Chapter 11:
BERT – Improvements & Alternatives

Matthias Aßenmacher

January 27, 2021

What we will cover in this lecture

Disclaimer I:
The overview presented in this chapter is everything but exhaustive!

We will discuss *Shortcomings and limitations* of BERT & Co.

Architectures

- using *different objectives* (XLNet, ELECTRA),
- exhibiting *different architectural design choices* (XLNet, T5) and
- *alleviating some of the shortcomings* (e.g. BigBird, Linformer)

will be presented.

Further the concept of model distillation will be covered.

Disclaimer II:
Before you ask: Yes, GPT-3 will also be covered (but in the next chapter).

Limitations // Shortcomings of BERT

Pretrain-finetune discrepancy

- BERT *artificially* introduces `[MASK]` tokens during pre-training
- `[MASK]`-token does not occur during fine-tuning
  $\rightarrow$ Lacks the ability to model joint probabilities
  $\rightarrow$ Assumes independence of predicted tokens (given the context)

Maximum sequence length

- Based on the encoder part of the Transformer
  $\rightarrow$ Computational complexity of Self-Attention mechanism scales quadratically with the sequence length
- BERT can only consume sequences of length ≤ 512

XLNet

▸ Yang et al., 2019

Autoregressive (AR) language modeling vs. Autoencoding (AE)

- *AR:* Factorizes likelihood to $p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t|\mathbf{x}< t)$
- Only uni-directional (backward factorization instead would also be possible)
- *AE:* Objective to reconstruct masked tokens $\bar{\mathbf{x}}$ from corrupted sequence $\hat{\mathbf{x}}$: $p(\bar{\mathbf{x}}|\hat{\mathbf{x}}) = \prod_{t=1}^{T} m_t \cdot p(x_t|\hat{\mathbf{x}})$, m_t as masking indicator

Drawbacks / Advantages

- No corruption of input sequences when using AR approach
- AE approach induces independence assumption between corrupted tokens
- AR approach only conditions on left side context
  $\Rightarrow$ No bidirectionality

Alternative objective funktion

Permutation language modeling (PLM)

- Solves the pretrain-finetune discrepancy
- Allows for bidirectionality while keeping AR objective
- Consists of two "*streams*" of the Attention mechanism
  - Content-stream attention
  - Query-stream attention

Manipulating the factorization order

- Consider permutations $\mathbf{z}$ of the index sequence $[1, 2, \ldots, T]$
  $\rightarrow$ Used to permute the factorization order, *not* the sequence order.
- Original sequence order is retained by using positional encodings
- PLM objective (with $\mathcal{Z}_T$ as set of all possible permutations):

