PARAMETER EFFICIENT FINE-TUNING

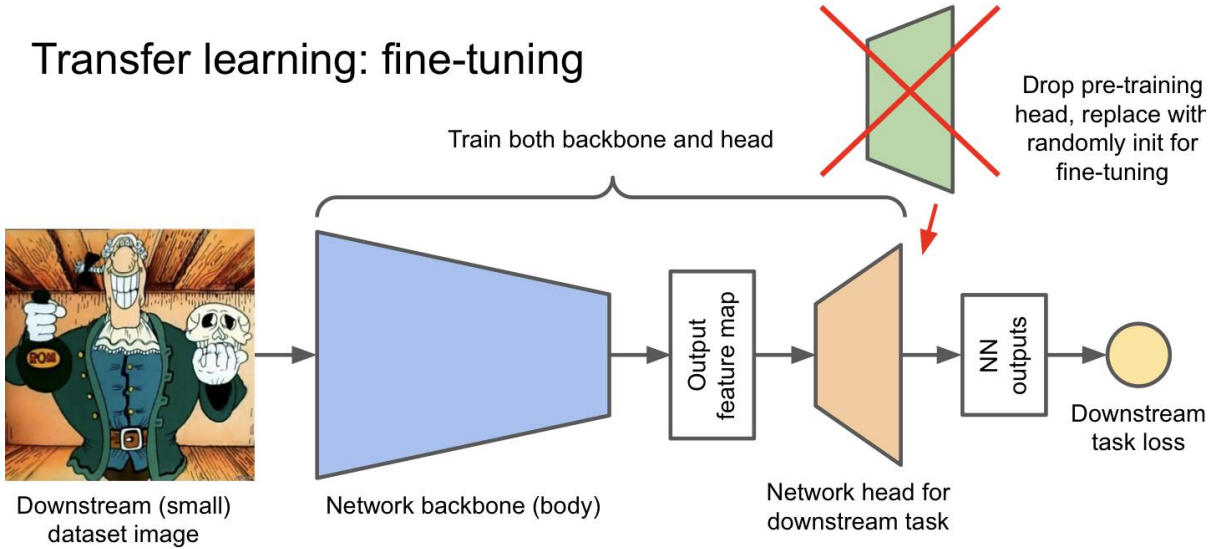
Файн-тюнинг малой части параметров модели

План доклада

- Что такое Fine-tuning?
- Идея Parameter Efficient Fine-Tuning (PEFT)
- Классификация методов PEFT
- Напоминание о трансформерах
- Метод Adapter
- Метод Prefix-Tuning
- Метод LoRA
- Заключение

FINE-TUNING

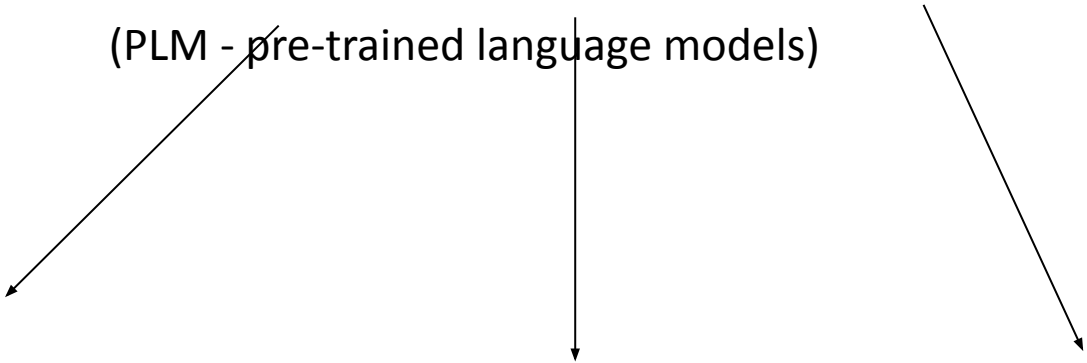
Fine-Tuning - процесс адаптации предварительно обученной модели к конкретной задаче помощью дополнительного обучения на специфическом наборе данных.



Проблема Fine-Tuning

Огромный размер числа параметров обученных языковых моделей

(PLM - pre-trained language models)



