Transformer tricks & PEFT

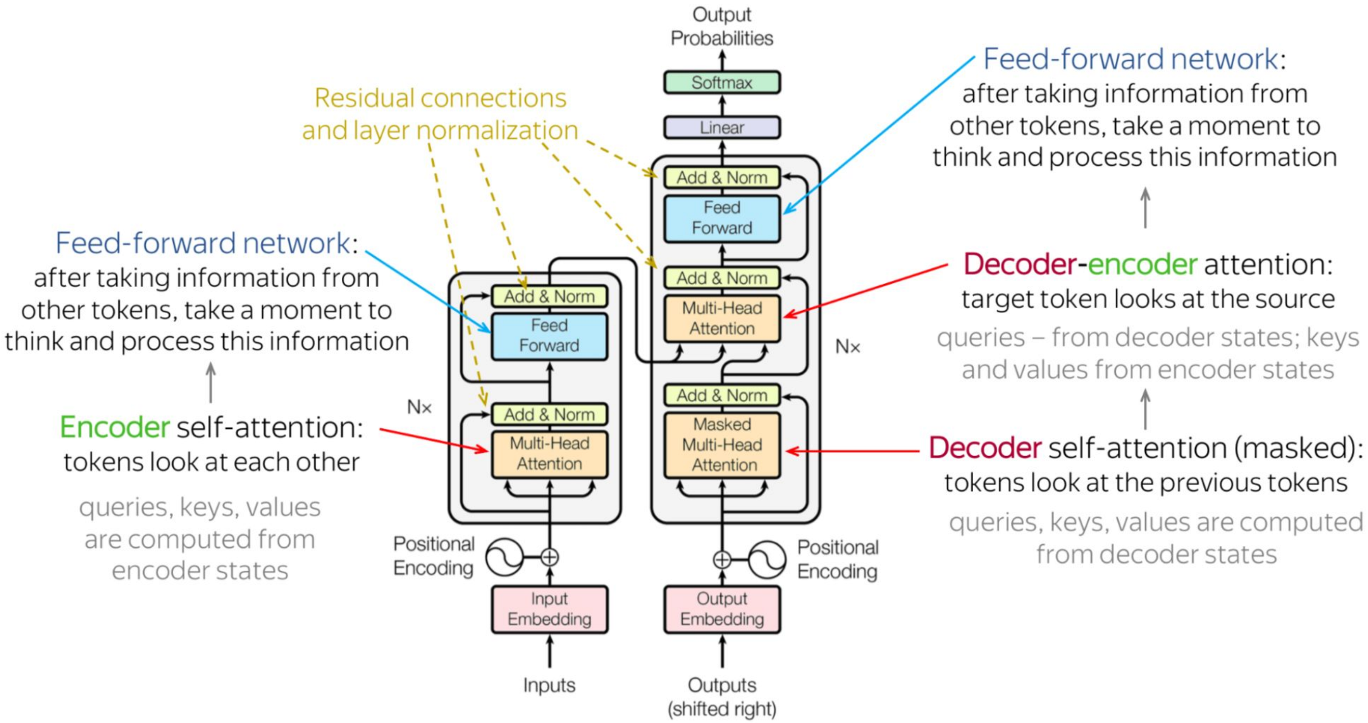
NLP

lecturer: Mollaev D. E.
Sber AI Lab

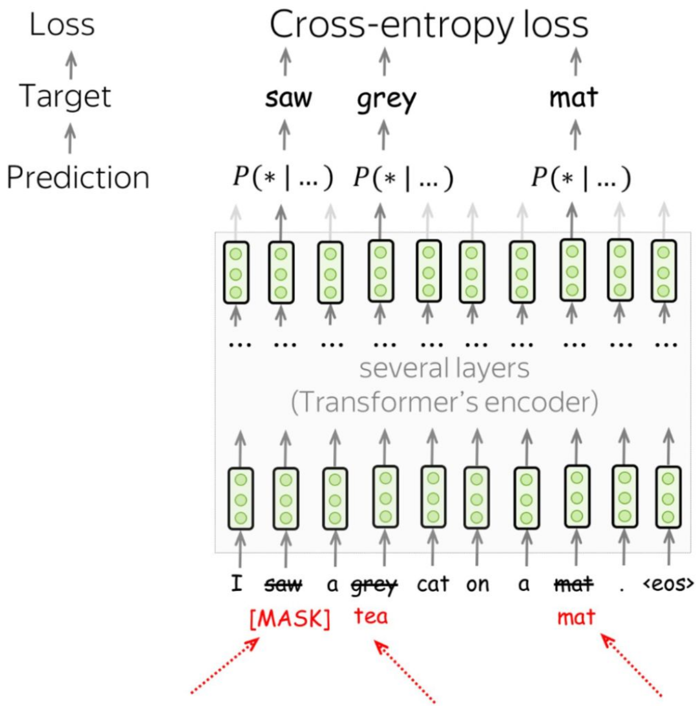
Lecture plan

- Transformer tricks
  - Position encodings
  - Activations
- Friends of BERT
- PEFT