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

Content- vs. Query-stream

Content-stream

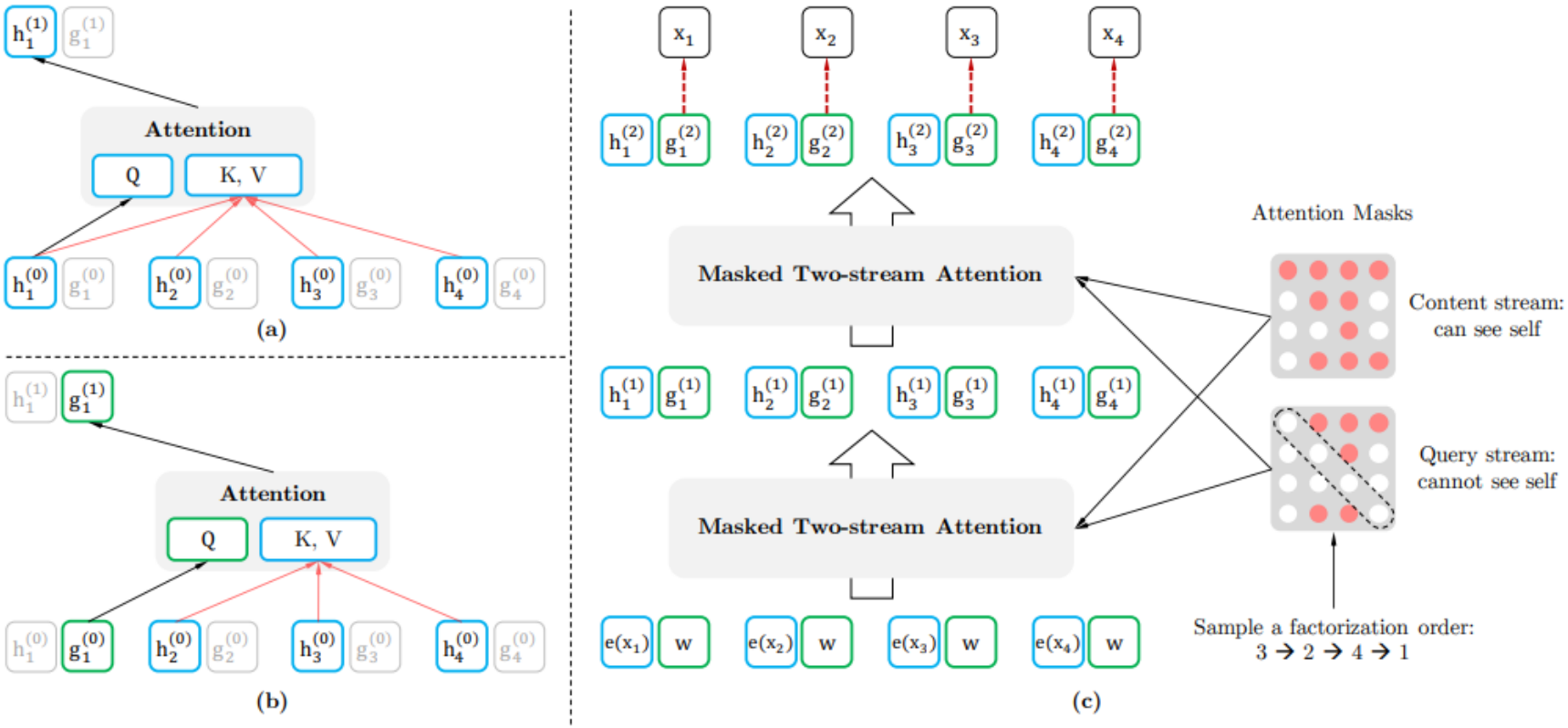
- "Normal" Self-Attention (despite with special attention masks)
  $\rightarrow$ Attentions masks depend on the factorization order
- Info about the position in the sequence is lost, see ▸ Example in A.1
- Sets of queries (Q), keys (K) and values (V) from content stream*
- Yields a *content embedding*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

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

XLNet – Graphical representation



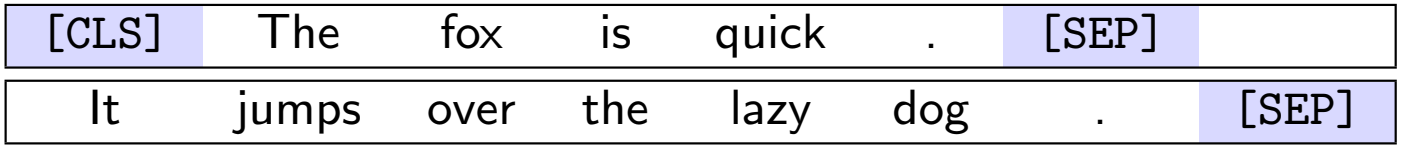
(a) content-stream; (b) query-stream; (c) whole model

Source: Yang et al. (2019)

XLNet – Model Input

Generation of samples:

- Randomly sample two sentences and use concatenation* as input



*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 ($\mathbf{s}_+$ & $\mathbf{s}_-$)
- *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**

