Using the Transformer

Post-BERT architectures

Learning goals

- Understand the developments of the post-BERT era
- Get to know different self-supervised objectives
- Understand how to tackle BERTs critical shortcomings

SUCCESSORS OF BERT

October 2018 - BERT

BERT (**Devlin et al., 2018**) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

10/2018

SUCCESSORS OF BERT

October 2018 - BERT

BERT (**Devlin et al., 2018**) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

February 2019 - GPT2

Radford et al., 2019 massively scale up their GPT model from 2018 (up to 1.5 billion parameters). Despite the size, there are only smaller architectural changer, thus it remains a **unidirectionally contextual** model.

Controversial debate about this model, since OpenAI (at first) refuses to make their pre-trained architecture publicly available due to concerns about “malicious applications”.

SUCCESSORS OF BERT

October 2018 - BERT

BERT (**Devlin et al., 2018**) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

June 2019 - XLNet

Yang et al., 2019 design a new pre-training objective in order to overcome some weaknesses they spotted in the one used by BERT.

The use **Permutation Language Modelling** to avoid the discrepancy between pre-training and fine-tuning introduced by the artificial MASK token.

10/2018
02/2019
06/2019

February 2019 - GPT2

Radford et al., 2019 massively scale up their GPT model from 2018 (up to 1.5 billion parameters). Despite the size, there are only smaller architectural changer, thus it remains a **unidirectionally contextual** model.

Controversial debate about this model, since OpenAI (at first) refuses to make their pre-trained architecture publicly available due to concerns about “malicious applications”.

SUCCESSORS OF BERT

October 2018 - BERT

BERT (**Devlin et al., 2018**) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

June 2019 - XLNet

Yang et al., 2019 design a new pre-training objective in order to overcome some weaknesses they spotted in the one used by BERT.

The use **Permutation Language Modelling** to avoid the discrepancy between pre-training and fine-tuning introduced by the artificial MASK token.

10/2018
02/2019
06/2019
10/2019

February 2019 - GPT2

Radford et al., 2019 massively scale up their GPT model from 2018 (up to 1.5 billion parameters). Despite the size, there are only smaller architectural changer, thus it remains a **unidirectionally contextual** model.

Controversial debate about this model, since OpenAI (at first) refuses to make their pre-trained architecture publicly available due to concerns about “malicious applications”.

October 2019 - T5

T5 (**Raffel et al., 2019**) a complete **encoder-decoder** Transformer based architecture (**text-to-text transfer transformer**).

They approach transfer learning by transforming all inputs as well as all outputs to strings and fine-tuned their model simultaneously on data sets with multiple different tasks.

SUCCESSORS OF BERT

October 2018 - BERT

BERT (**Devlin et al., 2018**) is a bidirectional contextual embedding model purely relying on Self-Attention by using multiple **Transformer encoder** blocks.

BERT (and its successors) rely on the **Masked Language Modelling objective** during pre-training on huge unlabelled corpora of text.

February 2019 - GPT2

Radford et al., 2019 massively scale up their GPT model from 2018 (up to 1.5 billion parameters). Despite the size, there are only smaller architectural changer, thus it remains a **unidirectionally contextual** model.

Controversial debate about this model, since OpenAI (at first) refuses to make their pre-trained architecture publicly available due to concerns about “malicious applications”.

June 2019 - XLNet

Yang et al., 2019 design a new pre-training objective in order to overcome some weaknesses they spotted in the one used by BERT.

The use **Permutation Language Modelling** to avoid the discrepancy between pre-training and fine-tuning introduced by the artificial MASK token.

October 2019 - T5

T5 (**Raffel et al., 2019**) a complete **encoder-decoder** Transformer based architecture (**text-to-text transfer transformer**).

They approach transfer learning by transforming all inputs as well as all outputs to strings and fine-tuned their model simultaneously on data sets with multiple different tasks.

March 2020 - ELECTRA

ELECTRA (**Clark et al., 2020**) introduces a **discriminative pre-training strategy**, allowing for a more efficient use of the pre-training corpus.