Transformers



BERT



- [MASK], with p = 80%
- Random token, with p = 10%
- Original token, with p = 10%

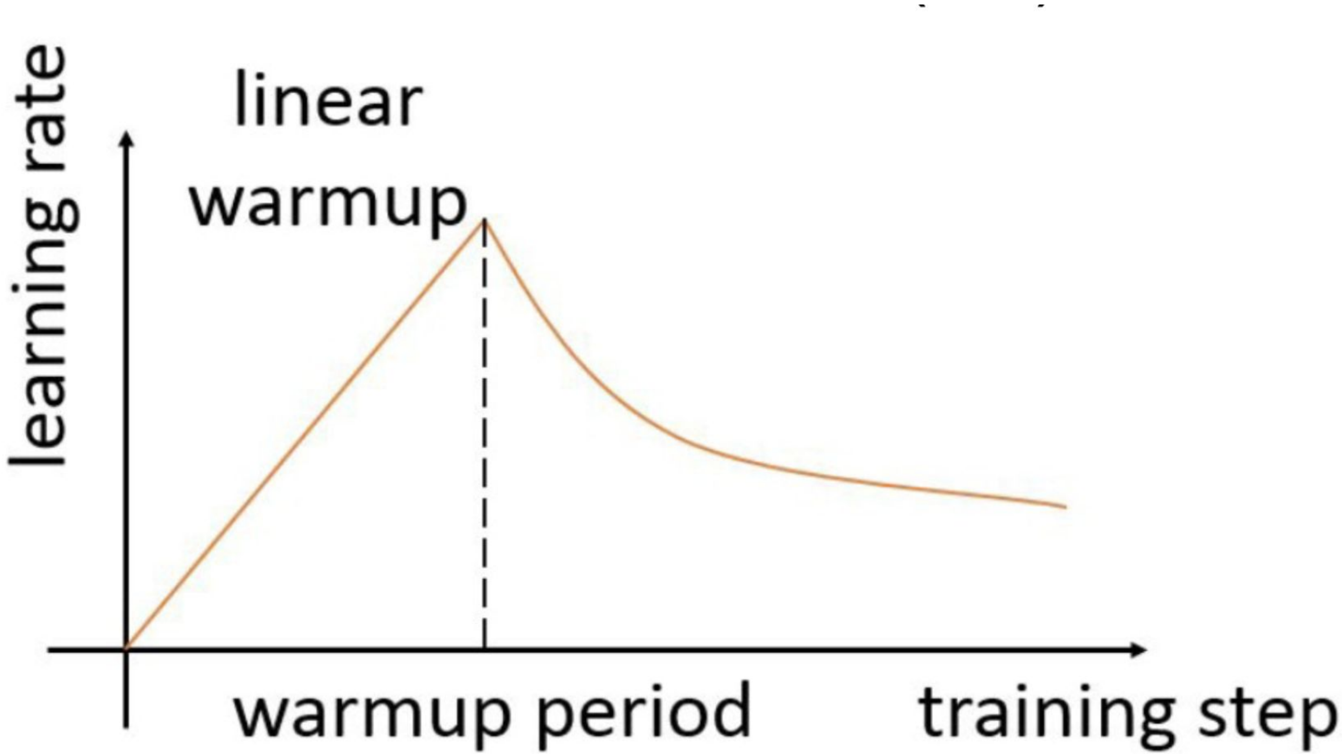
At each training step:

- pick randomly 15% of tokens
- replace each of the chosen tokens with something
- predict original chosen tokens

Lecture plan

- Transformer tricks
  - Position encodings
  - Activations
- Friends of BERT
- PEFT