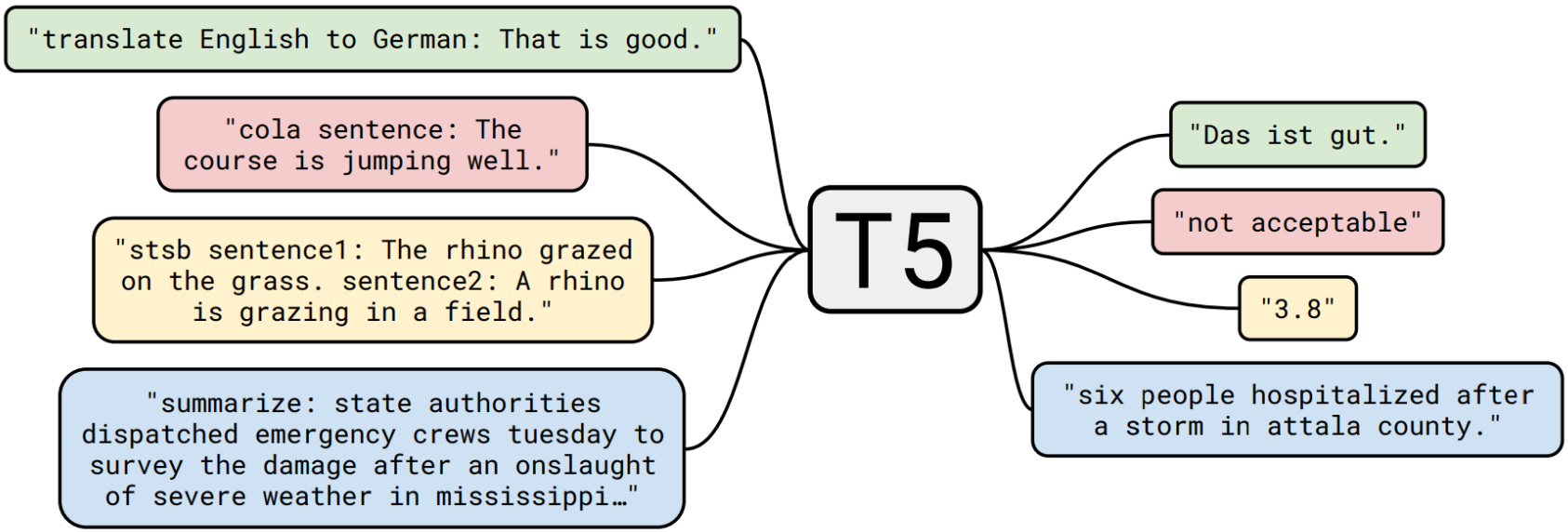
Table 5: Results on GLUE. $*$ indicates using ensembles, and $\dagger$ denotes single-task results in a multi-task row. All dev results are the median of 10 runs. The upper section shows direct comparison on dev data and the lower section shows comparison with state-of-the-art results on the public leaderboard.

Source: Yang et al. (2019)

Google's T5 ▸ Raffel et al. (2019)

T5: Text-to-Text Transfer Transformer:

- A complete encoder-decoder Transformer architecture
- All tasks reformulated as text-to-text tasks
- From BERT-size up to 11 Billion parameters



Source: Raffel et al. (2019)

T5 – The **C**olossal **C**lean **C**rawled **C**orpus (C4)

- Effort to measure the effect of quality, characteristics & size of the pre-training resources
- Common Crawl as basis, careful cleaning and filtering for English language
- Orders of magnitude larger (750GB) compared to commonly used corpora

Experiments (with respect to C4)

Data set	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	**19.24**	80.88	71.36	**26.98**	**39.82**	**27.65**
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21
RealNews-like	35GB	**83.83**	**19.23**	80.39	72.38	**26.75**	**39.90**	**27.48**
WebText-like	17GB	**84.03**	**19.31**	**81.42**	71.40	**26.80**	**39.74**	**27.59**
Wikipedia	16GB	81.85	**19.31**	81.29	68.01	**26.94**	39.69	**27.67**
Wikipedia + TBC	20GB	83.65	**19.28**	**82.08**	**73.24**	**26.77**	39.63	**27.57**

Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Full data set	0	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
2^{29}	64	**82.87**	**19.19**	**80.97**	**72.03**	**26.83**	**39.74**	**27.63**
2^{27}	256	82.62	**19.20**	79.78	69.97	**27.02**	**39.71**	27.33
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56	26.80
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84	25.81

Source: Raffel et al. (2019)

T5 - Exhaustive Experiments

Performed experiments with respect to ..

- .. architecture, size & objective
- .. details of the Denoising objective
- .. fine-tuning methods & multi-taks learning strategies