хранение многих параметров

вычислительные требования

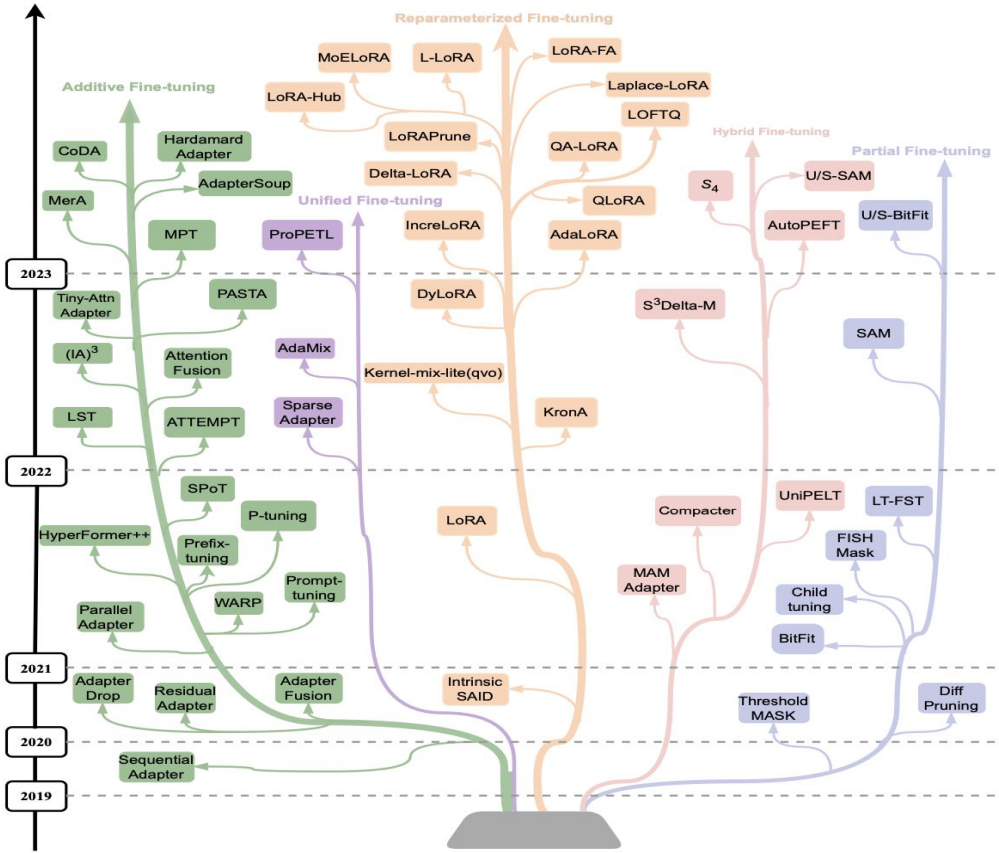
риск переобучения

Идея PEFT

Parameter Efficient Fine-Tuning (PEFT) - настройка малой части параметров, которая предлагает эффективные решения, сокращая количество обучаемых параметров и использование памяти при этом обеспечивается производительность сравнимая с полной точной настройкой (FT).

Классификация методов PEFT

Рис. 1: Развитие методов PEFT



Трансформер

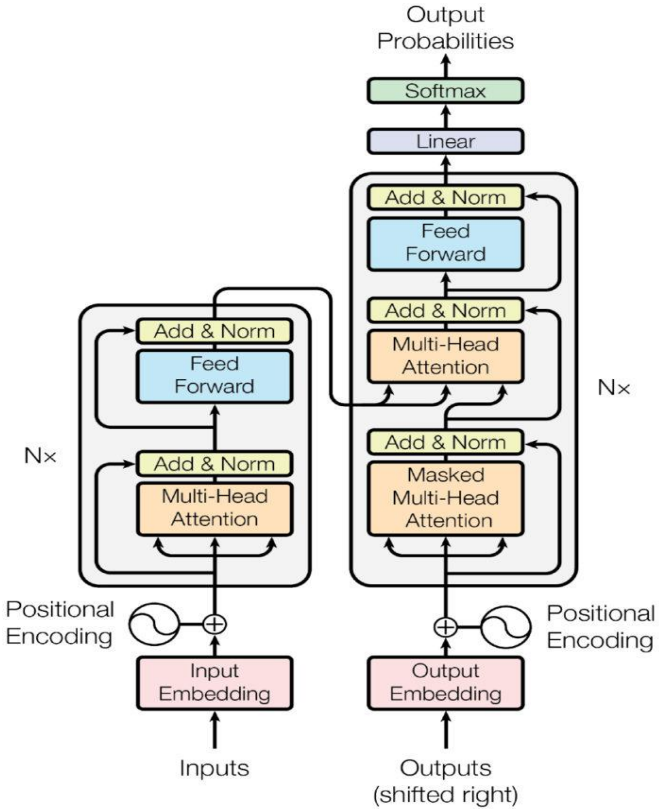


Рис. 2: Блок трансформера

Техники Fine-Tuning в NLP

Multi-task Learning

Обучение модели на несколько задач одновременно, используя общий набор параметров или часть параметров

Continual Learning

Последовательное переобучение под каждую новую задачу

Adapter Tuning

Хочется придумать технику, которая позволит решить следующие проблемы:

- Fine-Tuning обучает все веса модели, мы хотим обучать только малую часть весов
- Multi-task learning подразумевает одновременное решение задач, хочется решать несколько задач, при этом не имея доступа ко всем задачам одновременно
- Continual learning подразумевает дообучение имеющийся модели под новую задачу, хотим не терять способность решать старую задачу

