mathsf{x} \times d_\mathsf{attn}}$, $\boldsymbol{b}^h_{\boldsymbol{q}}, \boldsymbol{b}^h_{\boldsymbol{k}} \in \mathbb{R}^{d_\mathsf{attn}}$, $\boldsymbol{W}_{\boldsymbol{v}} \in \mathbb{R}^{d_\mathsf{x} \times d_\mathsf{mid}}$, $\boldsymbol{b}_{\boldsymbol{v}} \in \mathbb{R}^{d_\mathsf{mid}}$
- $\boldsymbol{W}_{\boldsymbol{o}} \in \mathbb{R}^{H \cdot d_\mathsf{mid} \times d_\mathsf{out}}$, $\boldsymbol{b}_{\boldsymbol{o}} \in \mathbb{R}^{d_\mathsf{out}}$

function MHAttention($\boldsymbol{X}; \boldsymbol{\mathcal{W}}$)
    for $h \in [H]$ **do**
        $\boldsymbol{Y}^h \leftarrow \text{Attention}(\boldsymbol{X}; \boldsymbol{\mathcal{W}}^h_{\boldsymbol{qkv}})$ $\triangleright$ $\boldsymbol{Y}^h \in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{mid}}$
    $\boldsymbol{Y} \leftarrow [\boldsymbol{Y}^1; \boldsymbol{Y}^2; \ldots; \boldsymbol{Y}^H]$ $\triangleright$ $\boldsymbol{Y} \in \mathbb{R}^{\ell_\mathsf{x} \times H \cdot d_\mathsf{mid}}$
    return $\tilde{\boldsymbol{V}} = \boldsymbol{Y}\boldsymbol{W}_{\boldsymbol{o}} + \boldsymbol{b}_{\boldsymbol{o}}$

Single unmasked self-attention head



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



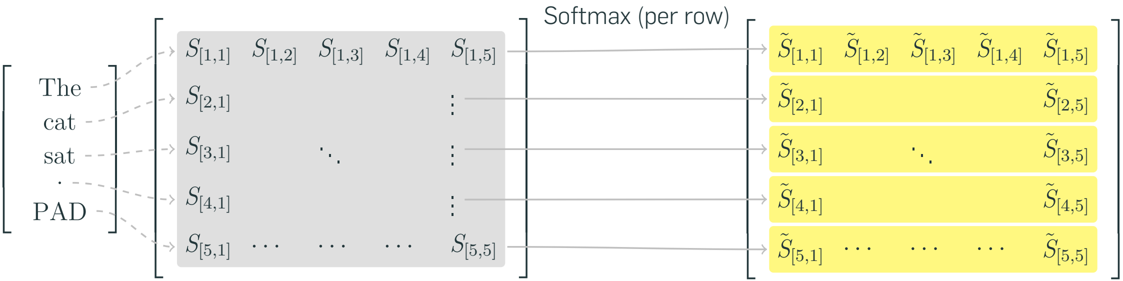
Token embeddings $\boldsymbol{X}$ (input or previous transformer layer)

$\boldsymbol{X}$ projected to query matrix $\boldsymbol{Q}$

$\boldsymbol{X}$ projected to key matrix $\boldsymbol{K}$ and transposed

Scaled dot product $\boldsymbol{S}$ (raw associations) (each entry divided by $\sqrt{d_{\text{attn}}}$)

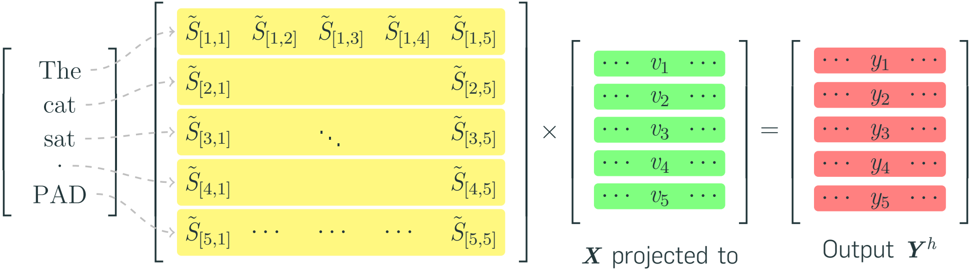
Self-attention in detail — softmax over scaled dot product