Despite requiring two models, the computational costs, for achieving a similar performance, are reduced to this gain in efficiency.

DIFFERENT PRE-TRAINING REGIMES (1)

ELECTRA ▸ Clark et al, 2020

- ELECTRA consinsts of two separate models

→ (Small) generator model G + (large) Discriminator model D

→ This might resemble a GAN setup, but they are not trained in an adversarial manner

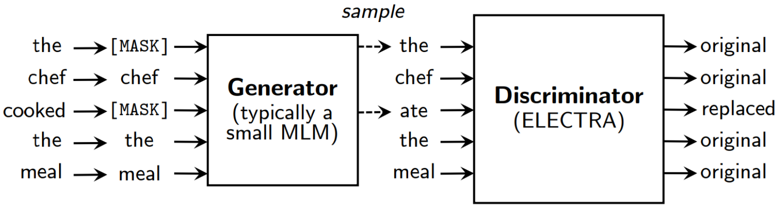
- Generator task: Masked language modeling
- Discriminator task: *Replaced token detection*

→ Predict for each token, whether it is "original" or produced by G

- ELECTRA learns from *all* of the tokens (not just from a small portion of 15%, like e.g. BERT)

ELECTRA – ARCHITECTURE

Joint pre-training:



▸ Source: Clark et al, 2020

- *G* and *D* are (Transformer) encoders which are trained jointly
- *G* replaces `[MASK]`s in an input sequence
  → Passes corrupted input sequence $\vec{x}^{corrupt}$ to *D*

ELECTRA – TRAINING DETAILS

Joint pre-training:

- Generation of samples:

$$m_i \sim \text{unif}\{1, n\} \text{ for } i = 1 \text{ to } k \qquad \boldsymbol{x}^{\text{masked}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \texttt{[MASK]})$$

$$\hat{x}_i \sim p_G(x_i | \boldsymbol{x}^{\text{masked}}) \text{ for } i \in \boldsymbol{m} \qquad \boldsymbol{x}^{\text{corrupt}} = \text{REPLACE}(\boldsymbol{x}, \boldsymbol{m}, \hat{\boldsymbol{x}})$$

  with approx. 15% of the tokens masked out (via choice of *k*)

- *D* predicts whether $x_t,\ t \in 1, \ldots, T$ is "*real*" or generated by *G*
  - Softmax output layer for *G* (probability distr. over all words)
  - Sigmoid output layer for *D* (Binary classification real vs. generated)

ELECTRA – TRAINING DETAILS

Using the masked & corrupted input sequences, the (joint) loss can be written down as follows:

Loss functions:

$$\mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) = \mathbb{E}\left(\sum_{i \in \boldsymbol{m}} -\log p_G(x_i | \boldsymbol{x}^{\text{masked}})\right)$$

$$\mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D) = \mathbb{E}\left(\sum_{t=1}^{n} -\mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\boldsymbol{x}^{\text{corrupt}}, t) - \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\boldsymbol{x}^{\text{corrupt}}, t))\right)$$

Combined:

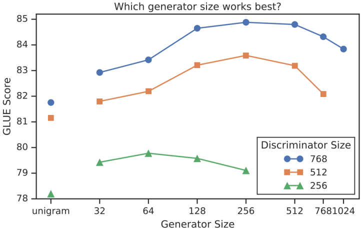
$$\min_{\theta_G, \theta_D} \sum_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) + \lambda \mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D)$$

with λ set to 50, since the discriminator's loss is typically much lower than the geneator's.