Conclusions

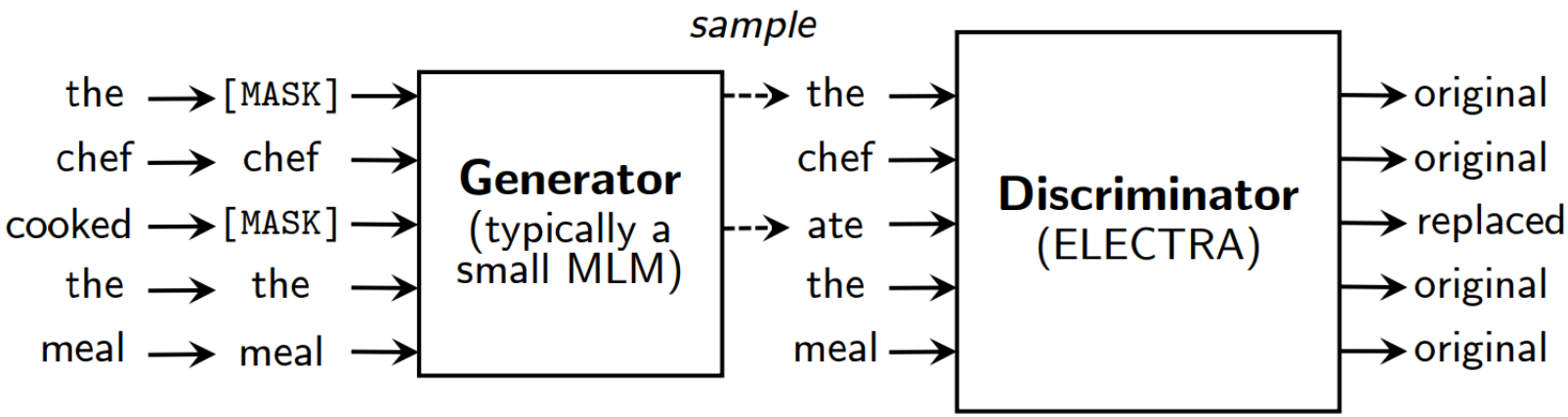
- Encoder-decoder architecture works best in this "text-to-text" setting
- More data, larger models & ensembling all boost the performance
  - Larger models trained for fewer steps better than smaller models on more data
  - Ensembling: Using same base pre-trained models worse than complete separate model ensembles
- Different denoising objectives work similarly well
- Updating *all* model parameters during fine-tuning works best (but expensive)

ELECTRA

▸ Clark et al. (2020)

ELECTRA: a different pre-training objective

- (Small) generator model G + (large) Discriminator model D
- Generator task: Masked language modeling
- Discriminator task: *Replaced token detection*
- ELECTRA learns from *all* of the tokens (not just from a small portion, like e.g. BERT)



Source: Clark et al. (2020)

ELECTRA – Training details

Joint pre-training (but not in a GAN-fashion):

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

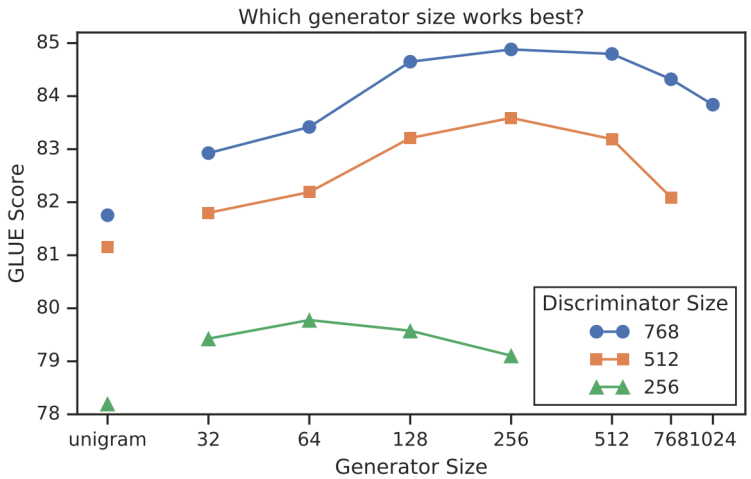
$$\min_{\theta_G, \theta_D} \sum_{\boldsymbol{x} \in \mathcal{X}} \mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) + \lambda \mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D)$$

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 as much compute per training step + too challenging for D
- Smaller Generators are preferable ($1/4$ – $1/2$ the size of D)

