Алихан Зиманов

Факультет компьютерных наук
НИУ ВШЭ

Parameter-Efficient Fine-Tuning

feat. Илья Пахалко

НИС, Москва, 2023

Содежрание

Transfer Learning

Definition

Применение опыта, полученного при решении одной задачи для решения новой задачи из той же области.

Transfer Learning

Definition

Применение опыта, полученного при решении одной задачи для решения новой задачи из той же области.

Example (Классификация изображений)

ResNet, обученный на ImageNet будет содержать в себе информацию о паттернах в картинках в целом, поэтому можно эту абстрактную информацию переиспользовать для решения других, более узких задач.

Transfer Learning

Definition

Применение опыта, полученного при решении одной задачи для решения новой задачи из той же области.

Example (Классификация изображений)

ResNet, обученный на ImageNet будет содержать в себе информацию о паттернах в картинках в целом, поэтому можно эту абстрактную информацию переиспользовать для решения других, более узких задач.

Example (Языковая модель)

Эмбеддинги, обученные на колоссальных датасетах будут содержать в себе сложную информацию о структуре языка, поэтому их переиспользование для других задач NLP будет весьма удобным.

Подходы в NLP

Pre-trained text representations (feature-based transfer)

Использование эбмеддингов, обученных на огромных датасетах для получения полезных признаковых описаний токенов.

Подходы в NLP

Pre-trained text representations (feature-based transfer)

Использование эбмеддингов, обученных на огромных датасетах для получения полезных признаковых описаний токенов.

Fine-tuning

Использование уже обученной, но на другую задачу модели как инициализацию весов модели, решающей текущую задачу.

Подходы в NLP

Pre-trained text representations (feature-based transfer)

Использование эбмеддингов, обученных на огромных датасетах для получения полезных признаковых описаний токенов.

Fine-tuning

Использование уже обученной, но на другую задачу модели как инициализацию весов модели, решающей текущую задачу.

Parameter-Efficient Fine-Tuning

Fine-tuning, обучающий как можно меньшее количество добавленных параметров или параметров исходной модели, при этом не теряющий точности решения задачи.

Техники обучения в NLP

Multi-task Learning

Обучение модели на несколько задач одновременно. У модели имеются общие по всем задачам глубокие слои и специализированные под каждую задачу верхние слои.

Continual Learning

Последовательное переобучение[a] модели под каждую новую задачу.

[a] не overfitting, a re-training

Adapter Tuning

Первая рассматриваемая техника PEFT это Adapter Tuning [1]. Хочется придумать такую технику, которая позволит решить следующие проблемы:

Adapter Tuning

Первая рассматриваемая техника PEFT это Adapter Tuning [1]. Хочется придумать такую технику, которая позволит решить следующие проблемы:

- Fine-tuning обучает все веса модели, что тяжело и хочется обучать только малую часть весов;

Adapter Tuning

Первая рассматриваемая техника PEFT это Adapter Tuning [1]. Хочется придумать такую технику, которая позволит решить следующие проблемы:

- Fine-tuning обучает все веса модели, что тяжело и хочется обучать только малую часть весов;
- Multi-task learning подразумевает одновременное решение задач, что иногда бывает сложным, поэтому хочется уметь решать несколько задач, при этом не имея доступ ко всем задачам одновременно;

Adapter Tuning

Первая рассматриваемая техника PEFT это Adapter Tuning [1]. Хочется придумать такую технику, которая позволит решить следующие проблемы:

- Fine-tuning обучает все веса модели, что тяжело и хочется обучать только малую часть весов;
- Multi-task learning подразумевает одновременное решение задач, что иногда бывает сложным, поэтому хочется уметь решать несколько задач, при этом не имея доступ ко всем задачам одновременно;
- Continual learning подразумевает дообучение имеющейся модели под новую задачу, при этом либо способность модели решать старую задачу утрачивается и сильно страдает качество, либо количество обучаемых параметров линейно возрастает с каждой следующей задачей.

Махания руками

Рассмотрим функцию (нейронную сеть) с параметрами w: $\phi_w(x)$.

Feature-based Transfer

Этот метод предлагает использовать композицию ϕ_w с новой функцией χ_v, получая, таким образом, новую нейронную сеть $\chi_v(\phi_w(x))$. После этого обучаются только веса v, специфичные под текущую задачу.

Fine-tuning

Этот метод предлагает дообучение имеющихся параметров w под новую задачу, ограничивая компактность[a] модели.