Scaled dot product
(raw associations)
(each entry divided by $\sqrt{d_{\text{attn}}}$)

Normalized
associations

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



Normalized associations

$\boldsymbol{X}$ projected to value matrix $\boldsymbol{V}$

Output $\boldsymbol{Y}^h$ of y-th self-attention head

Some notation details

How to add a single vector $\boldsymbol{b}$ to each row in a matrix $\boldsymbol{W}$ ($\boldsymbol{W} \in \mathbb{R}^{m \times n}, \boldsymbol{b} \in \mathbb{R}^{n}$)

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

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

Example

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

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

Some notation details

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

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

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

Bidirectional unmasked self-attention precisely

Input: $\boldsymbol{X} \in \mathbb{R}^{\ell_\mathsf{x} \times d_\mathsf{x}}$, vector representations of the sequence of length ℓ_x

Output: $\tilde{\boldsymbol{V}} \in \mathbb{R}^{\ell_\mathsf{x} \times d_\text{out}}$, updated vector representations of tokens in $\boldsymbol{X}$

Params $\boldsymbol{\mathcal{W}}_{\boldsymbol{qkv}}$: $\boldsymbol{W_q}, \boldsymbol{W_k} \in \mathbb{R}^{d_\mathsf{x} \times d_\text{attn}}$, $\boldsymbol{b_q}, \boldsymbol{b_k} \in \mathbb{R}^{d_\text{attn}}$, $\boldsymbol{W_v} \in \mathbb{R}^{d_\mathsf{x} \times d_\text{out}}$, $\boldsymbol{b_v} \in \mathbb{R}^{d_\text{out}}$

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

2: $\quad \boldsymbol{Q} \leftarrow \boldsymbol{X}\boldsymbol{W_q} +_{\text{(rows)}} \boldsymbol{b_q}$ ▷ Query $\in \mathbb{R}^{\ell_\mathsf{x} \times d_\text{attn}}$

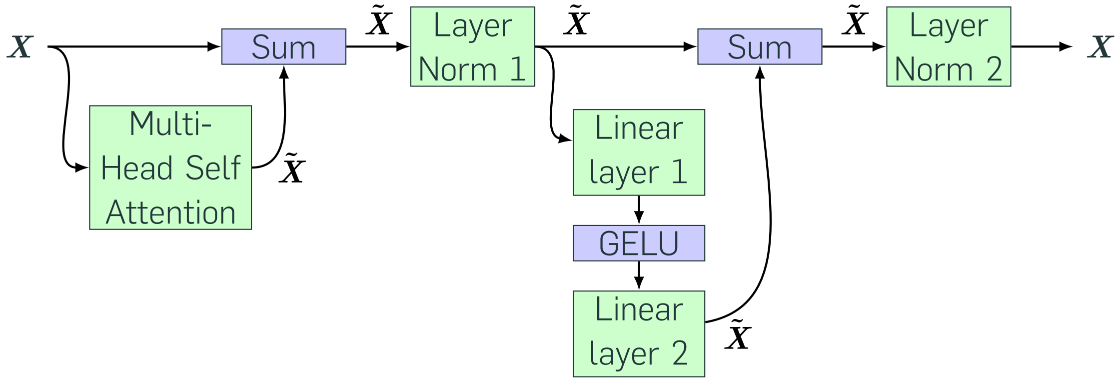
3: $\quad \boldsymbol{K} \leftarrow \boldsymbol{X}\boldsymbol{W_k} +_{\text{(rows)}} \boldsymbol{b_k}$ ▷ Key $\in \mathbb{R}^{\ell_\mathsf{x} \times d_\text{attn}}$

4: $\quad \boldsymbol{V} \leftarrow \boldsymbol{X}\boldsymbol{W_v} +_{\text{(rows)}} \boldsymbol{b_v}$ ▷ Value $\in \mathbb{R}^{\ell_\mathsf{x} \times d_\text{out}}$

5: $\quad \boldsymbol{S} \leftarrow \frac{1}{\sqrt{d_\text{attn}}}(\boldsymbol{Q}\boldsymbol{K}^\top)$ ▷ Scaled score $\in \mathbb{R}^{\ell_\mathsf{x} \times \ell_\mathsf{x}}$