Transformer tricks: **warm up**



Transformer tricks: **warm up**

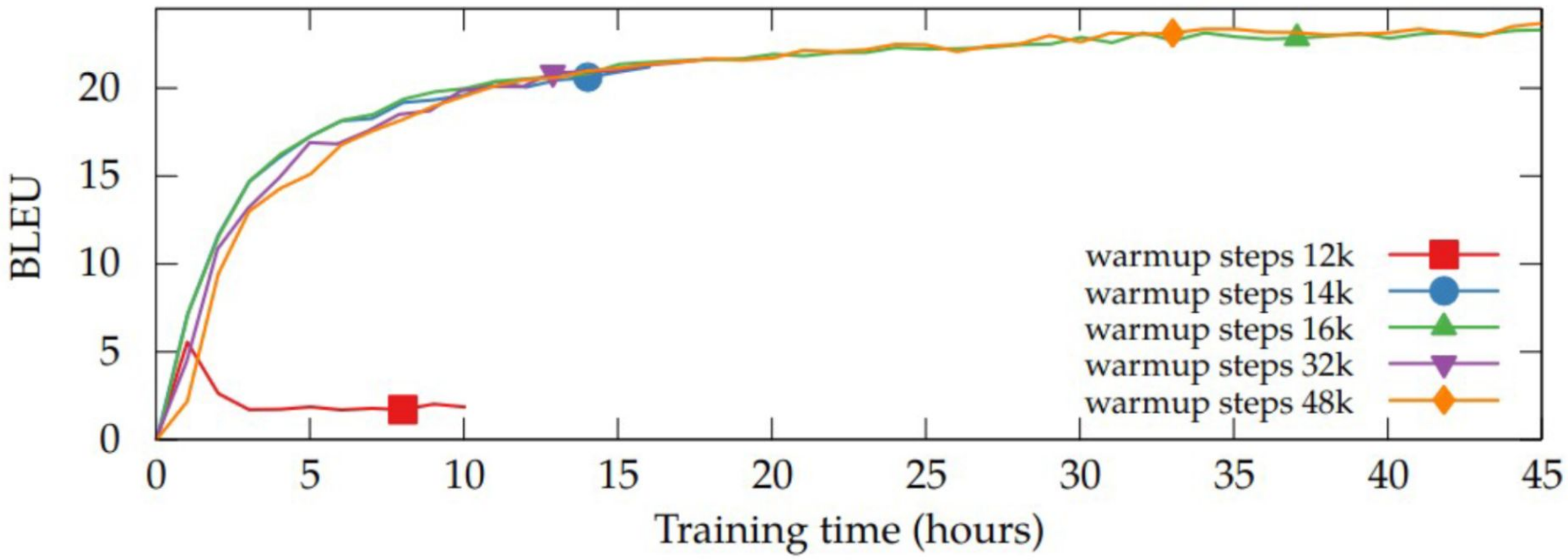


Figure 8: Effect of the warmup steps on a single GPU. All trained on CzEng 1.0 with the default batch size (1500) and learning rate (0.20).

Transformer tricks: **large batch size**

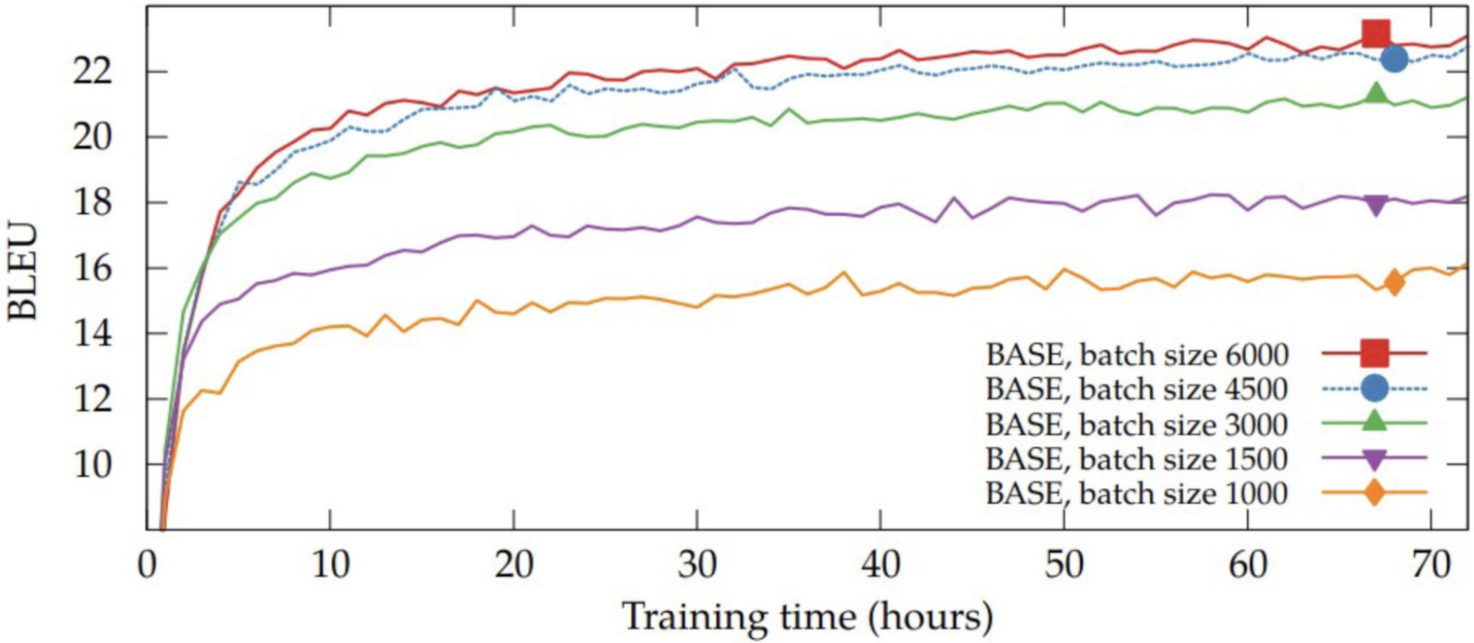


Figure 5: Effect of the batch size with the BASE model. All trained on a single GPU.

