Parameter-Efficient Fine-tuning

Parameter-Efficient Fine-tuning

- A method of pre-training models by changing a small number of parameters
- Used for training LLMs that are too costly to fine-tune
- When training on several different tasks, you can store only the set of changed parameters for each task

Few-shot and Zero-shot for GPT

- GPT-3 has been trained on 45 TB of data. This gives it useful properties.

- The model can be applied to a new problem by showing it examples of solutions in a prompt.

```
Translate from Russian to English:
стол => table
сыр => cheese
дом =>
```

It's called **Few-shot** learning.

Few-shot and Zero-shot for GPT

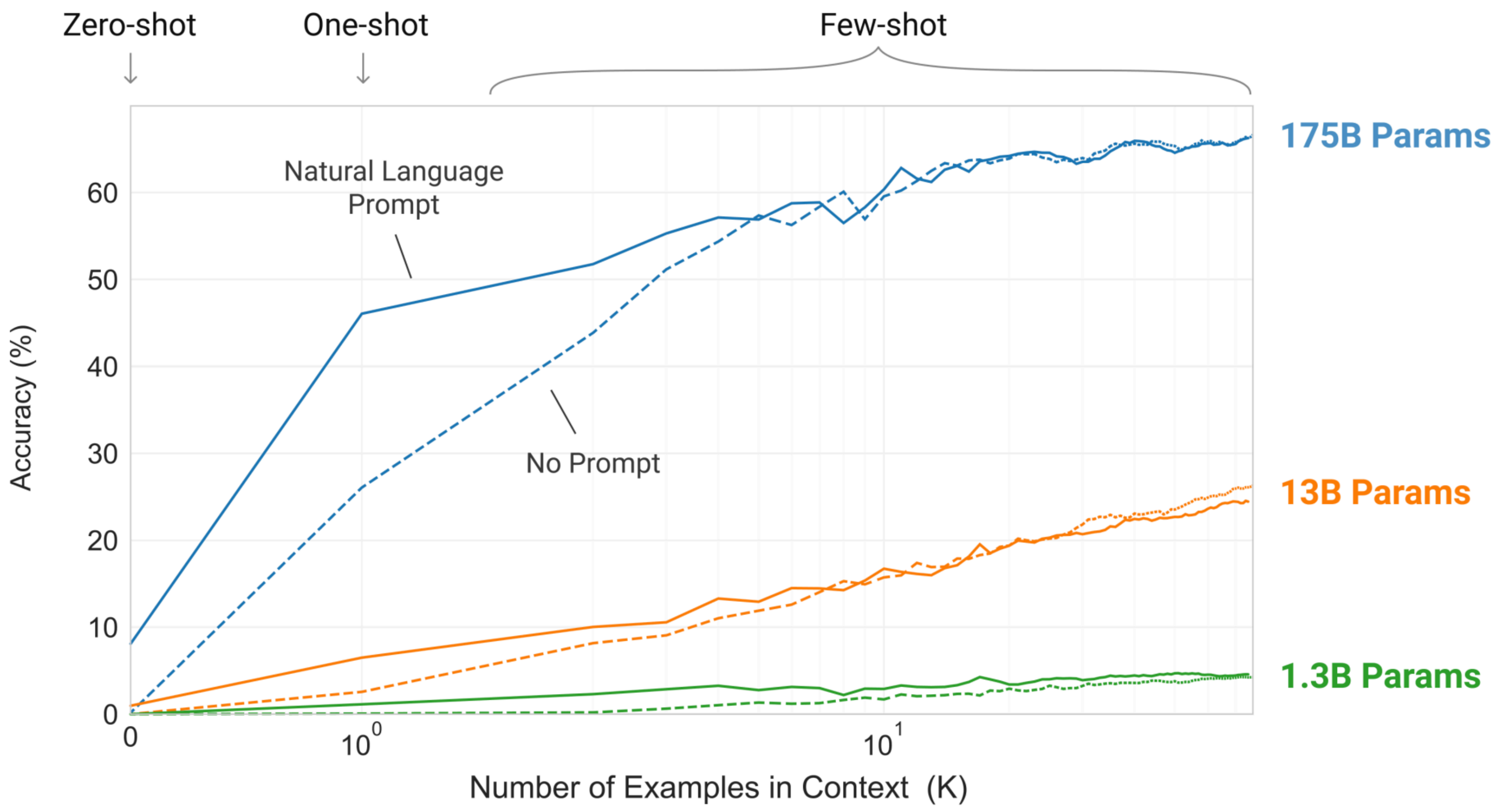
It is possible not to show the correct solutions at all

```
Translate from Russian to English:
дом =>
```

It's called **Zero-shot** learning.

Few-shot and Zero-shot for GPT

The larger the model, the better it performs.