6: $\quad$ **return** $\tilde{\boldsymbol{V}} = \mathrm{softmax}_\text{row}(\boldsymbol{S})\,\boldsymbol{V}$

Transformer encoder layer (BERT)



Let's look at Layer Normalization and GELU

Layer normalization

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

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

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

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

(some transformers use $m = \beta = 0$)

Simplifying notation: Perform LayerNorm on each row

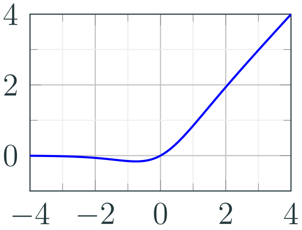
function LayerNormEachRow($\boldsymbol{X} \in \mathbb{R}^{m \times n} | \boldsymbol{\gamma}, \boldsymbol{\beta}$)
    for $t \in [m]$ **do**
        $\boldsymbol{X}[t, :] \leftarrow$ LayerNorm($\boldsymbol{X}[t, :] | \boldsymbol{\gamma}, \boldsymbol{\beta}$)
    return $\boldsymbol{X}$

GELU — Gaussian Error Linear Units

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

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

$$\begin{aligned} \mathrm{GELU}(x) &= x \cdot \Phi(x) \\ &\approx x \cdot \sigma(1.702x) \qquad \text{(if speed > exactness)} \end{aligned}$$



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

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

BERT (encoding-only transformer, forward pass)

function ETransformer($\boldsymbol{x}; \boldsymbol{\mathcal{W}}$)

$\quad \ell \leftarrow \text{length}(\boldsymbol{x})$

$\quad$ for $t \in [\ell]$: $\boldsymbol{e}_t \leftarrow \boldsymbol{W_e}[x[t], :] + \boldsymbol{W_p}[t, :]$ $\triangleright$ Token emb. + positional emb.

$\quad \boldsymbol{X} \leftarrow \text{Stack row-wise}[\boldsymbol{e}_1, \boldsymbol{e}_2, \dots \boldsymbol{e}_\ell]$

$\quad$ **for** $l = 1, 2, \dots, L$ **do**

$\qquad \boldsymbol{X} \leftarrow \boldsymbol{X} + \text{MHAttention}(\boldsymbol{X} | \boldsymbol{\mathcal{W}}_l)$ $\triangleright$ Multi-head att., residual conn

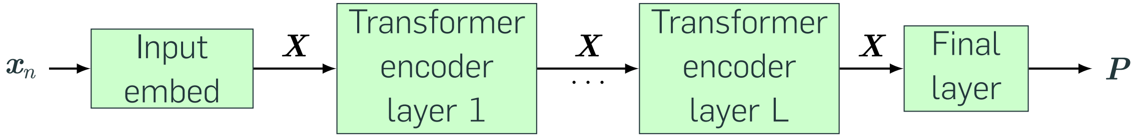
$\qquad \boldsymbol{X} \leftarrow \text{LayerNormPerRow}(\boldsymbol{X} | \boldsymbol{\gamma}_l^{\mathbf{1}}, \boldsymbol{\beta}_l^{\mathbf{1}})$

$\qquad \boldsymbol{X} \leftarrow \boldsymbol{X} + \left(\text{GELU}(\boldsymbol{X} \boldsymbol{W}_l^{\text{mlp1}} +_{(\text{row})} \boldsymbol{b}_l^{\text{mlp1}}) \boldsymbol{W}_l^{\text{mlp2}} +_{(\text{row})} \boldsymbol{b}_l^{\text{mlp2}}\right)$ $\triangleright$ MLP

$\qquad \boldsymbol{X} \leftarrow \text{LayerNormPerRow}(\boldsymbol{X} | \boldsymbol{\gamma}_l^{\mathbf{2}}, \boldsymbol{\beta}_l^{\mathbf{2}})$