Transformer tricks: other

- Adam optimizer
- Learning rate warmup (inverse sqrt decay)
- Large batch sizes - accumulate gradient
- Gather sentence with similar length

Lecture plan

- Transformer tricks
  - Position encodings
  - Activations
- Friends of BERT
- PEFT

Position encodings

- Add sin(t) and cos(t) to inputs
- Use training embeddings for fixed length(512, 2048) sentence

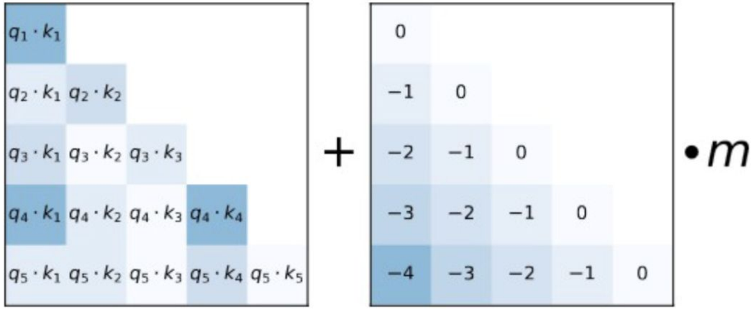
Position encodings: **Relative position encoding**

$$\text{RelativeAttention} = \text{Softmax}\left(\frac{QK^\top + S_{rel}}{\sqrt{D_h}}\right)V$$

Here, S_{rel} is a learned function of [query position – key position]

Position encodings: **ALiBi embeddings**

Add a Linear Bias to each attention head's pre-softmax logits



Here, m is a constant (non-trained) vector with one value per head:

*m[i] = 2 ^ (- i * scale)*

e.g. if scale = 0.5, *m =* $\frac{1}{2^{0.5}}, \frac{1}{2^{1}}, \frac{1}{2^{1.5}}, \dots, \frac{1}{2^{8}}$

Position encodings: **Rotary embeddings**

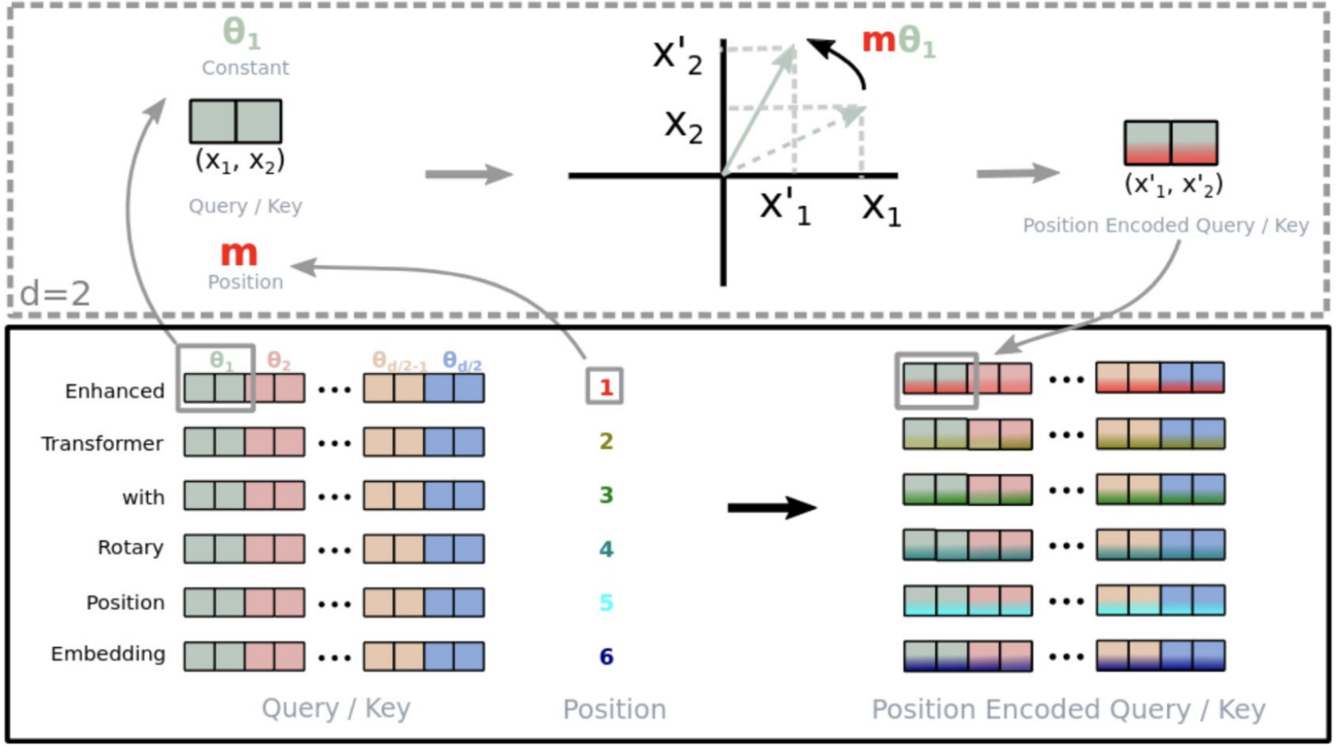
Multiply Q and K vectors by rotation matrix:

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

… using an array of fixed (non-trainable) angles

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, ..., d/2]\}.$$

Position encodings: **Rotary embeddings**



Lecture plan

- Transformer tricks
  - Position encodings
  - Activations
- Friends of BERT
- PEFT

Activations

Original FFN

$$\text{FFN}(x, W_1, W_2, b_1, b_2) = \max(0, xW_1 + b_1)W_2 + b_2$$

Gated FFN variants

$$\text{FFN}_{\text{GLU}}(x, W, V, W_2) = (\sigma(xW) \otimes xV)W_2$$
$$\text{FFN}_{\text{Bilinear}}(x, W, V, W_2) = (xW \otimes xV)W_2$$
$$\text{FFN}_{\text{ReGLU}}(x, W, V, W_2) = (\max(0, xW) \otimes xV)W_2$$
$$\text{FFN}_{\text{GEGLU}}(x, W, V, W_2) = (\text{GELU}(xW) \otimes xV)W_2$$
$$\text{FFN}_{\text{SwiGLU}}(x, W, V, W_2) = (\text{Swish}_1(xW) \otimes xV)W_2$$

Activations

Table 2: GLUE Language-Understanding Benchmark [Wang et al., 2018] (dev).

	Score Average	CoLA MCC	SST-2 Acc	MRPC F1	MRPC Acc	STSB PCC	STSB SCC	QQP F1	QQP Acc	MNLIm Acc	MNLImm Acc	QNLI Acc	RTE Acc
FFN_{ReLU}	83.80	51.32	94.04	**93.08**	**90.20**	89.64	89.42	89.01	91.75	85.83	86.42	92.81	80.14
FFN_{GELU}	83.86	53.48	94.04	92.81	**90.20**	89.69	89.49	88.63	91.62	85.89	86.13	92.39	80.51
$\text{FFN}_{\text{Swish}}$	83.60	49.79	93.69	92.31	89.46	89.20	88.98	88.84	91.67	85.22	85.02	92.33	81.23
FFN_{GLU}	84.20	49.16	94.27	92.39	89.46	89.46	89.35	88.79	91.62	86.36	86.18	92.92	**84.12**
$\text{FFN}_{\text{GEGLU}}$	84.12	53.65	93.92	92.68	89.71	90.26	90.13	89.11	91.85	86.15	86.17	92.81	79.42
$\text{FFN}_{\text{Bilinear}}$	83.79	51.02	**94.38**	92.28	89.46	90.06	89.84	88.95	91.69	**86.90**	**87.08**	92.92	81.95
$\text{FFN}_{\text{SwiGLU}}$	84.36	51.59	93.92	92.23	88.97	**90.32**	**90.13**	**89.14**	**91.87**	86.45	86.47	**92.93**	83.39
$\text{FFN}_{\text{ReGLU}}$	**84.67**	**56.16**	**94.38**	92.06	89.22	89.97	89.85	88.86	91.72	86.20	86.40	92.68	81.59
[Raffel et al., 2019]	83.28	53.84	92.68	92.07	88.92	88.02	87.94	88.67	91.56	84.24	84.57	90.48	76.28
ibid. stddev.	0.235	1.111	0.569	0.729	1.019	0.374	0.418	0.108	0.070	0.291	0.231	0.361	1.393

Lecture plan

- Transformer tricks
  - Position encodings
  - Activations
- $\boxed{\text{Friends of BERT}}$
- PEFT

Friends of Bert: **Roberta**

- Dynamic masking: new random mask each epoch (see table)
- ???
- ???
- ???

Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Friends of Bert: **Roberta**

- Dynamic masking: new random mask each epoch (see table)
- Tune parameters ;) *the original BERT didn't train to convergence!*
- ???
- ???

the effect of pretraining batch size & lr

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

Friends of Bert: **Roberta**

- Dynamic masking: new random mask each epoch (see table)
- Tune parameters ;) *the original BERT didn't train to convergence!*
- Play with inputs and losses: NSP is not necessary!
- ???

Model	**SQuAD 1.1/2.0**	**MNLI-m**	**SST-2**	**RACE**
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3