Идея Adapter Tuning

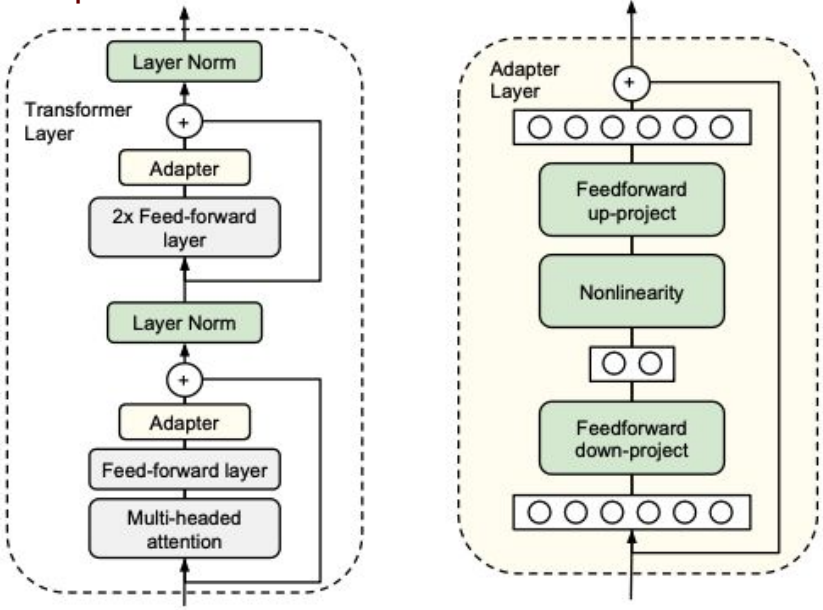
Рассмотрим функцию $\boldsymbol{w}$: $\phi_{\boldsymbol{w}}(\boldsymbol{x})$

Adapter tuning

Этот метод предлагает ввести новую функцию $\psi_{w,v}(x)$, где w это в точности те же параметры, что и в ϕ_w. Новые параметры v инициализируются так, чтобы новая функция повторяла исходную: $\psi_{w,v_0}(x) \approx \phi_w(x)$. Во время обучения обновляются только веса v, а w остаются неизменными. В идеале мы хотим строить функции ψ такие, что $|v| \ll |w|$, чтобы размер модели практически не возрастал при добавлении новых параметров.

Как это сделать?

Рис. 3: Слева блок трансформера, справа слой адаптера



Adapter

Слой адаптера по смыслу сначала проецирует d-мерные вектора признаков в меньшее m-мерное пространство, применяют нелинейность и проецируют вектора обратно в d-мерное пространство, добавив skip connection.

Сколько у нас обучаемых параметров?

Количество параметров

Заметим, что количество обучаемых параметров в одном adapter слое будет $2md + d + m$. Если брать $m \ll d$, мы можем легко ограничить количество добавляемых параметров для решения новой задачи. На практике можно добиться добавления всего $0.5 - 8\%$ параметров от исходной модели.

Инициализация

Если инициализировать веса adapter слоя околонулевыми числами, то skip connection слой позволит новой модели имитировать исходную модель.

Эксперименты

На GLUE бенчмарке Adapter Tuning позволяет достичь результат в диапазоне 0.4% от FT всей модели BERT, при этом Adapter Tuning добавляет и обучает всего 3% от общего количества параметров модели

	Total num params	Trained params / task	CoLA	SST	MRPC	STS-B	QQP	MNLI_m	MNLI_mm	QNLI	RTE	Total
BERT_LARGE	9.0×	100%	60.5	94.9	89.3	87.6	72.1	86.7	85.9	91.1	70.1	80.4
Adapters (8-256)	1.3×	3.6%	59.5	94.0	89.5	86.9	71.8	84.9	85.1	90.7	71.5	80.0
Adapters (64)	1.2×	2.1%	56.9	94.2	89.6	87.3	71.8	85.3	84.6	91.4	68.8	79.6

Рис. 4: Результаты на GLUE бенчмарке

Эксперименты

Dataset	No BERT baseline	$\text{BERT}_{\text{BASE}}$ Fine-tune	$\text{BERT}_{\text{BASE}}$ Variable FT	$\text{BERT}_{\text{BASE}}$ Adapters
20 newsgroups	91.1	92.8 ± 0.1	92.8 ± 0.1	91.7 ± 0.2
Crowdflower airline	84.5	83.6 ± 0.3	84.0 ± 0.1	84.5 ± 0.2
Crowdflower corporate messaging	91.9	92.5 ± 0.5	92.4 ± 0.6	92.9 ± 0.3
Crowdflower disasters	84.9	85.3 ± 0.4	85.3 ± 0.4	84.1 ± 0.2
Crowdflower economic news relevance	81.1	82.1 ± 0.0	78.9 ± 2.8	82.5 ± 0.3
Crowdflower emotion	36.3	38.4 ± 0.1	37.6 ± 0.2	38.7 ± 0.1
Crowdflower global warming	82.7	84.2 ± 0.4	81.9 ± 0.2	82.7 ± 0.3
Crowdflower political audience	81.0	80.9 ± 0.3	80.7 ± 0.8	79.0 ± 0.5
Crowdflower political bias	76.8	75.2 ± 0.9	76.5 ± 0.4	75.9 ± 0.3
Crowdflower political message	43.8	38.9 ± 0.6	44.9 ± 0.6	44.1 ± 0.2
Crowdflower primary emotions	33.5	36.9 ± 1.6	38.2 ± 1.0	33.9 ± 1.4
Crowdflower progressive opinion	70.6	71.6 ± 0.5	75.9 ± 1.3	71.7 ± 1.1
Crowdflower progressive stance	54.3	63.8 ± 1.0	61.5 ± 1.3	60.6 ± 1.4
Crowdflower US economic performance	75.6	75.3 ± 0.1	76.5 ± 0.4	77.3 ± 0.1
Customer complaint database	54.5	55.9 ± 0.1	56.4 ± 0.1	55.4 ± 0.1
News aggregator dataset	95.2	96.3 ± 0.0	96.5 ± 0.0	96.2 ± 0.0
SMS spam collection	98.5	99.3 ± 0.2	99.3 ± 0.2	95.1 ± 2.2
Average	72.7	73.7	74.0	73.3
Total number of params	—	17×	9.9×	1.19×
Trained params/task	—	100%	52.9%	1.14%