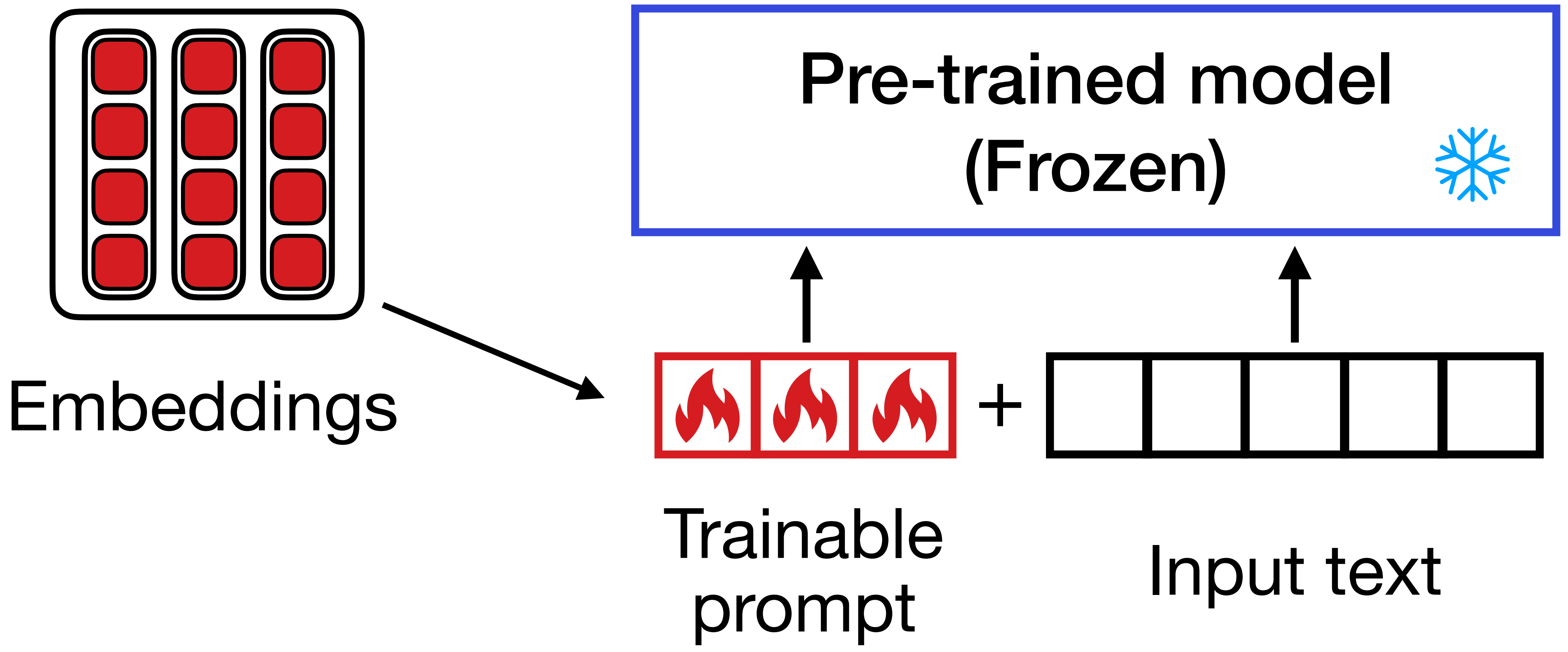
However, the result may be poor because of the fact that