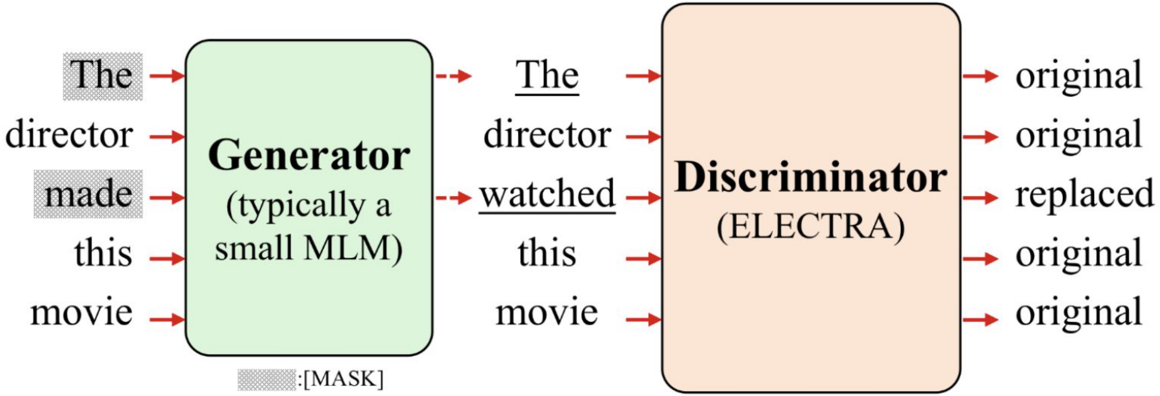
Friends of Bert: **Roberta**

- Dynamic masking: new random mask each epoch (see table)
- Tune parameters ;) *the original BERT didn't train to convergence!*
- Play with inputs and losses: NSP is not necessary!
- Feed it with more data!

Model	**data**	**bsz**	**steps**	**SQuAD (v1.1/2.0)**	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

Friends of Bert: **ELECTRA**

Two models **generator** and **discriminator** (see figure below)



Friends of Bert: **ELECTRA**

Two models **generator** and **discriminator** (see figure below)

Note: the generator is just BERT, not adversarial to discriminator!

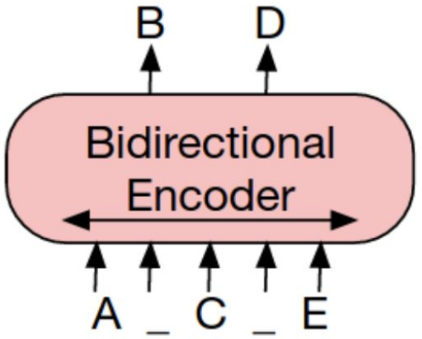
Results: faster / cheaper training, final model ≈ RoBERTa

Model	**Train / Infer FLOPs**	**Speedup**	**Params**	**Train Time + Hardware**	**GLUE**
ELMo	3.3e18 / 2.6e10	19x / 1.2x	96M	14d on 3 GTX 1080 GPUs	71.2
GPT	4.0e19 / 3.0e10	1.6x / 0.97x	117M	25d on 8 P6000 GPUs	78.8
BERT-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100 GPU	75.1
BERT-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	82.2
ELECTRA-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100 GPU	79.9
50% trained	7.1e17 / 3.7e9	90x / 8x	14M	2d on 1 V100 GPU	79.0
25% trained	3.6e17 / 3.7e9	181x / 8x	14M	1d on 1 V100 GPU	77.7
12.5% trained	1.8e17 / 3.7e9	361x / 8x	14M	12h on 1 V100 GPU	76.0
6.25% trained	8.9e16 / 3.7e9	722x / 8x	14M	6h on 1 V100 GPU	74.1
ELECTRA-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	85.1

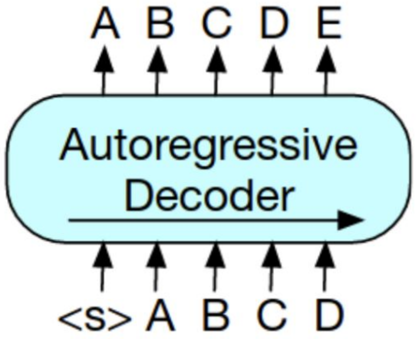
Friends of Bert: **BART**

BERT: full attention, but outputs are predicted independently
GPT: joint prediction, but past tokens cannot look on future tokens

BERT



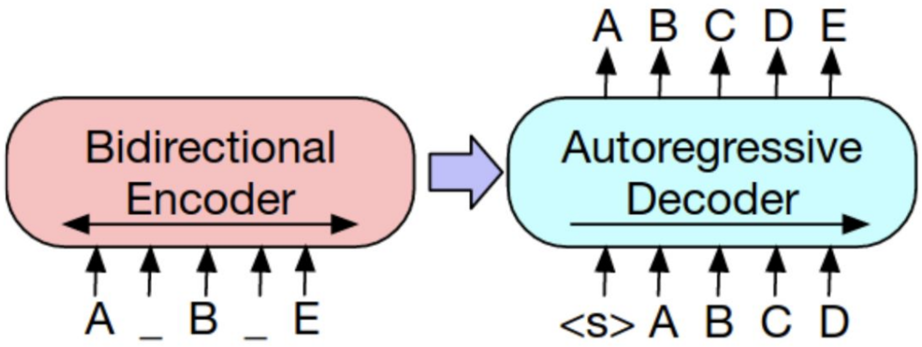
GPT



Friends of Bert: **BART**

BERT: full attention, but outputs are predicted independently
GPT: structured prediction, but past tokens cannot into future

BART: full attention (encoder) and structured prediction (decoder)