ELECTRA – TRAINING DETAILS

Generator size:

- Same size of *G* and *D*:
    - Twice the compute per training step + too challenging for *D*
- Smaller Generators are preferable (1/4 – 1/2 the size of *D*)

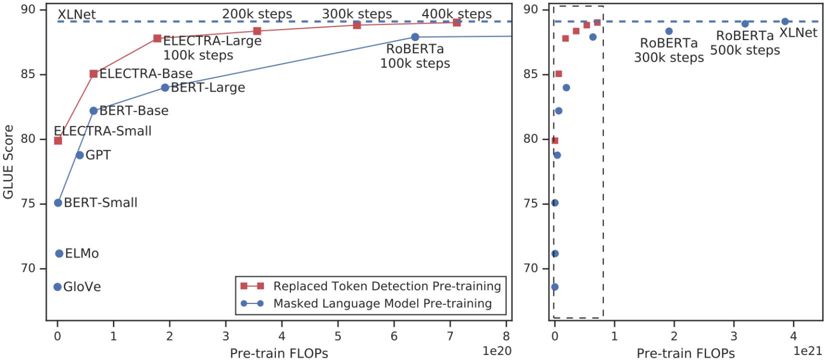


▸ Source: Clark et al, 2020

Weight sharing (experimental):

- Same size of *G* and *D*: All weights can be tied
- *G* smaller than *D*: Share token & positional embeddings

ELECTRA – MODEL COMPARISON

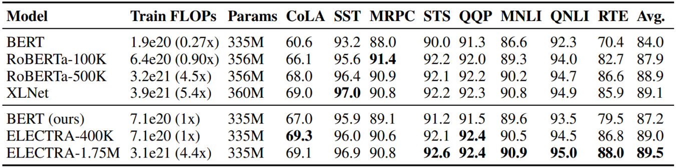


▸ Source: Clark et al, 2020

Note: Different batch sizes (2k vs. 8k) for ELECTRA vs. RoBERTa/XLNet explain why same number of steps lead to approx. 1/4 of the compute for ELECTRA.

ELECTRA – SOTA PERFORMANCE

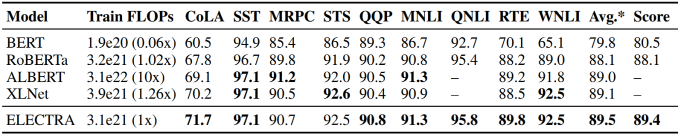
Performance differences vs. BERT/RoBERTa (GLUE dev set):



Model	Train FLOPs	Params	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	Avg.
BERT	1.9e20 (0.27x)	335M	60.6	93.2	88.0	90.0	91.3	86.6	92.3	70.4	84.0
RoBERTa-100K	6.4e20 (0.90x)	356M	66.1	95.6	**91.4**	92.2	92.0	89.3	94.0	82.7	87.9
RoBERTa-500K	3.2e21 (4.5x)	356M	68.0	96.4	90.9	92.1	92.2	90.2	94.7	86.6	88.9
XLNet	3.9e21 (5.4x)	360M	69.0	**97.0**	90.8	92.2	92.3	90.8	94.9	85.9	89.1
BERT (ours)	7.1e20 (1x)	335M	67.0	95.9	89.1	91.2	91.5	89.6	93.5	79.5	87.2
ELECTRA-400K	7.1e20 (1x)	335M	**69.3**	96.0	90.6	92.1	**92.4**	90.5	94.5	86.8	89.0
ELECTRA-1.75M	3.1e21 (4.4x)	335M	69.1	96.9	90.8	**92.6**	**92.4**	**90.9**	**95.0**	**88.0**	**89.5**

‣ Source: Clark et al, 2020

SOTA performance (GLUE test set):



Model	Train FLOPs	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	WNLI	Avg.*	Score
BERT	1.9e20 (0.06x)	60.5	94.9	85.4	86.5	89.3	86.7	92.7	70.1	65.1	79.8	80.5
RoBERTa	3.2e21 (1.02x)	67.8	96.7	89.8	91.9	90.2	90.8	95.4	88.2	89.0	88.1	88.1
ALBERT	3.1e22 (10x)	69.1	**97.1**	**91.2**	92.0	90.5	**91.3**	–	89.2	91.8	89.0	–
XLNet	3.9e21 (1.26x)	70.2	**97.1**	90.5	**92.6**	90.4	90.9	–	88.5	**92.5**	89.1	–
ELECTRA	3.1e21 (1x)	**71.7**	**97.1**	90.7	92.5	**90.8**	**91.3**	**95.8**	**89.8**	**92.5**	**89.5**	**89.4**