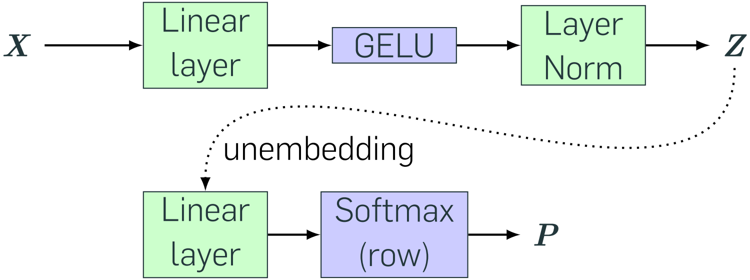
$\quad \dots$

Transformer encoder (BERT)



Let's look at the final layers

Final layer (BERT)



BERT (encoding-only transformer, forward pass)

function ETransformer($\boldsymbol{x}; \boldsymbol{\mathcal{W}}$)

  $\ell \leftarrow$ length($\boldsymbol{x}$)

  for $t \in [\ell]$: $\boldsymbol{e}_t \leftarrow \boldsymbol{W_e}[x[t], :] + \boldsymbol{W_p}[t, :]$ ▷ Token emb. + positional emb.

  $\boldsymbol{X} \leftarrow$ Stack row-wise$[\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots \boldsymbol{e}_\ell]$

  for $l = 1, 2, \ldots, L$ **do**

    $\boldsymbol{X} \leftarrow \boldsymbol{X} +$ MHAttention($\boldsymbol{X} | \boldsymbol{\mathcal{W}}_l$) ▷ Multi-head att., residual conn

    $\boldsymbol{X} \leftarrow$ LayerNormPerRow($\boldsymbol{X} | \boldsymbol{\gamma}_l^{\boldsymbol{1}}, \boldsymbol{\beta}_l^{\boldsymbol{1}}$)

    $\boldsymbol{X} \leftarrow \boldsymbol{X} + \left(\text{GELU}(\boldsymbol{X} \boldsymbol{W}_l^{\text{mlp1}} +_{\text{(row)}} \boldsymbol{b}_l^{\text{mlp1}}) \boldsymbol{W}_l^{\text{mlp2}} +_{\text{(row)}} \boldsymbol{b}_l^{\text{mlp2}} \right)$ ▷ MLP

    $\boldsymbol{X} \leftarrow$ LayerNormPerRow($\boldsymbol{X} | \boldsymbol{\gamma}_l^{\boldsymbol{2}}, \boldsymbol{\beta}_l^{\boldsymbol{2}}$)

  $\boldsymbol{X} \leftarrow$ GELU($\boldsymbol{X} \boldsymbol{W_f} +_{\text{(row)}} \boldsymbol{b_f}$)

  $\boldsymbol{X} \leftarrow$ LayerNormPerRow($\boldsymbol{X} | \boldsymbol{\gamma}_l, \boldsymbol{\beta}_l$)

  return $\boldsymbol{P} = \mathrm{softmax}(\boldsymbol{X} \boldsymbol{W_u})$ ▷ Project to vocab., probabilities

BERT parameters and hyperparameters

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

Parameters:

$\boldsymbol{W_e} \in \mathbb{R}^{N_{\text{V}} \times d_{\text{e}}}$, $\boldsymbol{W_p} \in \mathbb{R}^{\ell_{\text{max}} \times d_{\text{e}}}$, the token and positional embedding matrices

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

- $\boldsymbol{\gamma}_l^{\boldsymbol{1}}, \boldsymbol{\beta}_l^{\boldsymbol{1}}, \boldsymbol{\gamma}_l^{\boldsymbol{2}}, \boldsymbol{\beta}_l^{\boldsymbol{2}}$, two sets of layer-norm parameters
- $\boldsymbol{W}_l^{\text{mlp1}} \in \mathbb{R}^{d_{\text{e}} \times d_{\text{mlp}}}$, $\boldsymbol{b}_l^{\text{mlp1}} \in \mathbb{R}^{d_{\text{mlp}}}$
- $\boldsymbol{W}_l^{\text{mlp2}} \in \mathbb{R}^{d_{\text{mlp}} \times d_{\text{e}}}$, $\boldsymbol{b}_l^{\text{mlp2}} \in \mathbb{R}^{d_{\text{e}}}$

$\boldsymbol{W_f} \in \mathbb{R}^{d_{\text{e}} \times d_{\text{f}}}$, $\boldsymbol{b_f} \in \mathbb{R}^{d_{\text{f}}}$, $\boldsymbol{\gamma}, \boldsymbol{\beta} \in \mathbb{R}^{d_{\text{f}}}$, the final linear projection and layer-norm parameters.