Friends of Bert: **BART**

Model	**SQuAD 1.1** F1	**MNLI** Acc	**ELI5** PPL	**XSum** PPL	**ConvAI2** PPL	**CNN/DM** PPL
BERT Base (Devlin et al., 2019)	88.5	**84.3**	-	-	-	-
Masked Language Model	90.0	83.5	24.77	7.87	12.59	7.06
Masked Seq2seq	87.0	82.1	23.40	6.80	11.43	6.19
Language Model	76.7	80.1	**21.40**	7.00	11.51	6.56
Permuted Language Model	89.1	83.7	24.03	7.69	12.23	6.96
Multitask Masked Language Model	89.2	82.4	23.73	7.50	12.39	6.74
BART Base						
w/ Token Masking	90.4	84.1	25.05	7.08	11.73	6.10
w/ Token Deletion	90.4	84.1	24.61	6.90	11.46	5.87
w/ Text Infilling	**90.8**	84.0	24.26	**6.61**	**11.05**	5.83
w/ Document Rotation	77.2	75.3	53.69	17.14	19.87	10.59
w/ Sentence Shuffling	85.4	81.5	41.87	10.93	16.67	7.89
w/ Text Infilling + Sentence Shuffling	**90.8**	83.8	24.17	6.62	11.12	**5.41**

T5 – combine best practices

Paper: https://arxiv.org/abs/1910.10683

- *Encoder-model (like BART)*
- *Model & training hacks (relative pos.emb, modified objective)*
- *Large model, huge data*

DeBERTa v3 – combine best pracices

Paper: https://arxiv.org/abs/2111.09543

- *Generator + discriminator (like ELECTRA)*
- *Model & training hacks (relative pos.emb, sharing hacks)*
- *All kinds of model sizes, huge data*

Lecture plan

- Transformer tricks
  - Position encodings
  - Activations
- Friends of BERT
- $\boxed{\text{PEFT}}$

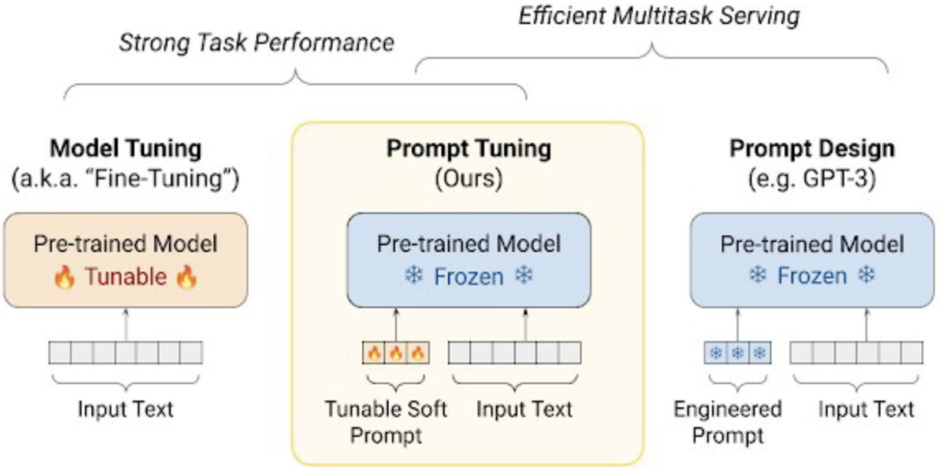
Prompting

- Zero-shot
- Few-shot
- Chain-of-thought

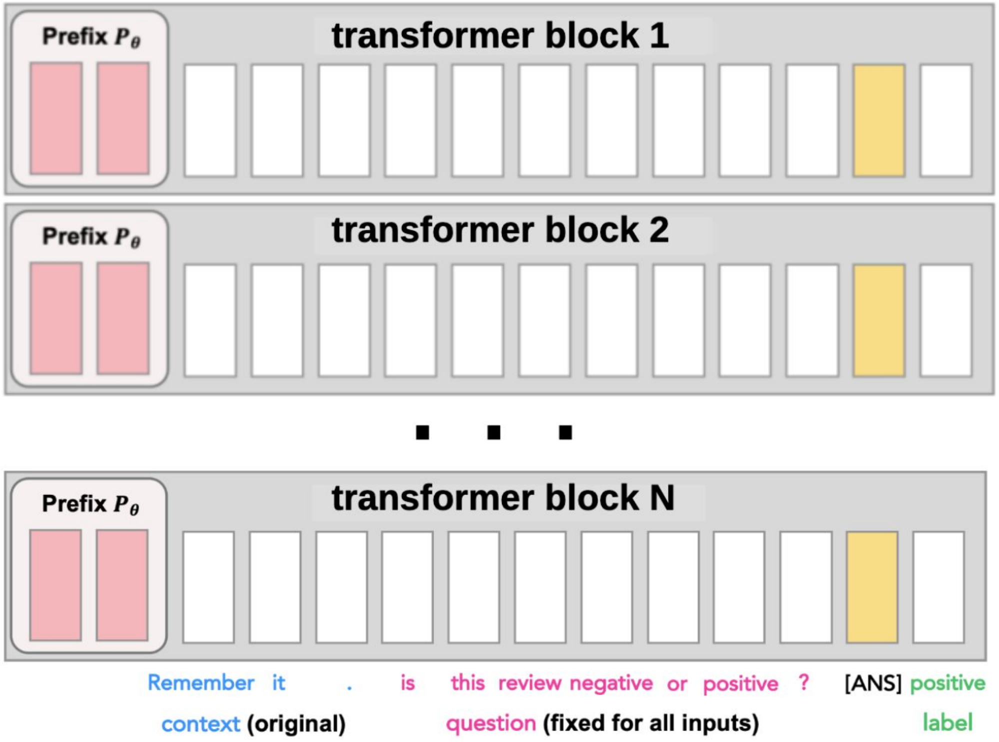
PEFT: **Parameter-Efficient Fine-Tuning**