* Avg. excluding QNLI to ensure comparability

‣ Source: Clark et al, 2020

DIFFERENT PRE-TRAINING REGIMES (2)

Autoregressive (AR) language modeling

- Factorizes likelihood to

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t|\mathbf{x}< t)$$

- Only uni-directional (backward factorization also possible)

vs. (Denoising) Autoencoding (AE)

- Reconstruct masked tokens $\bar{\mathbf{x}}$ from corrupted sequence $\hat{\mathbf{x}}$:

$$p(\bar{\mathbf{x}}|\hat{\mathbf{x}}) = \prod_{t=1}^{T} m_t \cdot p(x_t|\hat{\mathbf{x}}), \text{with } m_t \text{ as masking indicator}$$

- Possible "*corruptions*" (besides MLM):

  → Replacing words with predictions (cf. ELECTRA)

  → Shuffling tokens or dropping them

DIFFERENT PRE-TRAINING REGIMES (2)

Conceptual Differences

- **AR:** No corruption of input sequences required
- “Causal“ structure in AR approach (sometimes needed)
- **AE:** Assumes independence between corrupted tokens

  Question: Why?
- AR approach only conditions on left side context

  → No bidirectionality possible

PERMUTATION LANGUAGE MODELING (PLM)

XLNet ▸ Yang et al., 2019

- Alternative objective function
- Solves the pretrain-finetune discrepancy

→ No artificial `[MASK]` token is introduced

- Allows for bidirectionality while keeping an AR objective

→ Best of both worlds

- Consists of two "*streams*" of the Attention mechanism
  - Content-stream attention
  - Query-stream attention

PERMUTATION LANGUAGE MODELING (PLM)

Manipulating the factorization order

- Consider permutations $\mathbf{z}$ of the index sequence $[1, 2, \ldots, T]$

→ Used to permute the factorization order, *not* the sequence order.

- Original order of the sequence is retained via positional encodings
- PLM objective (with $\mathcal{Z}_T$ as set of all possible permutations):

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z} \sim \mathcal{Z}_T} \left[\sum_{t=1}^{T} \log p_\theta(x_{z_t} | \mathbf{x}_{\mathbf{z}_{<t}}) \right]$$

PLM – CONTENT- VS. QUERY-STREAM

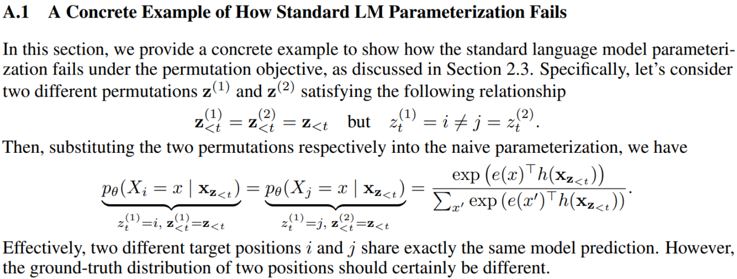
Content-stream

- “Ordinary“ Self-Attention (with special attention masks)
  $\rightarrow$ Attentions masks depend on the factorization order
- Information about the position in the sequence is lost, see ▸ Example in A.1 (Yang et al., 2019)
- Sets of queries (Q), keys (K) and values (V) from content stream*
- Yields a *content embedding* denoted as $h_\theta(\mathbf{x}_{\mathbf{z}_{\leq t}})$

*For a nice visual disentanglement, see ▸ Figures in A.7 (Yang et al., 2019)

PLM – CONTENT- VS. QUERY-STREAM

Content-stream



A.1 A Concrete Example of How Standard LM Parameterization Fails

In this section, we provide a concrete example to show how the standard language model parameterization fails under the permutation objective, as discussed in Section 2.3. Specifically, let's consider two different permutations $\mathbf{z}^{(1)}$ and $\mathbf{z}^{(2)}$ satisfying the following relationship

$$\mathbf{z}^{(1)}_{<t} = \mathbf{z}^{(2)}_{<t} = \mathbf{z}_{<t} \quad \text{but} \quad z^{(1)}_t = i \neq j = z^{(2)}_t.$$