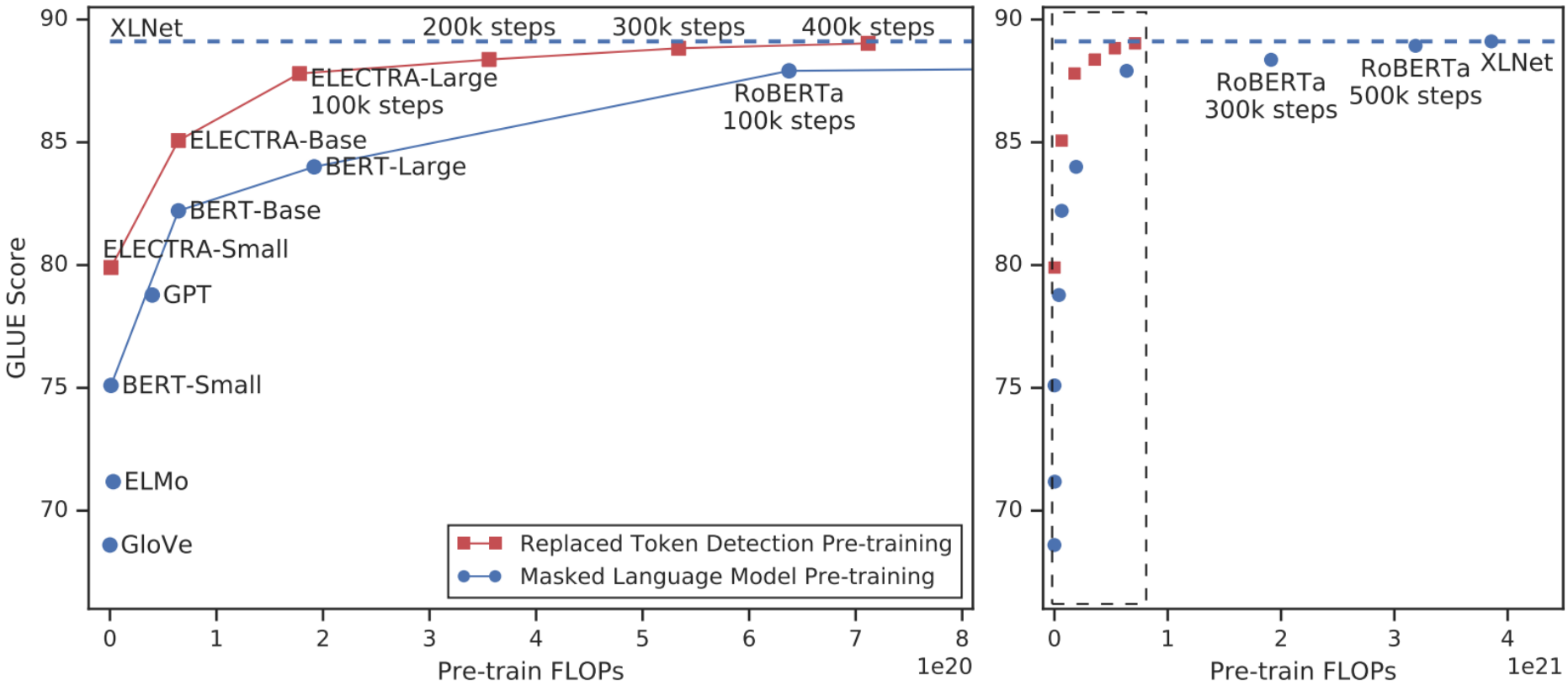


Source: Clark et al. (2020)

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)

Model distillation

Hinton et al. (2015)

Model compression scheme:

- Motivation comes from having computationally expensive, cumbersome ensemble models. Bucila et al. (2006)
- Compressing the knowlegde of the ensemble into a single model has the benefit of easier deployment and better generalization
- Reasoning:
  - Cumbersome model generalizes well, because it is the average of an ensemble.
  - Small model trained to generalize in the same way typically better than small model trained "the normal way".