- The model has not learnt how to solve this problem
- Prompt is not good enough

Prompt Tuning

Idea: Let's try to automatically select the most appropriate prompt

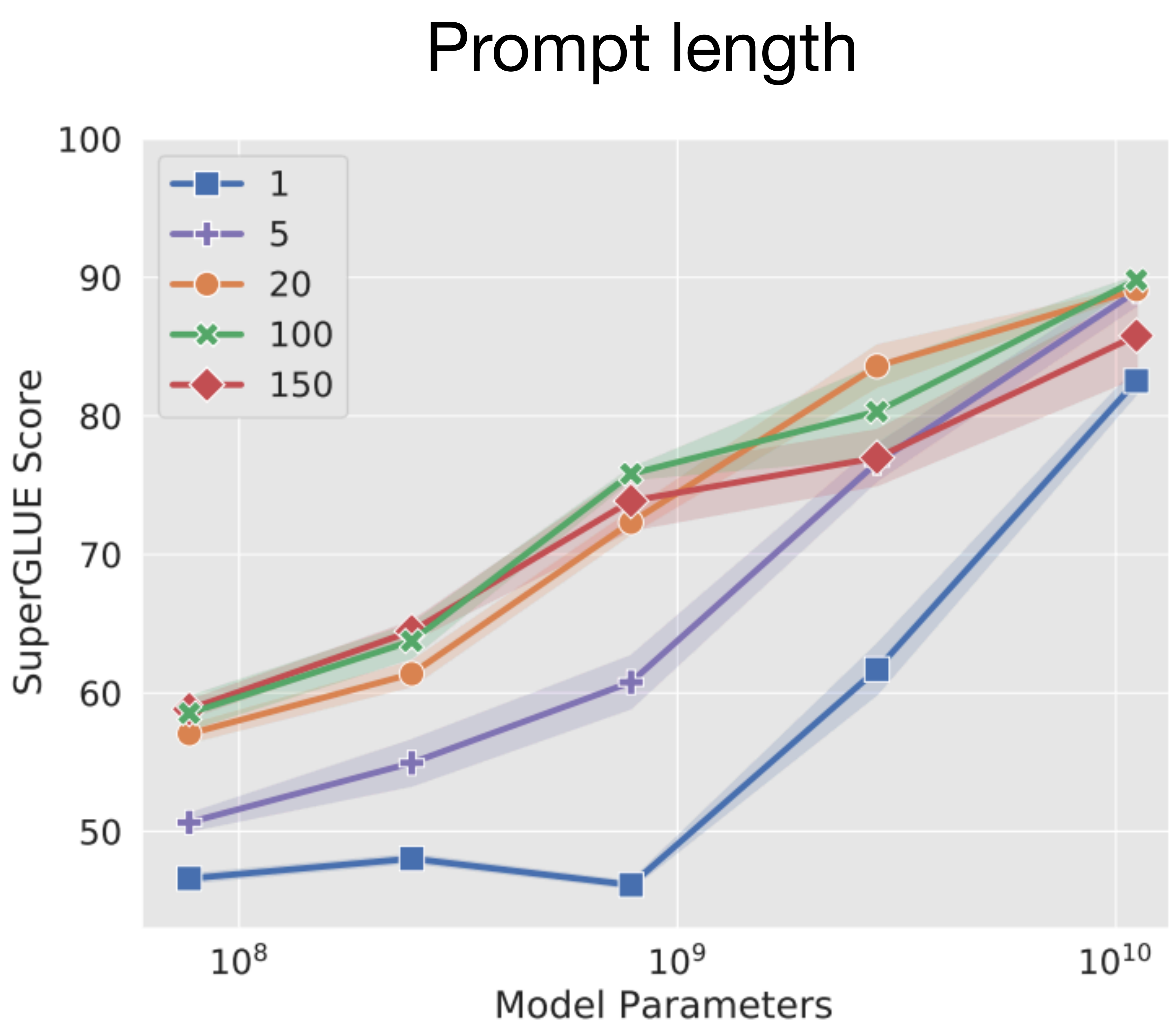
- Initialise the prompt embeddings randomly, specifying only the number of them
- We can initialize the embeddings with embeddings of some prompt.
- For the classification task we need to train the head additionally