Then, substituting the two permutations respectively into the naive parameterization, we have

$$\underbrace{p_\theta(X_i = x \mid \mathbf{x}_{\mathbf{z}_{<t}})}_{z^{(1)}_t = i,\ \mathbf{z}^{(1)}_{<t} = \mathbf{z}_{<t}} = \underbrace{p_\theta(X_j = x \mid \mathbf{x}_{\mathbf{z}_{<t}})}_{z^{(1)}_t = j,\ \mathbf{z}^{(2)}_{<t} = \mathbf{z}_{<t}} = \frac{\exp\left(e(x)^\top h(\mathbf{x}_{\mathbf{z}_{<t}})\right)}{\sum_{x'} \exp\left(e(x')^\top h(\mathbf{x}_{\mathbf{z}_{<t}})\right)}.$$

Effectively, two different target positions i and j share exactly the same model prediction. However, the ground-truth distribution of two positions should certainly be different.

▸ Source: Yang et al., 2019

PLM – CONTENT- VS. QUERY-STREAM

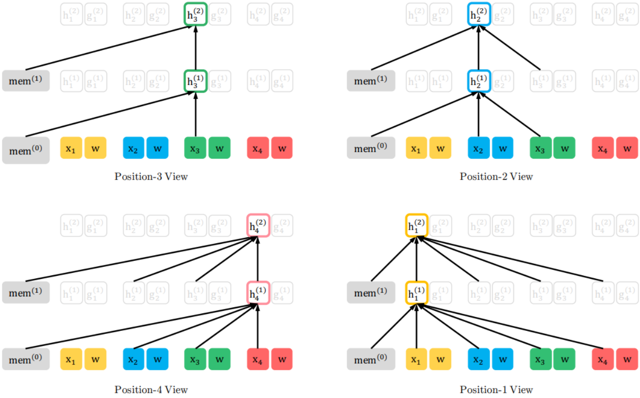
Query-stream

- Access to context through content-stream, but no access to the content of the current position (only location information)
- Q from the query stream, K and V from the content stream*
- Yields a *query embedding* denoted as $g_\theta(\mathbf{x}_{\mathbf{z}_{<t}}, z_t)$

*For a nice visual disentanglement, see ▸ Figures in A.7 (Yang et al., 2019)

PLM – CONTENT- VS. QUERY-STREAM

Content-stream


$h_1^{(2)}$ $g_1^{(2)}$ $h_2^{(2)}$ $g_2^{(2)}$ $h_3^{(2)}$ $g_3^{(2)}$ $h_4^{(2)}$ $g_4^{(2)}$
mem$^{(1)}$
$h_1^{(1)}$ $g_1^{(1)}$ $h_2^{(1)}$ $g_2^{(1)}$ $h_3^{(1)}$ $g_3^{(1)}$ $h_4^{(1)}$ $g_4^{(1)}$
mem$^{(0)}$
x_1 w x_2 w x_3 w x_4 w

Position-3 View

Position-2 View

Position-4 View

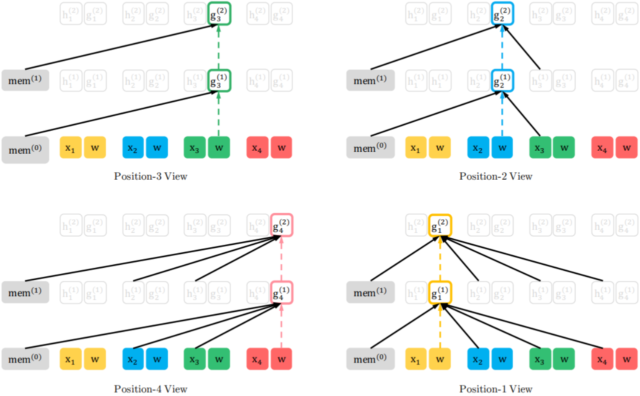
Position-1 View

Split View of the Content Stream
(Factorization order: 3 → 2 → 4 → 1)