Рис. 5: Точность для различных задач классификации

Prefix-Tuning

Вторая техника - Prefix Tuning. Пытается решить те же проблемы, что и Adapter Tuning.

Задачи

Авторы статьи предлагают применить эту технику для задач типа table-to-text и summarization.

- Table-to-text — генерация текстового описания объекта по структурированному табличному описанию. Например, из табличного представления (name : 'Starbucks', type : 'coffee shop') хотим получить текстовое описание: 'Starbucks serves coffee'.
- Summarization — генерация краткого текстового описания более длинного исходного текста.

Интуиция Prefix-Tuning

В статье предполагают, что наличие контекста может хорошо управлять языковой моделью. Например: чтобы сгенерировать слово Obama, мы можем добавить его общие словосочетания в виде контекста, например Barack. Расширяя эту интуицию помимо создания одного слова или предложения, хочется найти контекст, который позволит решать задачу NLG (natural language generation).

Так давайте добавлять префиксы (подсказки)!

Сравнение FT и Prefix-Tuning

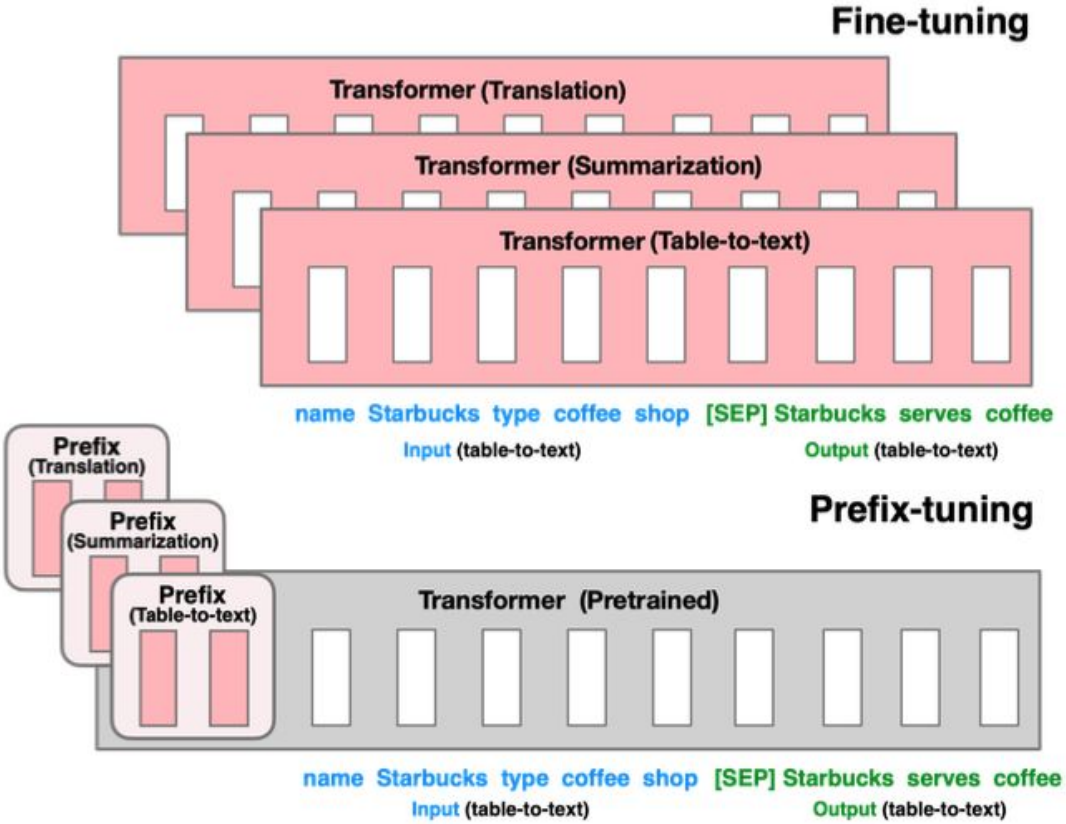
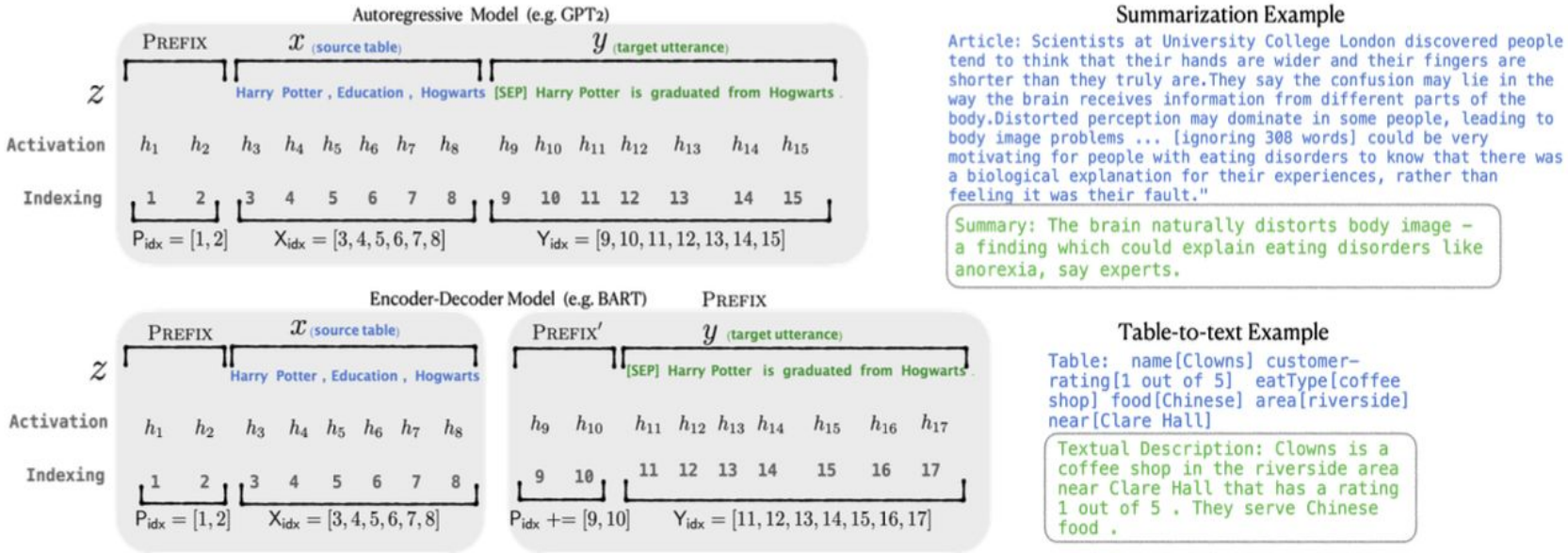