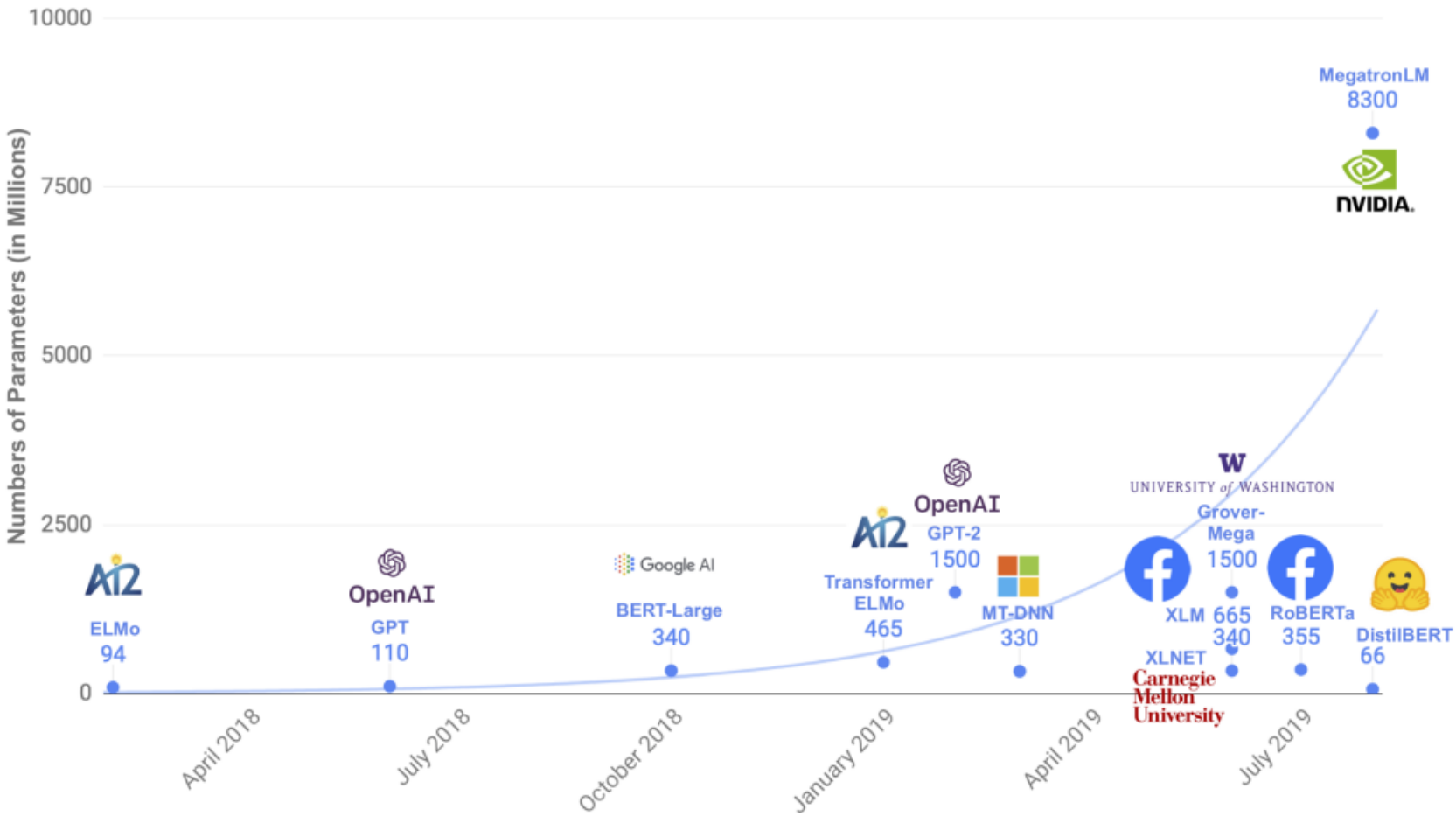
Distillation:

- Temperature T in the softmax: $q_i = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$
- Knowledge transfer via soft targets with high T from original model.
- When true labels are known: Weighted average of two different objective functions

DistilBERT

▸ Sanh et al. (2019)

Motivation:



Source: Sanh et al. (2019)

DistilBERT

▸ Sanh et al. (2019)

Student architecture (DistilBERT):

- Half the number of layers compared to BERT*
- Half of the size of BERT, but retains 95% of the performance
- Initialize from BERT (taking one out of two hidden layers)
- Same pre-training data as BERT (Wiki + BooksCorpus)

Training and performance

- Distillation loss $L_{ce} = \sum_i t_i \cdot \log(s_i)$ + MLM-Loss L_{mlm} + Cosine-Embedding-Loss L_{cos}
- Drops NSP, use dynamic masking, train with large batches

*Rationale for "only" reducing the number of layers:

Larger influence on the computation efficiency compared to e.g. hidden size dimension

DistilBERT

▸ Sanh et al. (2019)

Performance differences to BERT:

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Source: Sanh et al. (2019)

Ablation study regarding the loss:

Table 4: **Ablation study.** Variations are relative to the model trained with triple loss and teacher weights initialization.

Ablation	**Variation on GLUE macro-score**
$\emptyset$ - L_{cos} - L_{mlm}	-2.96
L_{ce} - $\emptyset$ - L_{mlm}	-1.46
L_{ce} - L_{cos} - $\emptyset$	-0.31
Triple loss + random weights initialization	-3.69

Source: Sanh et al. (2019)

The $\mathcal{O}(n^2)$ problem

Quadratic time & memory complexity of Self-Attention

- *Inductive bias of Transformer models:*
  Connect all tokens in a sequence to each other
- **Pro:** Can (theoretically) learn contexts of arbitrary length
- **Con:** Bad scalability limiting (feasible) context size

Resulting Problems:

- Several tasks require models to consume longer sequences
- *Efficiency:* Are there more efficient modifications which achieve similar or even better performance?