▸ Source: Yang et al., 2019

PLM – CONTENT- VS. QUERY-STREAM

Query-stream

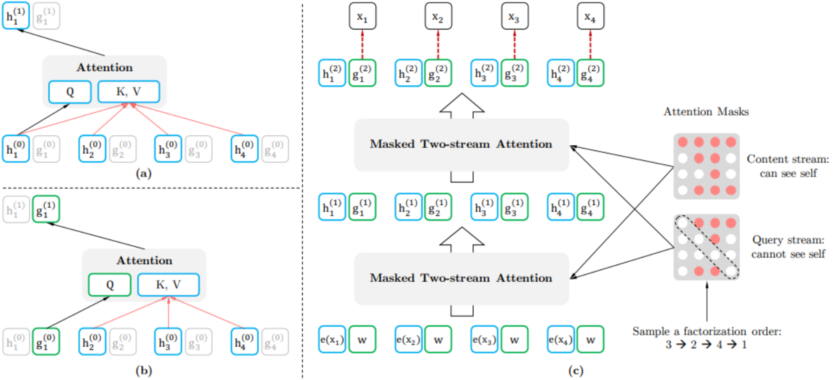


$h_1^{(2)}$ $g_1^{(2)}$ $h_2^{(2)}$ $g_2^{(2)}$ $h_3^{(2)}$ $g_3^{(2)}$ $h_4^{(2)}$ $g_4^{(2)}$
mem$^{(1)}$
$h_1^{(1)}$ $g_1^{(1)}$ $h_2^{(1)}$ $g_2^{(1)}$ $h_3^{(1)}$ $g_3^{(1)}$ $h_4^{(1)}$ $g_4^{(1)}$
mem$^{(0)}$
x_1 w x_2 w x_3 w x_4 w
Position-3 View
Position-2 View
Position-4 View
Position-1 View

Split View of the Query Stream
(Factorization order: 3 → 2 → 4 → 1)

Source: Yang et al., 2019

PLM – CONTENT- VS. QUERY-STREAM


$h_1^{(1)}$ $g_1^{(1)}$
Attention
Q
K, V
$h_1^{(0)}$ $g_1^{(0)}$ $h_2^{(0)}$ $g_2^{(0)}$ $h_3^{(0)}$ $g_3^{(0)}$ $h_4^{(0)}$ $g_4^{(0)}$
(a)
$h_1^{(1)}$ $g_1^{(1)}$
Attention
Q
K, V
$h_1^{(0)}$ $g_1^{(0)}$ $h_2^{(0)}$ $g_2^{(0)}$ $h_3^{(0)}$ $g_3^{(0)}$ $h_4^{(0)}$ $g_4^{(0)}$
(b)
x_1 x_2 x_3 x_4
$h_1^{(2)}$ $g_1^{(2)}$ $h_2^{(2)}$ $g_2^{(2)}$ $h_3^{(2)}$ $g_3^{(2)}$ $h_4^{(2)}$ $g_4^{(2)}$
Masked Two-stream Attention
$h_1^{(1)}$ $g_1^{(1)}$ $h_2^{(1)}$ $g_2^{(1)}$ $h_3^{(1)}$ $g_3^{(1)}$ $h_4^{(1)}$ $g_4^{(1)}$
Masked Two-stream Attention
$e(x_1)$ w $e(x_2)$ w $e(x_3)$ w $e(x_4)$ w
(c)
Attention Masks
Content stream:
can see self
Query stream:
cannot see self
Sample a factorization order:
3 → 2 → 4 → 1

(a) content-stream; (b) query-stream; (c) whole model

▸ Source: Yang et al., 2019

XLNET – MODEL INPUT

Generation of samples:

- Randomly sample two sentences and use concatenation* as input

[CLS]	The	fox	is	quick	.	[SEP]	
It	jumps	over	the	lazy	dog	.	[SEP]

*Nevertheless: XLNet does *not* use the NSP objective

Additional encodings:

- *Relative* segment encodings:
  - BERT adds absolute segment embeddings (E_A & E_B)
  - XLNet uses relative encodings ($\vec{s}_+$ & $\vec{s}_-$)
- *Relative* Positional encodings:
  - BERT encodes information about the absolute position ($E_0, E_1, E_2, \ldots$)
  - XLNet uses relative encodings (R_{i-j})

XLNET – SPECIAL REMARKS

- **Partial Prediction:** Only predict the last tokens in a factoriztion order (reduces optimization difficulty)

$$\max_{\theta} \quad \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}_T}\left[\sum_{t=c+1}^{|\mathbf{z}|} \log p_\theta(x_{z_t}|\mathbf{x}_{\mathbf{z}_{<t}})\right], \quad \text{with } c \text{ as cutting point}$$

- **Segment recurrence mechanism:** Allow for learning extended contexts in Transformer-based architectures, see ▸ Dai et al. (2019)
- **No independence assumption:**

$$\mathcal{J}_{\text{BERT}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city}),$$

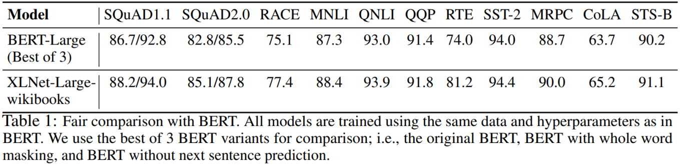
$$\mathcal{J}_{\text{XLNet}} = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{New}, \text{is a city})$$

Prediction of [New, York] given the factorization order [is, a, city, New, York]

▸ Source: Yang et al., 2019

XLNET – SOTA PERFORMANCE

Performance differences to BERT:

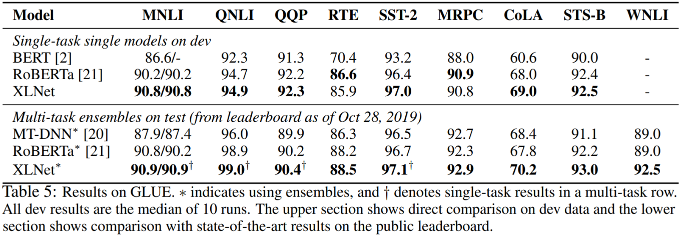


Model	SQuAD1.1	SQuAD2.0	RACE	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B
BERT-Large (Best of 3)	86.7/92.8	82.8/85.5	75.1	87.3	93.0	91.4	74.0	94.0	88.7	63.7	90.2
XLNet-Large-wikibooks	88.2/94.0	85.1/87.8	77.4	88.4	93.9	91.8	81.2	94.4	90.0	65.2	91.1

Table 1: Fair comparison with BERT. All models are trained using the same data and hyperparameters as in BERT. We use the best of 3 BERT variants for comparison; i.e., the original BERT, BERT with whole word masking, and BERT without next sentence prediction.

▸ Source: Yang et al., 2019

SOTA performance:



Model	**MNLI**	**QNLI**	**QQP**	**RTE**	**SST-2**	**MRPC**	**CoLA**	**STS-B**	**WNLI**
Single-task single models on dev									
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
RoBERTa [21]	90.2/90.2	94.7	92.2	**86.6**	96.4	**90.9**	68.0	92.4	-
XLNet	**90.8/90.8**	**94.9**	**92.3**	85.9	**97.0**	90.8	**69.0**	**92.5**	-
Multi-task ensembles on test (from leaderboard as of Oct 28, 2019)									
MT-DNN* [20]	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0
RoBERTa* [21]	90.8/90.2	98.9	90.2	88.2	96.7	92.3	67.8	92.2	89.0
XLNet*	**90.9/90.9**†	**99.0**†	**90.4**†	**88.5**	**97.1**†	**92.9**	**70.2**	**93.0**	**92.5**

Table 5: Results on GLUE. $*$ indicates using ensembles, and † denotes single-task results in a multi-task row. All dev results are the median of 10 runs. The upper section shows direct comparison on dev data and the lower section shows comparison with state-of-the-art results on the public leaderboard.

▸ Source: Yang et al., 2019