Рис. 6: Сравнение количества обучаемых параметров в FT и в Prefix-Tuning

Куда добавляем префиксы?

Рис. 7: Добавление префиксов



Summarization Example

Article: Scientists at University College London discovered people tend to think that their hands are wider and their fingers are shorter than they truly are.They say the confusion may lie in the way the brain receives information from different parts of the body.Distorted perception may dominate in some people, leading to body image problems ... [ignoring 308 words] could be very motivating for people with eating disorders to know that there was a biological explanation for their experiences, rather than feeling it was their fault."

Summary: The brain naturally distorts body image - a finding which could explain eating disorders like anorexia, say experts.

Table-to-text Example

Table: name[Clowns] customer-rating[1 out of 5] eatType[coffee shop] food[Chinese] area[riverside] near[Clare Hall]

Textual Description: Clowns is a coffee shop in the riverside area near Clare Hall that has a rating 1 out of 5 . They serve Chinese food .

Модели

Авторегрессионая языковая модель

Предположим, что у нас есть модель $p_\phi(y|x)$, основанная на трансформере и параметризованная от ϕ. Пусть $z = [x; y]$ — конкатенация входных и выходных данных одного примера. Так как это авторегрессионая модель, то можем ввести вектора активации $h_i = [h_i^{(1)}; \ldots; h_i^{(n)}]$, где i отвечает за момент времени i (то есть когда мы будем пихать в модель i-ый токен из z), а верхний индекс отвечает за номер слоя модели. Тогда модель будет вычислять эти вектора как:

$$h_i = LM_\phi(z_i, h_{<i}),$$

а распределение вероятностей для предсказания следующего токена вычисляется как софтмакс от классификатора после последнего слоя $h_i^{(n)}$.

Encoder-Decoder

Эта архитектура отличается только тем, что вектора h_i для входных данных считаются в энкодере, а для выходных данных считаются в декодере.

Обучение в Prefix-Tuning

База

В авторегрессионной модели допихиваем префикс как $z = [\text{PREFIX}; x; y]$, а в encoder-decoder $z = [\text{PREFIX}; x; \text{PREFIX'}; y]$. Пусть P_{idx} — последовательность индексов добавленных в z префиксов. Тогда мы хотим обучать матрицу P_θ, хранящую вектора активации для префиксов:

$$h_i = \begin{cases} P_\theta[i, :], & \text{if } i \in P_{idx}, \\ LM_\phi(z_i, h_{<i}), & \text{otherwise.} \end{cases}$$

Фундамент

Учить матрицу P_θ в лоб выходит неэффективно, поэтому можно представить её в виде $P_\theta[i, :] = MLP_\theta(P'_\theta[i, :])$ с помощью меньшей матрицы P'_θ и нейронной сети MLP_θ.