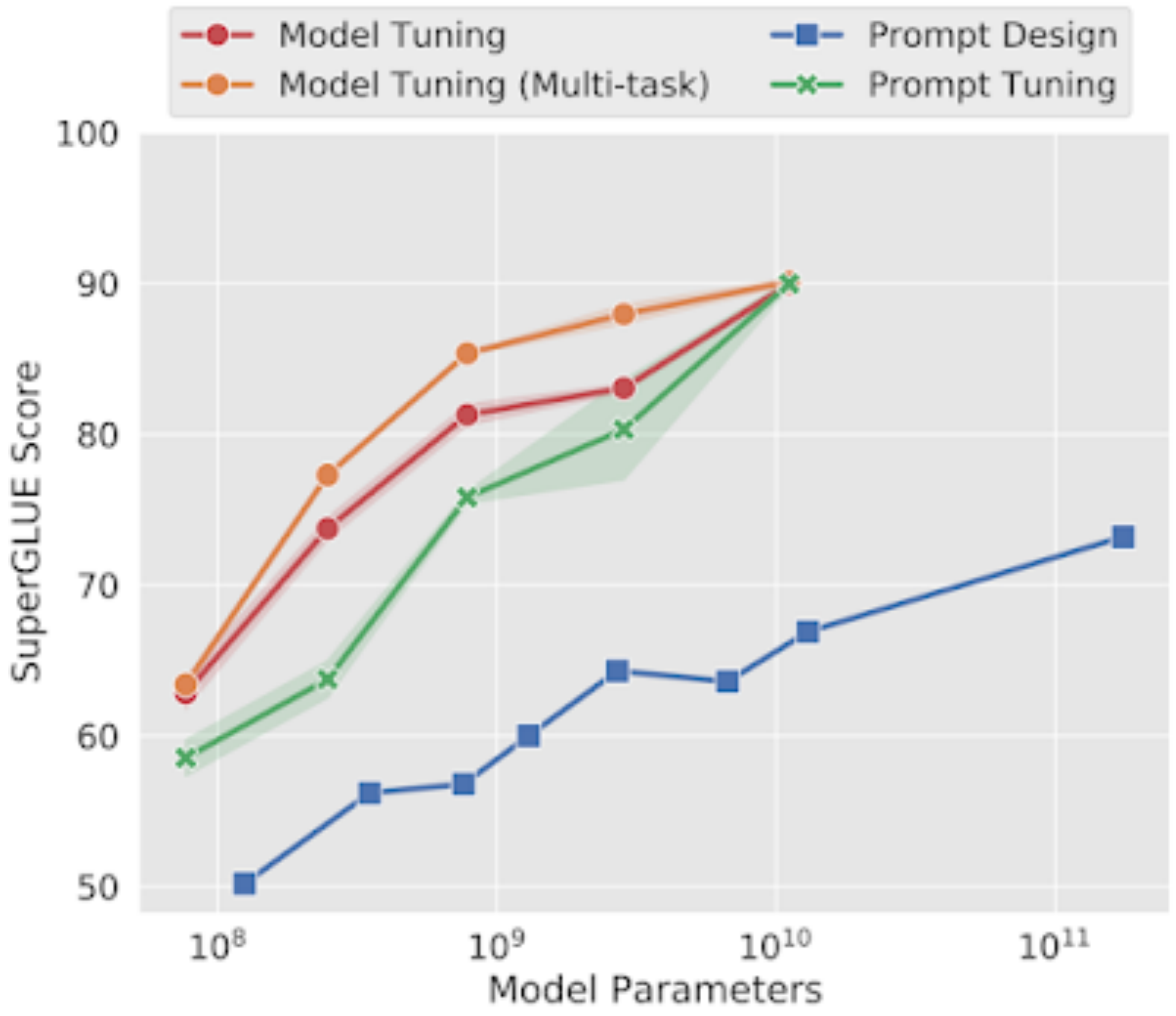


https://arxiv.org/abs/2104.08691

Prompt Tuning

- The length of the prompt directly affects the quality of the model
- However, the larger the model, the smaller the difference in quality