Efficient Transformers ▸ Tay et al. (2020)

Broad overview on so-called "X-formers":

- Efficient & fast Transformer-based models
  $\rightarrow$ Reduce complexity from $\mathcal{O}(n^2)$ to (up to) $\mathcal{O}(n)$
- Claim on-par (or even) superior performance
- Different techniques used:
  - Fixed/Factorized/Random Patterns
  - Learnable Patterns (extension of the above)
  - Low-Rank approximations or Kernels
  - Recurrence (see e.g. ▸ Transformer-XL (Dai et al., 2019))
  - Memory modules

Side note:

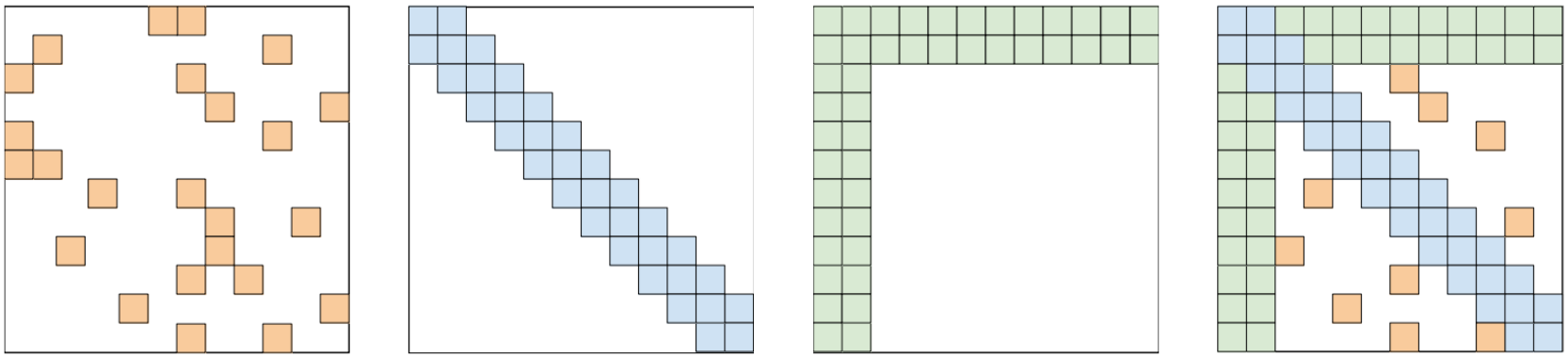
- Most Benchmark data sets not explicitly designed for evaluating long-range abilities of the models.
- Recently proposed: ▸ *Longe Range Arena: A benchmark for efficient transformers* (Tay et al., 2020)

Introducing Patterns

Reasoning:

- Making every token attend to every other token might be unnecessary
- Introduce sparsity in the commonly dense attention matrix

Example:



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

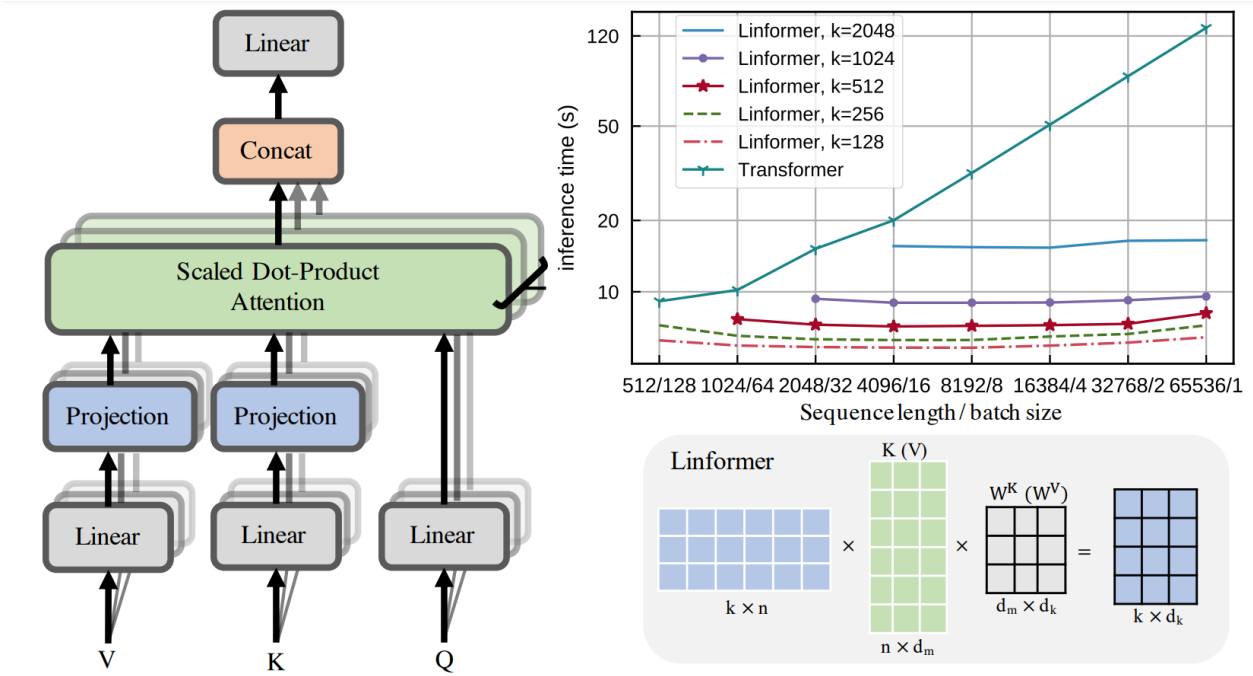
Source: Zaheer et al. (2020)

