Using the Transformer

XLNet

Yang et al., 2019

Learning goals

- Understand the improvements over BERT
- Permutation language modeling

CONCEPTUAL DIFFERENCES

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 $\hat{\mathbf{x}}$:

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

  with m_t as masking indicator
- Flexibility: Also other "*corruptions*" possible

$\rightarrow$ Replacing words with predictions (cf. ELECTRA)

$\rightarrow$ Shuffling tokens or dropping them

CONCEPTUAL DIFFERENCES

Drawbacks / Advantages

- No corruption (needed) of input sequences when using AR approach
- "Causal" structure in AR approach (sometimes needed)
- AE approach induces independence assumption between corrupted tokens

$\rightarrow$ **Why?**

- AR approach only conditions on left side context

$\rightarrow$ No (deep) bidirectionality possible (just e.g. biLSTMs)

ALTERNATIVE OBJECTIVE FUNCTION

Permutation language modeling (PLM)

- Solves the pretrain-finetune discrepancy

$\rightarrow$ No artificial `[MASK]` token is introduced

- Allows for bidirectionality while keeping an AR objective

$\rightarrow$ Best of both worlds

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

ALTERNATIVE OBJECTIVE FUNCTION

Manipulating the factorization order

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

$\rightarrow$ Used to permute the factorization order, *not* the sequence order.

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

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

CONTENT- VS. QUERY-STREAM

Content-stream

- "Normal" Self-Attention (despite with special attention masks)
  $\rightarrow$ Attentions masks depend on the factorization order
- Info about the position in the sequence is lost, see ▸ Example in A.1
- 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

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)

CONTENT- VS. QUERY-STREAM

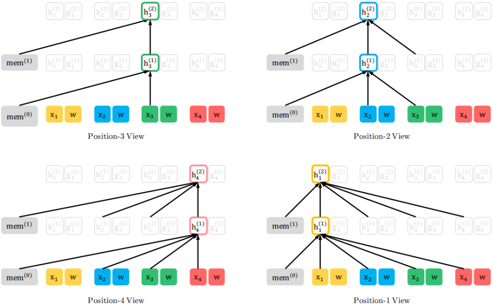
Query-stream

- 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

CONTENT- VS. QUERY-STREAM

Content-stream

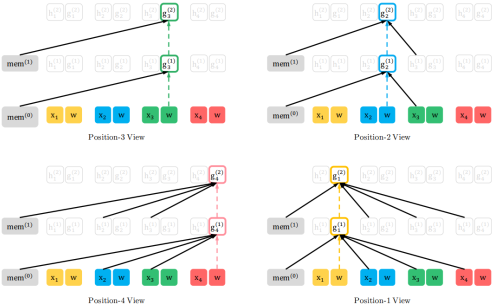


Split View of the Content Stream
(Factorization order: 3 → 2 → 4 → 1)

Source: Yang et al. (2019)

CONTENT- VS. QUERY-STREAM

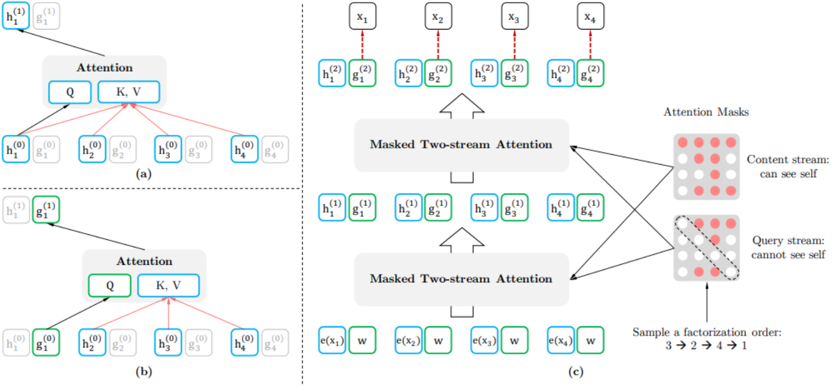
Query-stream



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

Source: Yang et al. (2019)

XLNET – GRAPHICAL REPRESENTATION



(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 $\dagger$ 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)