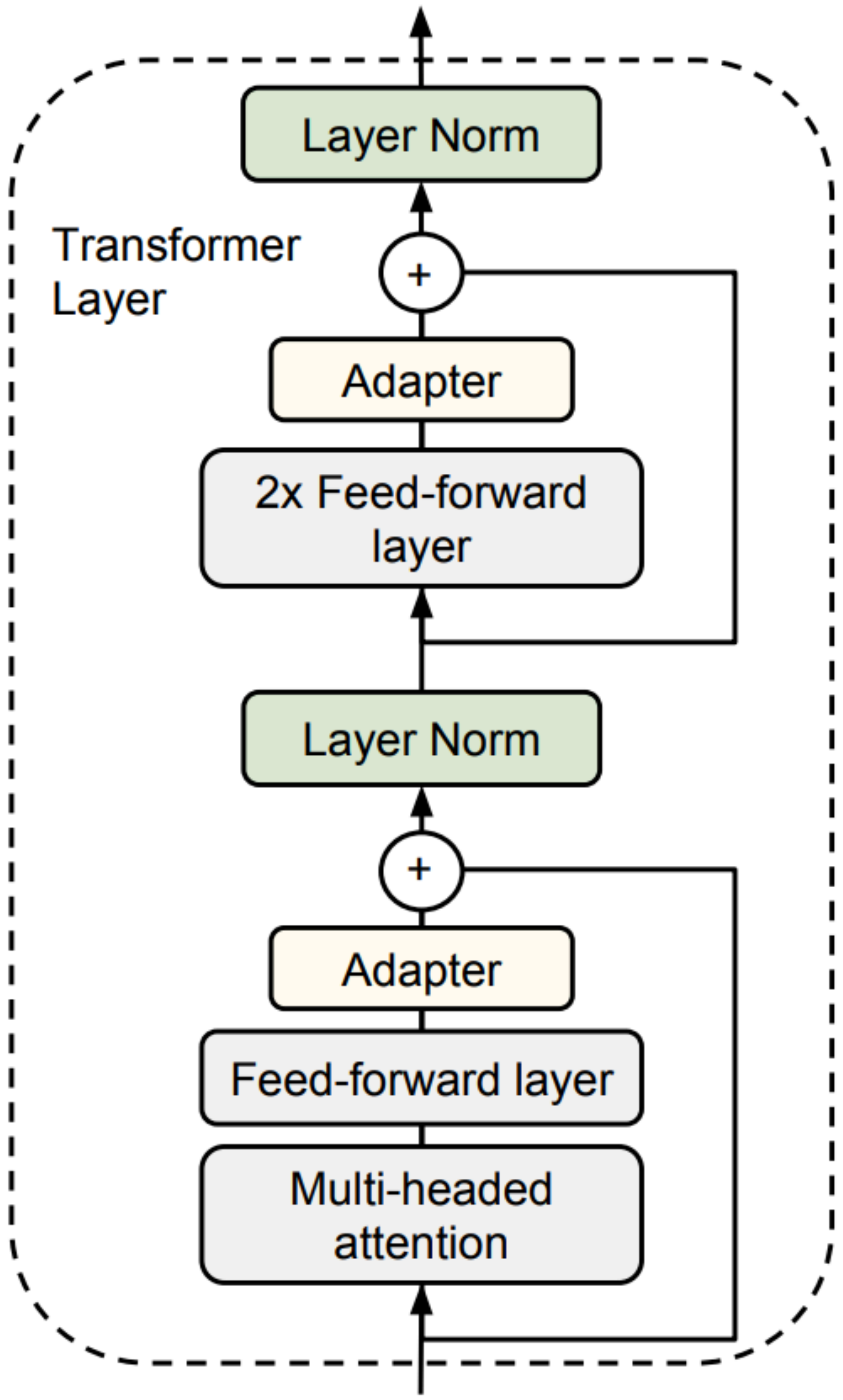
Prompt Tuning: Disadvantages

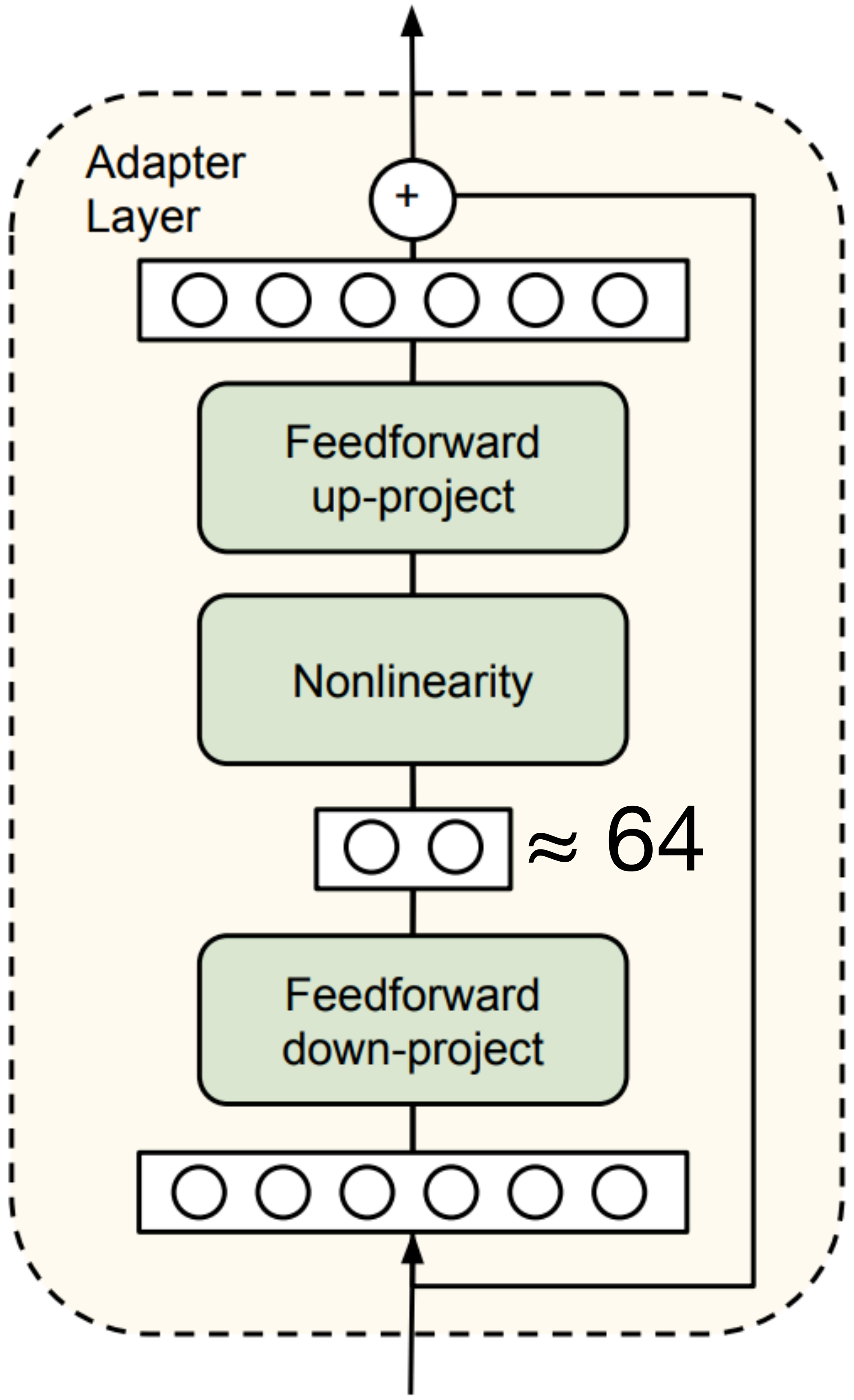
- Addition of a trained prompt limits the maximum length and slows down the model

- Prompt Tuning is very unstable and quality changes non-monotonically as the size of the prompt and model increases

Adapters

- After each attention layer and FFN, a **small** trainable adapter is added
- The adapter has skip-connection and two full-connection layers with dimensionality reduction (reduces the number of parameters)
- Only the adapters, normalizations and head are trained
- Such technique reaches fine-tuning in terms of quality