Что мы учим?

В FT мы инициализируем веса уже обученные. И тогда чтобы найти распределение языковой модели мы делаем градиентный спуск по логарифму правдоподобия.

$$\max_{\phi} \log p_\phi(y \mid x) = \sum_{i \in \mathrm{Y_{idx}}} \log p_\phi(z_i \mid h_{<i}). \quad (2)$$

В Prefix-Tuning мы учим матрицу только для префиксов, как и в задаче (2).

Блок трансформера

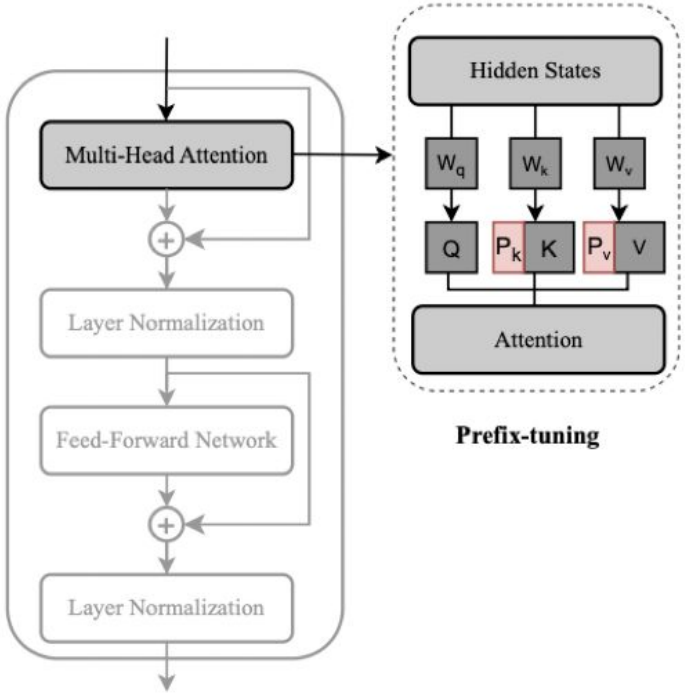


Рис. 8: Блок трансформера при Prefix-Tuning

Эксперименты

	E2E					WebNLG									DART					
	BLEU	NIST	MET	R-L	CIDEr	BLEU			MET			TER ↓			BLEU	MET	TER ↓	Mover	BERT	BLEURT
						S	U	A	S	U	A	S	U	A						
GPT-2$_{\text{MEDIUM}}$																				
FINE-TUNE	68.2	8.62	**46.2**	71.0	2.47	**64.2**	27.7	46.5	**0.45**	0.30	0.38	**0.33**	0.76	0.53	46.2	**0.39**	**0.46**	**0.50**	**0.94**	**0.39**
FT-TOP2	68.1	8.59	46.0	70.8	2.41	53.6	18.9	36.0	0.38	0.23	0.31	0.49	0.99	0.72	41.0	0.34	0.56	0.43	0.93	0.21
ADAPTER(3%)	68.9	8.71	46.1	71.3	2.47	60.4	**48.3**	54.9	0.43	**0.38**	**0.41**	0.35	**0.45**	**0.39**	45.2	0.38	**0.46**	**0.50**	**0.94**	**0.39**
ADAPTER(0.1%)	66.3	8.41	45.0	69.8	2.40	54.5	45.1	50.2	0.39	0.36	0.38	0.40	0.46	0.43	42.4	0.36	0.48	0.47	**0.94**	0.33
PREFIX(0.1%)	**69.7**	**8.81**	46.1	**71.4**	**2.49**	62.9	45.6	**55.1**	0.44	**0.38**	**0.41**	0.35	0.49	0.41	**46.4**	0.38	**0.46**	**0.50**	**0.94**	**0.39**
GPT-2$_{\text{LARGE}}$																				
FINE-TUNE	68.5	8.78	46.0	69.9	2.45	**65.3**	43.1	55.5	**0.46**	0.38	**0.42**	**0.33**	0.53	0.42	**47.0**	**0.39**	0.46	**0.51**	**0.94**	**0.40**
Prefix	**70.3**	**8.85**	**46.2**	**71.7**	**2.47**	63.4	**47.7**	**56.3**	0.45	**0.39**	**0.42**	0.34	**0.48**	**0.40**	46.7	**0.39**	**0.45**	**0.51**	**0.94**	**0.40**
SOTA	68.6	8.70	45.3	70.8	2.37	63.9	52.8	57.1	0.46	0.41	0.44	-	-	-	-	-	-	-	-	-

Table 1: Metrics (higher is better, except for TER) for table-to-text generation on E2E (left), WebNLG (middle) and DART (right). With only 0.1% parameters, Prefix-tuning outperforms other lightweight baselines and achieves a comparable performance with fine-tuning. The best score is boldfaced for both GPT-2$_{\text{MEDIUM}}$ and GPT-2$_{\text{LARGE}}$.