$\boldsymbol{W_u} \in \mathbb{R}^{d_{\text{e}} \times N_{\text{V}}}$, 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

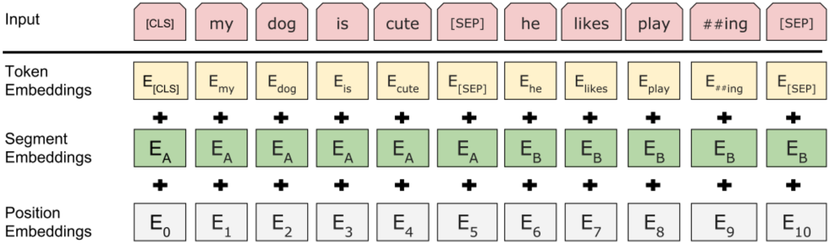
- Recall that WordPiece units are sub-word units
- 30,000 WordPiece units (newer models 110k units, 100 languages)

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 fist 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



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

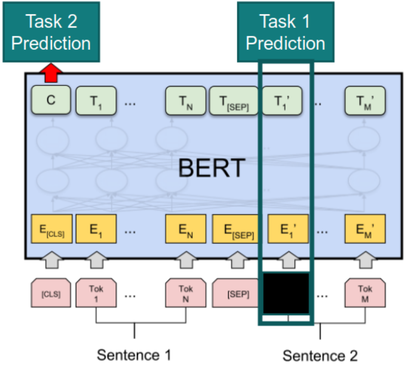
Prepare two auxiliary tasks that need no labeled data

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

function ETraining($\{\boldsymbol{x}_n\}_{n=1}^{N_\text{data}}$ seqs, $\boldsymbol{\theta}$ init. params; $p_\text{mask} \in (0, 1)$, N_epochs, η)
  for $i \in [N_\text{epochs}]$ **do**
    for $n \in [N_\text{data}]$ **do**
      $\ell \leftarrow \text{length}(\boldsymbol{x}_n)$
      for $t \in [\ell]$ **do**
        $\tilde{\boldsymbol{x}}_n[t] \leftarrow$ <mask_token> with prob. p_mask, otherwise $\boldsymbol{x}_n[t]$
      $\tilde{T} \leftarrow \{t \in [\ell] : \tilde{\boldsymbol{x}}_n[t] =$ <mask_token>$\}$ ▷ Indices of masked tokens
      $\boldsymbol{P}_{\boldsymbol{\theta}} \leftarrow \text{ETransformer}(\tilde{\boldsymbol{x}}_n | \boldsymbol{\theta})$
      $\text{loss}_{\boldsymbol{\theta}} \leftarrow -\sum_{t \in \tilde{T}} \log \boldsymbol{P}_{\boldsymbol{\theta}}[t, \boldsymbol{x}_n[t]]$
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla \text{loss}_{\boldsymbol{\theta}}$
  return $\boldsymbol{\theta}$

Simple example explaining lines 6–7 (masking)

$\begin{pmatrix}\text{The} & \text{cat} & \text{sat}\end{pmatrix} \to \boldsymbol{x}_n =$

$\begin{pmatrix}21 & 11987 & 5438\end{pmatrix}$ (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"

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

Explaining line 9 (negative log likelihood)

$\begin{pmatrix}\text{The} & \text{cat} & \text{sat}\end{pmatrix} \to \boldsymbol{x}_n = \begin{pmatrix}21 & 11987 & 5438\end{pmatrix}, \tilde{\boldsymbol{x}}_n = \begin{pmatrix}50001 & 11987 & 50001\end{pmatrix}, \tilde{T} = \{1, 3\}$

$\boldsymbol{P_\theta} \leftarrow \text{ETransformer}(\tilde{\boldsymbol{x}}_n | \boldsymbol{\theta})$

$$\boldsymbol{P_\theta} = \begin{pmatrix} 0.001 & 0.0007 & \dots & 0.0003 \\ 0.0013 & 0.0065 & \dots & 0.0001 \\ 0.079 & 0.015 & \dots & 0.0001 \end{pmatrix}$$