https://arxiv.org/abs/1902.00751

Adapters: Disadvantages

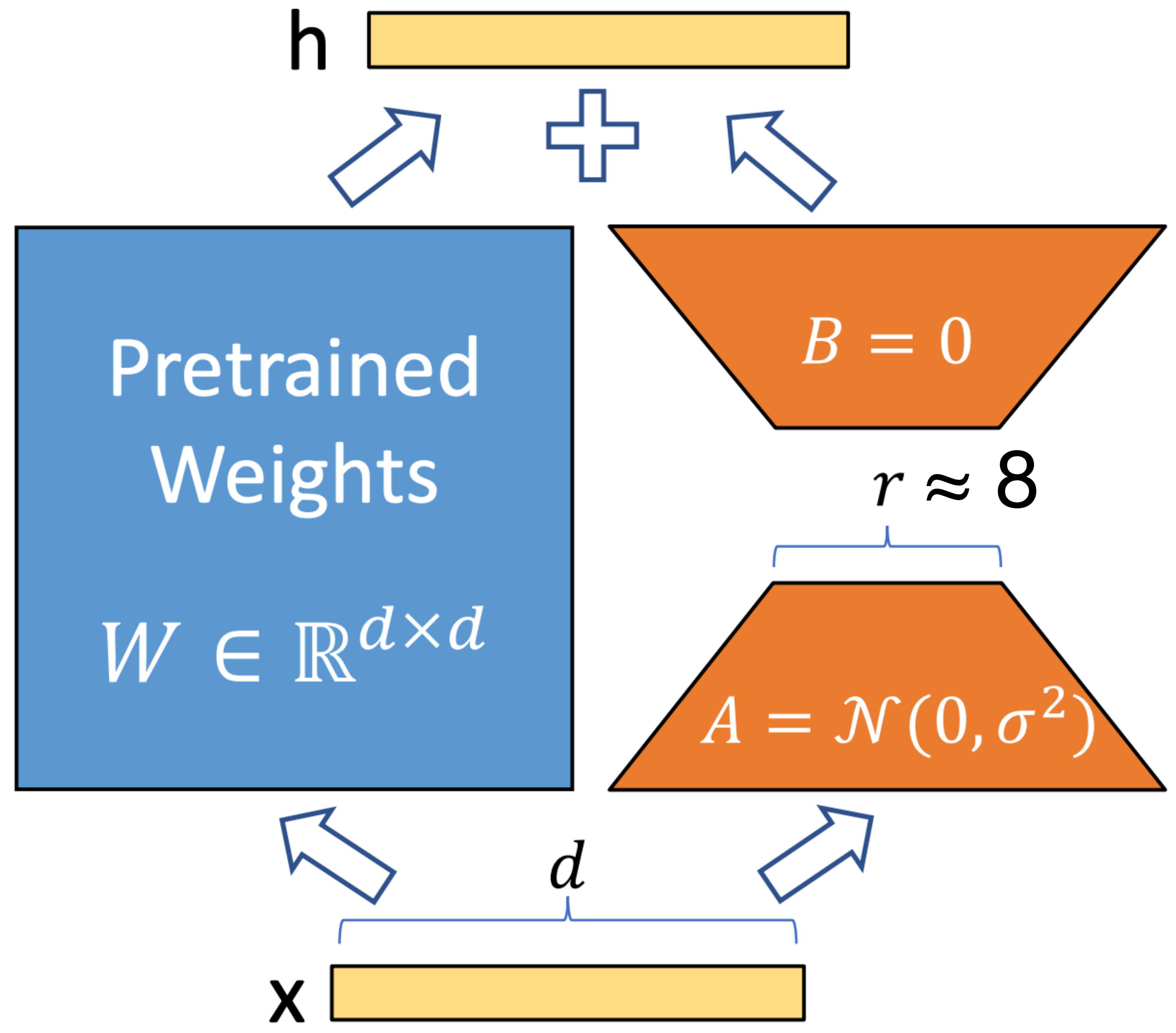
- The basic version of the adapter adds quite a few parameters compared to Prompt Tuning

- Adapters add extra layers that cannot be computed in parallel
- This slows down the model. Especially for small batch sizes

LoRA
Low-Rank Adaptation

- **Idea:** To adapt the model to the new task, we need to shift the weights towards the anti-gradient direction