Рис. 9: Метрики для задач table-to-text

Эксперименты

Рис. 10: Метрики для задач summarization

	R-1 ↑	R-2 ↑	R-L ↑
FINE-TUNE(Lewis et al., 2020)	45.14	22.27	37.25
PREFIX(2%)	43.80	20.93	36.05
PREFIX(0.1%)	42.92	20.03	35.05

Table 2: Metrics for summarization on XSUM. Prefix-tuning slightly underperforms fine-tuning.

	news-to-sports			within-news		
	R-1 ↑	R-2 ↑	R-L ↑	R-1 ↑	R-2 ↑	R-L ↑
FINE-TUNE	38.15	15.51	30.26	39.20	16.35	31.15
PREFIX	39.23	16.74	31.51	39.41	16.87	31.47

Table 3: Extrapolation performance on XSUM. Prefix-tuning outperforms fine-tuning on both news-to-sports and within-news splits.

LoRA

Рассмотрим последнюю модель: LoRA (Low-Rank Adaptation of LLM)
Мы также хотим придумать эффективный способ дообучения исходной модели под наши задачи. Прошлые методы (Adapter и prefix-tuning) решили многие проблемы, но у них есть недостатки:

Минусы Adapter:

- нельзя распараллелить
- требуется хранить много данных

Минусы Prefix tuning:

- сложно оптимизировать настройку префикса
- доступная длина последовательности

Общие минусы:

- увеличение inferency latency
- плохо комбинируют друг с другом

Постановка задачи

Модель

Пусть у нас имеется авторегрессионая модель $P_\Phi(y|x)$ параметризованная от Φ, например, GPT. Пусть мы уже имеем обученные веса Φ_0 на какую-то общую задачу и хотим, чтобы эта модель научилась решать другую задачу с датасетом $Z = \{(x_i, y_i)\}_{i=1,\dots,N}$.

Fine-tuning

Ищем такое $\Delta\Phi$, максимизирующее функцию правдоподобия на новом датасете:

$$\max_{\Delta\Phi} \sum_{(x,y)\in Z} \sum_{t=1}^{|y|} \log\left(P_{\Phi_0+\Delta\Phi}(y_t|x, y_{<t})\right).$$

При таком подходе нам надо хранить дополнительно $|\Delta\Phi| \sim |\Phi|$ параметров.

Идея

Перепараметризация

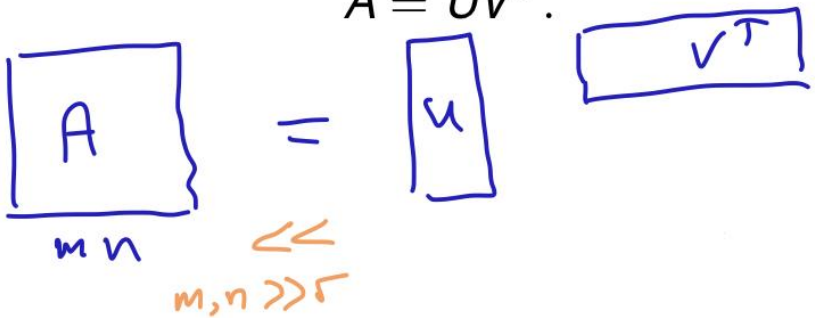
Давайте попробуем найти новое пространство параметров Θ такое, что оно будет достаточно хорошо приближать $\Delta\Phi(\Theta) \sim \Delta\Phi$, но при этом оно будет иметь сильно меньшее количество параметров $|\Theta| \ll |\Phi|$. Для этого нам необходимо вспомнить одно из часто забываемых определений ранга матрицы.

Скелетное разложение

Теорема

Для любой матрицы $A \in \mathbb{F}^{m \times n}$, $\mathbb{F} \in \{\mathbb{R}, \mathbb{C}\}$ *ранга* r *найдутся* $U \in \mathbb{F}^{m \times r}$ *и* $V \in \mathbb{F}^{n \times r}$*:*

$$A = UV^{\top}.$$



Как это нам поможет?

Для матрицы весов $W_0 \in \mathbb{R}^{d \times k}$ из исходной модели будем приближать ΔW с помощью low-rank декомпозиции $\Delta W = BA$, где $B \in \mathbb{R}^{d \times r}$ и $A \in \mathbb{R}^{r \times k}$ и $r \ll \min(d, k)$. Тогда получится:

$$h = W_0 x + \Delta W x = W_0 x + BAx$$

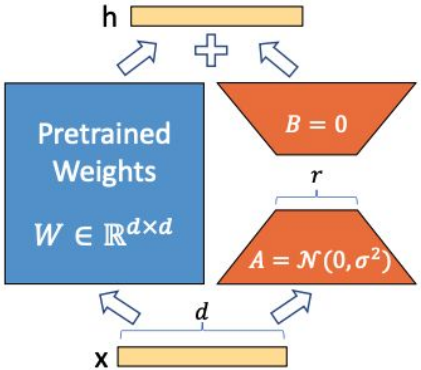
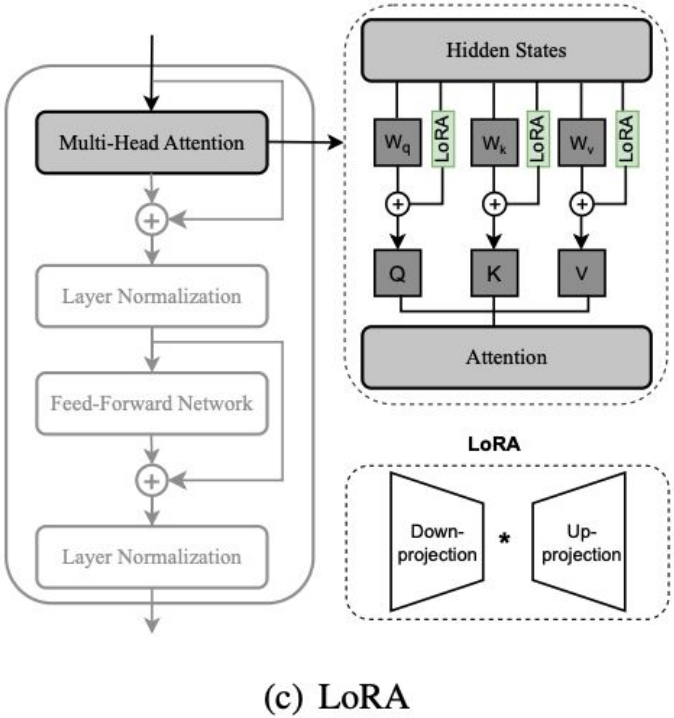


Рис. 11: Новая параметризация

Блок трансформера



(c) LoRA

Рис. 12: Блок трансформера в LoRA

Преимущества

Преимущества:

- Обобщение полного fine-tuning — увеличивая r данный метод сойдётся в честное представление матриц ΔW.
- Нулевой inference latency — будем нашу модель хранить в виде суммы исходных весов и произведения новых матриц $W = W_0 + BA$. Тогда новая модель будет иметь те же характеристики, что и исходная.
- Простое переключение задач — достаточно из текущей модели отнять все BA и добавить $B'A'$ новой задачи. Для запоминания моделей новых задач требуется супер мало памяти.
- Вычислительные преимущества — есть какие-то преимущества в виде меньшего использования VRAM и более быстрого обучения модели.

Гиперпараметры

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Рис. 13: Подбор гиперпараметров

Эксперименты

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**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

Table 2: $\text{RoBERTa}_{\text{base}}$, $\text{RoBERTa}_{\text{large}}$, and $\text{DeBERTa}_{\text{XXL}}$ with different adaptation methods on the GLUE benchmark. We report the overall (matched and mismatched) accuracy for MNLI, Matthew's correlation for CoLA, Pearson correlation for STS-B, and accuracy for other tasks. Higher is better for all metrics. * indicates numbers published in prior works. † indicates runs configured in a setup similar to Houlsby et al. (2019) for a fair comparison.

Рис. 14: Результаты экспериментов

Эксперименты

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 (Adapter^{L})*	0.37M	66.3	8.41	45.0	69.8	2.40
GPT-2 M (Adapter^{L})*	11.09M	68.9	8.71	46.1	71.3	2.47
GPT-2 M (Adapter^{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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L (Adapter^{L})	0.88M	$69.1_{\pm.1}$	$8.68_{\pm.03}$	$46.3_{\pm.0}$	$71.4_{\pm.2}$	$\mathbf{2.49}_{\pm.0}$
GPT-2 L (Adapter^{L})	23.00M	$68.9_{\pm.3}$	$8.70_{\pm.04}$	$46.1_{\pm.1}$	$71.3_{\pm.2}$	$2.45_{\pm.02}$
GPT-2 L (PreLayer)*	0.77M	70.3	8.85	46.2	71.7	2.47
GPT-2 L (LoRA)	0.77M	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

Table 3: GPT-2 medium (M) and large (L) with different adaptation methods on the E2E NLG Challenge. For all metrics, higher is better. LoRA outperforms several baselines with comparable or fewer trainable parameters. Confidence intervals are shown for experiments we ran. * indicates numbers published in prior works.

Рис. 15: Результаты экспериментов для GPT-2

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 (Adapter^{H})	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 (Adapter^{H})	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

Table 4: Performance of different adaptation methods on GPT-3 175B. We report the logical form validation accuracy on WikiSQL, validation accuracy on MultiNLI-matched, and Rouge-1/2/L on SAMSum. LoRA performs better than prior approaches, including full fine-tuning. The results on WikiSQL have a fluctuation around $\pm 0.5\%$, MNLI-m around $\pm 0.1\%$, and SAMSum around $\pm 0.2/\pm 0.2/\pm 0.1$ for the three metrics.

Рис. 16: Результаты экспериментов для GPT-3

Заключение

Все 3 метода являются хорошим инструментом для неполного Fine-Tuning. Лучшим является LoRA. Уже было сказано много плюсов этого метода, но можно отметить и минус - сложность одновременной обработки входных данных для разных задач с различными параметрами A и B

Также в LoRA есть возможность комбинировать ее с другими методами