$\boldsymbol{P_\theta} \in (0, 1)^{\ell_\mathsf{X} \times N_\mathsf{V}}$, where each row of $\boldsymbol{P}$ is a distribution over the vocabulary

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

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

$$\boldsymbol{P_\theta} = \begin{pmatrix} 0.001 & \ldots & 0.0041_{21} & \ldots 0.0003 \\ \vdots & & & \end{pmatrix}$$

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

Gold: $\boldsymbol{y} = (0, 0, \ldots, 1_{21}, \ldots, 0) \in \mathbb{R}^{N_V}$

Pred: $\hat{\boldsymbol{y}} = \boldsymbol{P_\theta}[1, :] = (0.001, \ldots, 0.0041_{21}, \ldots 0.0003) \in \mathbb{R}^{N_V}$

Recall: Categorical cross entropy loss

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) := -\sum_{k=1}^{K} \boldsymbol{y}_{[k]} \log\left(\hat{\boldsymbol{y}}_{[k]}\right)$$
$$= -1 \cdot \log(\hat{\boldsymbol{y}}[21]) = -\log(\boldsymbol{P_\theta}[1, 21])$$
$$= -\log(\boldsymbol{P_\theta}[1, \boldsymbol{x}_n[1]]) = -\log(\boldsymbol{P_\theta}[t, \boldsymbol{x}_n[t]])$$

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

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

$\boldsymbol{P_\theta} = \begin{pmatrix} \vdots & \ldots & \ldots & \end{pmatrix}$

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

Categorical cross entropy loss

$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) := -\sum_{k=1}^{K} \boldsymbol{y}_{[k]} \log\left(\hat{\boldsymbol{y}}_{[k]}\right)$
$= -1 \cdot \log(\hat{\boldsymbol{y}}[5438]) = -\log(\boldsymbol{P_\theta}[3, 5438]) = -\log(\boldsymbol{P_\theta}[t, \boldsymbol{x}_n[t]])$

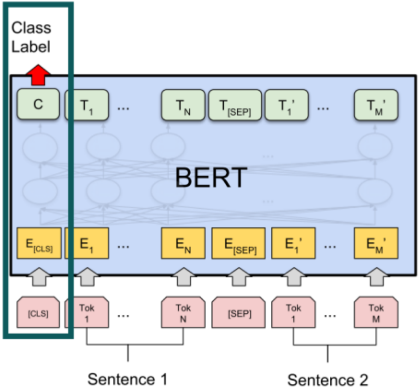
Sum over all masked token positions in $\tilde{T}$ gives us line 9:

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

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

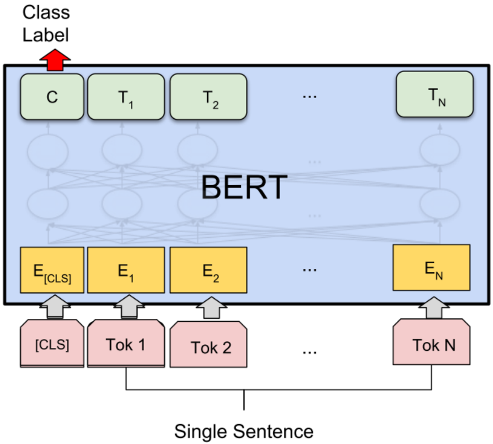
BERT: Representing various NLP tasks



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

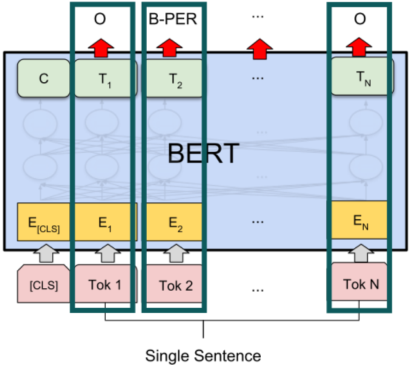
That explains the special [CLS] token at sequence start

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 TPUs[1]

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

[1] Can 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

[2]Highly recommended reading!

License and credits

Credits

Ivan Habernal