Prompt Tuning

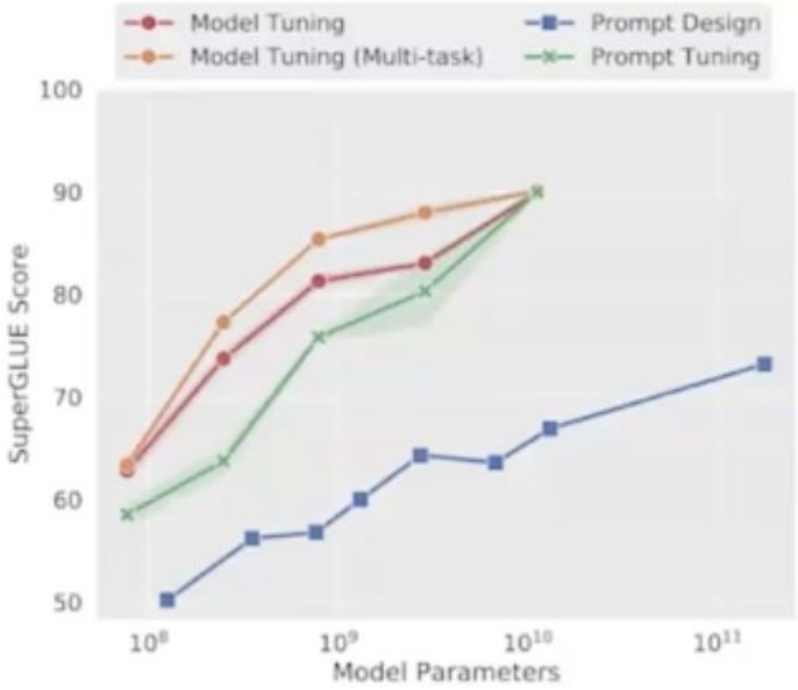
https://aclanthology.org/2021.emnlp-main.243.pdf



PEFT: **Prefix-tuning**



Prompt Tuning gets more competitive with scale!

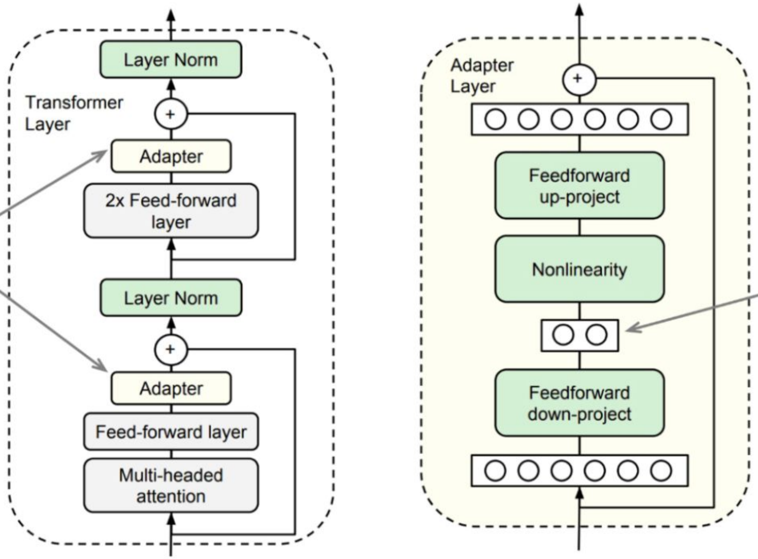


Adapters

https://arxiv.org/abs/1902.00751

Core idea: train small sub-networks

Only these are trained, everything else is fixed and is the same for all tasks

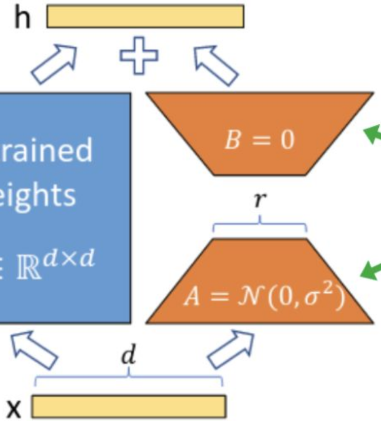


Small hidden size, i.e. an adaptor has only a few parameters (which is good!)

LoRA

https://arxiv.org/abs/2106.09685

Add adapters in parallel with linear layers



train only A & B

PEFT: Summary

- 10-100 samples - prompt engineering
- 100-1000 samples - prompt tuning
- >1000 - LoRA adapters