$$W' = W + \delta W$$



https://arxiv.org/abs/2106.09685

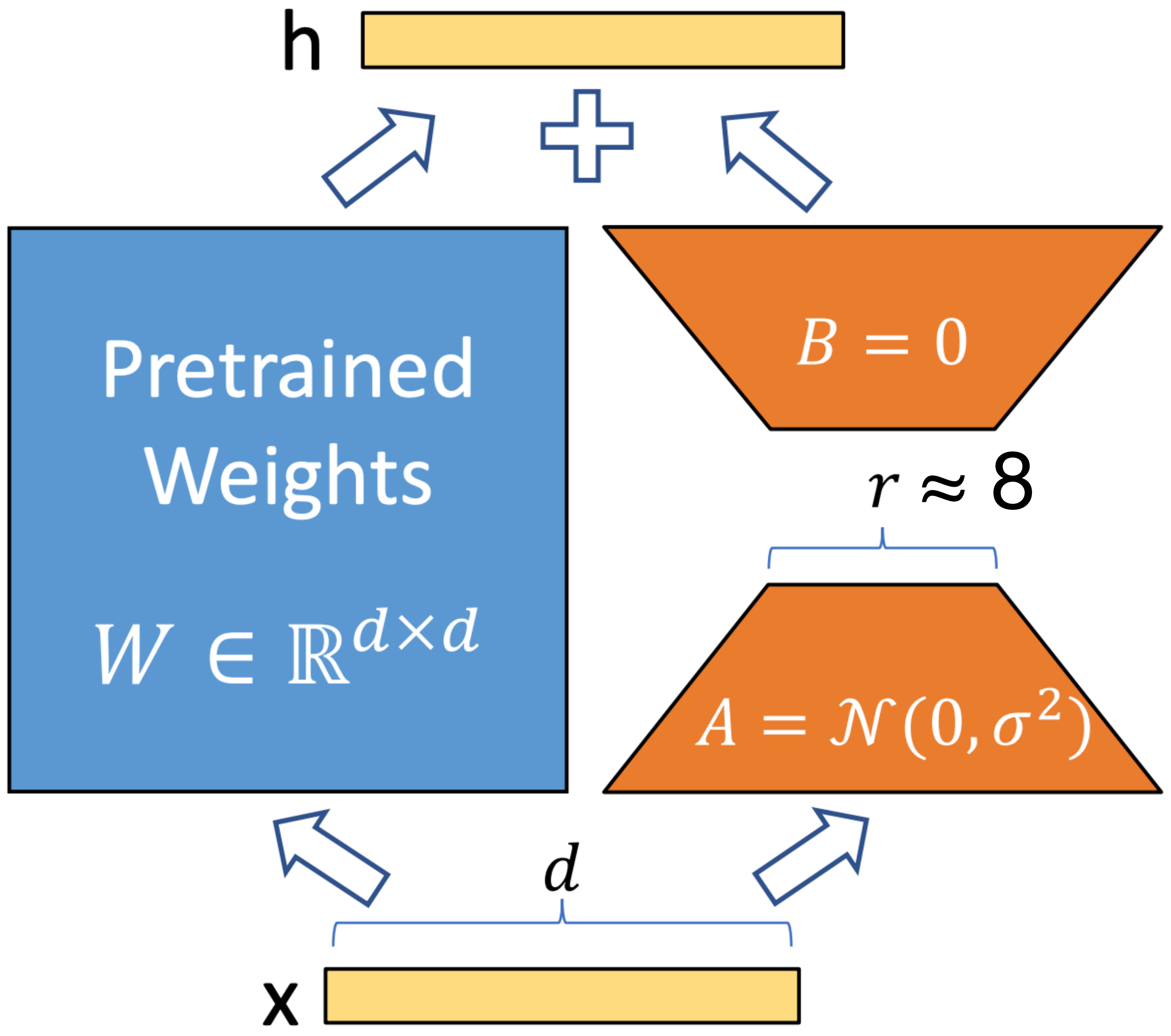
LoRA
Low-Rank Adaptation

- **Idea:** To adapt the model to the new task, we need to shift the weights towards the anti-gradient direction

$$W' = W + \delta W$$

- We approximate δW by the product of the training matrices AB

$$W' = W + AB$$



https://arxiv.org/abs/2106.09685

LoRA
Low-Rank Adaptation

- **Idea:** To adapt the model to the new task, we need to shift the weights towards the anti-gradient direction

$$W' = W + \delta W$$

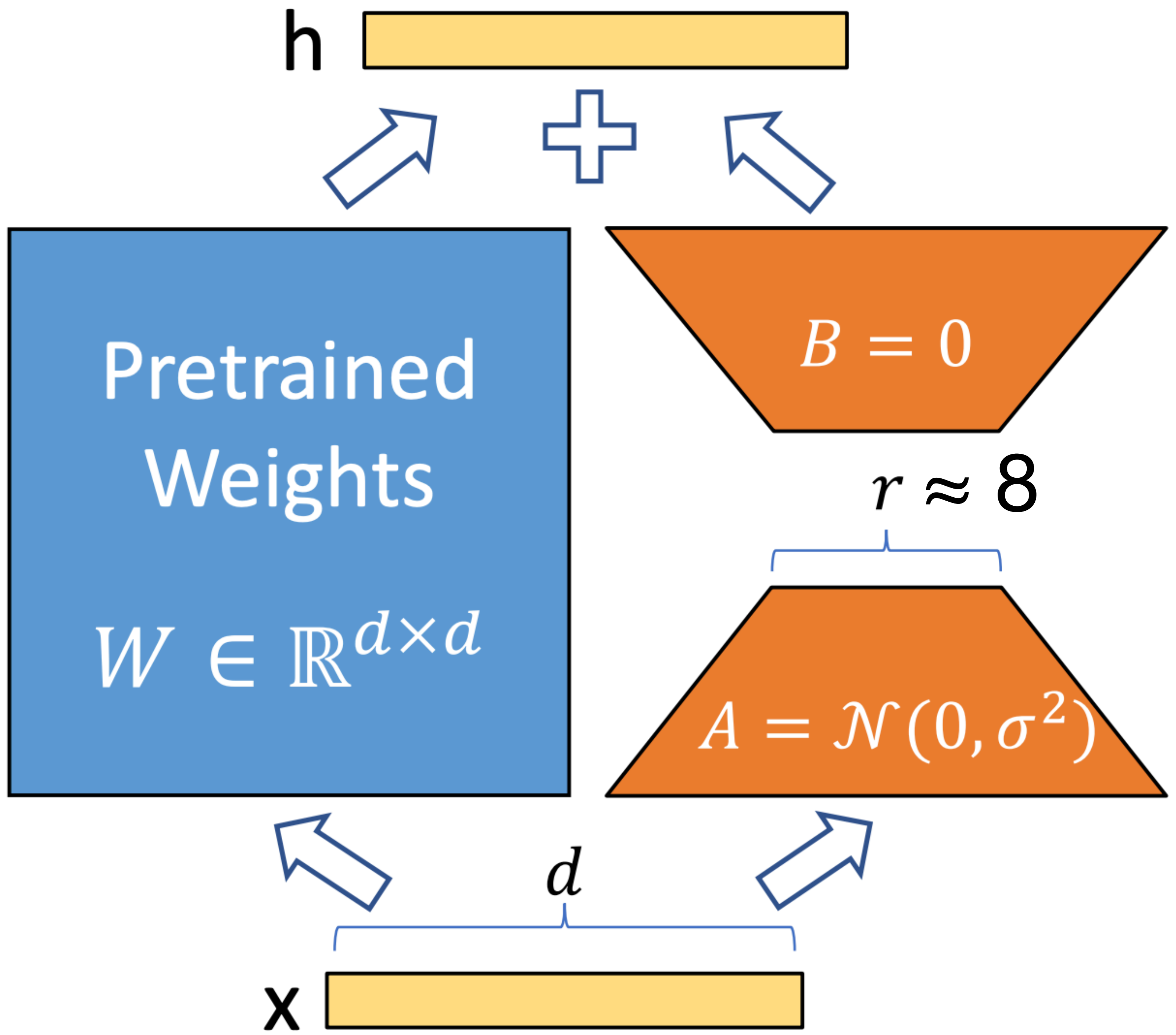
- We approximate δW by the product of the training matrices AB