Linear Self-Attention

▸ Wang et al. (2020)

Reasoning:

- Most information in the Self-Attention mechanism can be recovered from the first few, largest singular values
- Introduce additional k-dimensional projection before self-attention



Source: Wang et al. (2020)

DeBERTa

▸ He et al. (2020)

Disentangled Attention:

- Each token represented by two vectors for content ($\mathbf{H}_i$) and relative position ($\mathbf{P}_{i|j}$)
- Calculation of the Attention Score:

$$\begin{aligned} A_{i,j} &= \{\boldsymbol{H_i}, \boldsymbol{P_{i|j}}\} \times \{\boldsymbol{H_j}, \boldsymbol{P_{j|i}}\}^\intercal \\ &= \boldsymbol{H_i H_j^\intercal} + \boldsymbol{H_i P_{j|i}^\intercal} + \boldsymbol{P_{i|j} H_j^\intercal} + \boldsymbol{P_{i|j} P_{j|i}^\intercal} \end{aligned}$$

- with content-to-content, content-to-position, position-to-content and position-to-position attention

Disentangled Attention

Standard (Single-head) Self-Attention:

$$\boldsymbol{Q} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{q}}, \boldsymbol{K} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{k}}, \boldsymbol{V} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{v}}, \boldsymbol{A} = \frac{\boldsymbol{Q}\boldsymbol{K}^{\intercal}}{\sqrt{d}}$$

$$\boldsymbol{H}_{\boldsymbol{o}} = \mathrm{softmax}(\boldsymbol{A})\boldsymbol{V}$$

Disentangled Attention*:

$$\boldsymbol{Q}_{\boldsymbol{c}} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{q},\boldsymbol{c}}, \boldsymbol{K}_{\boldsymbol{c}} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{k},\boldsymbol{c}}, \boldsymbol{V}_{\boldsymbol{c}} = \boldsymbol{H}\boldsymbol{W}_{\boldsymbol{v},\boldsymbol{c}}, \boldsymbol{Q}_{\boldsymbol{r}} = \boldsymbol{P}\boldsymbol{W}_{\boldsymbol{q},\boldsymbol{r}}, \boldsymbol{K}_{\boldsymbol{r}} = \boldsymbol{P}\boldsymbol{W}_{\boldsymbol{k},\boldsymbol{r}}$$

$$\tilde{A}_{i,j} = \underbrace{\boldsymbol{Q}_{\boldsymbol{i}}^{\boldsymbol{c}}\boldsymbol{K}_{\boldsymbol{j}}^{\boldsymbol{c}\intercal}}_{\text{(a) content-to-content}} + \underbrace{\boldsymbol{Q}_{\boldsymbol{i}}^{\boldsymbol{c}}\boldsymbol{K}_{\boldsymbol{\delta}(\boldsymbol{i},\boldsymbol{j})}^{\boldsymbol{r}\intercal}}_{\text{(b) content-to-position}} + \underbrace{\boldsymbol{K}_{\boldsymbol{j}}^{\boldsymbol{c}}\boldsymbol{Q}_{\boldsymbol{\delta}(\boldsymbol{j},\boldsymbol{i})}^{\boldsymbol{r}\intercal}}_{\text{(c) position-to-content}}$$

$$\boldsymbol{H}_{\boldsymbol{o}} = \mathrm{softmax}(\frac{\tilde{\boldsymbol{A}}}{\sqrt{3d}})\boldsymbol{V}_{\boldsymbol{c}}$$

*Position-to-position part is removed since it, according to the authors, does not provide much additional information as *relative* position emebeddings are used.

Further reading

- BART ▸ Lewis et al. (2019)
- BORT ▸ de Wynter & Perry (2020)
- WT5?! ▸ Narang et al. (2020)