$$W' = W + AB$$

- Only the matrices W_q and W_v of the attention mechanism are changed in this way

- This way very few parameters are added and the addition can be considered in parallel with the main block

- The most popular PEFT method



https://arxiv.org/abs/2106.09685

BitFit

Bias-terms Fine-tuning

Idea: biases have very few parameters, we will train only them

Attention

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

$$\mathbf{h}_1^\ell = att\left(\mathbf{Q}^{1,\ell}, \mathbf{K}^{1,\ell}, \mathbf{V}^{1,\ell}, .., \mathbf{Q}^{m,\ell}, \mathbf{K}^{m,\ell}, \mathbf{V}^{m,l}\right)$$

Feed Forward Network

$$\mathbf{h}_2^\ell = \mathrm{Dropout}\left(\mathbf{W}_{m_1}^\ell \cdot \mathbf{h}_1^\ell + \mathbf{b}_{m_1}^\ell\right)$$
$$\mathbf{h}_3^\ell = \mathbf{g}_{LN_1}^\ell \odot \frac{(\mathbf{h}_2^\ell + \mathbf{x}) - \mu}{\sigma} + \mathbf{b}_{LN_1}^\ell$$
$$\mathbf{h}_4^\ell = \mathrm{GELU}\left(\mathbf{W}_{m_2}^\ell \cdot \mathbf{h}_3^\ell + \mathbf{b}_{m_2}^\ell\right)$$
$$\mathbf{h}_5^\ell = \mathrm{Dropout}\left(\mathbf{W}_{m_3}^\ell \cdot \mathbf{h}_4^\ell + \mathbf{b}_{m_3}^\ell\right)$$
$$\mathrm{out}^\ell = \mathbf{g}_{LN_2}^\ell \odot \frac{(\mathbf{h}_5^\ell + \mathbf{h}_3^\ell) - \mu}{\sigma} + \mathbf{b}_{LN_2}^\ell$$

https://arxiv.org/abs/2106.10199

Comparison of methods

Suppose we have a transformer with 350M parameters and 24 layers.

Method	Number of parameters
Prompt tuning (length: 20)	15k (0.006%)
Adapters (dim 64)	6M (2%)
LoRA (dim 8)	0.8M (0.22%)
BitFit	0.32M (0.09%)

All methods show comparable quality on some tasks, but LoRA is the most stable and is used more often than the others

Comparison of methods

Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**

Model & Method	# Trainable Parameters	E2E NLG Challenge BLEU	NIST	MET	ROUGE-L	CIDEr
GPT-2 M (FT)*	354.92M	68.2	8.62	46.2	71.0	2.47
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M ($\text{Adapter}^{\text{L}}$)*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M ($\text{Adapter}^{\text{H}}$)	11.09M	$67.3_{\pm.6}$	$8.50_{\pm.07}$	$46.0_{\pm.2}$	$70.7_{\pm.2}$	$2.44_{\pm.01}$
GPT-2 M (FT^{Top2})*	25.19M	68.1	8.59	46.0	70.8	2.41
GPT-2 M (PreLayer)*	0.35M	69.7	8.81	46.1	71.4	2.49
GPT-2 M (LoRA)	0.35M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$

https://arxiv.org/pdf/2106.09685