[a] компактная модель использует малое количество дополнительных параметров для решения новых задач

Махания руками 2.0

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.

Махания руками 2.0

Adapter tuning

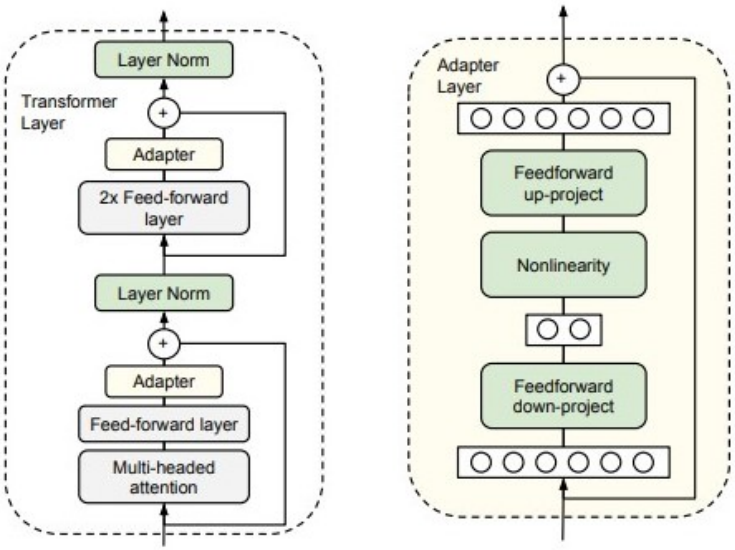
Этот метод предлагает ввести новую функцию $\psi_{w,v}(x)$, где w это в точности те же параметры, что и в ϕ_w. Новые параметры v инициализируются так, чтобы новая функция повторяла исходную: $\psi_{w,v_0}(x) \approx \phi_w(x)$. Во время обучения обновляются только веса v, а w остаются неизменными. В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

Как это делать?

Для решения каждой новой задачи между слоями исходной модели будем добавлять новые, так называемые, adapter слои. В процессе обучения обновляться будут только веса этих добавленных слоев.

Adapter tuning для трансформеров

Рис. 1: Слева блок трансформера, справа слой адаптера



Transformer

В трансформере есть два главных типа слоёв: *attention* слой и *feedforward* слой. Слой *adapter* добавляется сразу после каждой пары указанных слоев, но перед слоем нормализации и перед *skip connection* слоем.

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.

Махания руками 3.0

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.

Махания руками 3.0

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.

Удобно?

Да[a].

[a] потому что новые задачи не портят качество на старых задачах в силу неизменения старых весов и количество новых параметров очень мало

Эксперименты

Результаты

На GLUE бенчмарке adapter tuning позволяет достичь результата в диапазоне 0.4% от fine-tuning всех параметров модели BERT, при этом adapter tuning добавляет и обучает всего 3% от общего количества параметров модели.

Parameter-Efficient Transfer Learning for NLP

	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

Prefix-Tuning

Prefix-tuning

Вторая рассматриваемая техника PEFT это Prefix Tuning [2]. Эта техника пытается решить те же проблемы, что и adapter tuning, а именно обучаться на новые задачи не теряя способности решать старые, а также обучение минимального количества весов модели[a], чтобы для новых задач не требовалось хранить полные копии исходной модели.

[a] оказывается у этого есть название — lightweight fine-tunning

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 $(\text{name} : \text{'Starbucks'}, \text{type} : \text{'coffee shop'})$ хотим получить текстовое описание: 'Starbucks serves coffee'.
- Summarization — генерация краткого текстового описания более длинного исходного текста.

Махания руками

Контекст

В языковых моделях для генерации текста используются векторы контекста[a], призванные помогать модели понимать, что ей следует предсказывать дальше.

[a] вспоминаем рекуррентные нейронные сети

Махания руками

Контекст

В языковых моделях для генерации текста используются векторы контекста[a], призванные помогать модели понимать, что ей следует предсказывать дальше.

[a] вспоминаем рекуррентные нейронные сети

Гениальная идея

Можно ли придумать такой контекст в виде префикса, дописываемого перед каждыми входными данными, который будет указывать модели не просто генерируемый результат, а саму задачу, которая модель должна решать?

Махания руками

Контекст

В языковых моделях для генерации текста используются векторы контекста[a], призванные помогать модели понимать, что ей следует предсказывать дальше.

[a] вспоминаем рекуррентные нейронные сети

Гениальная идея

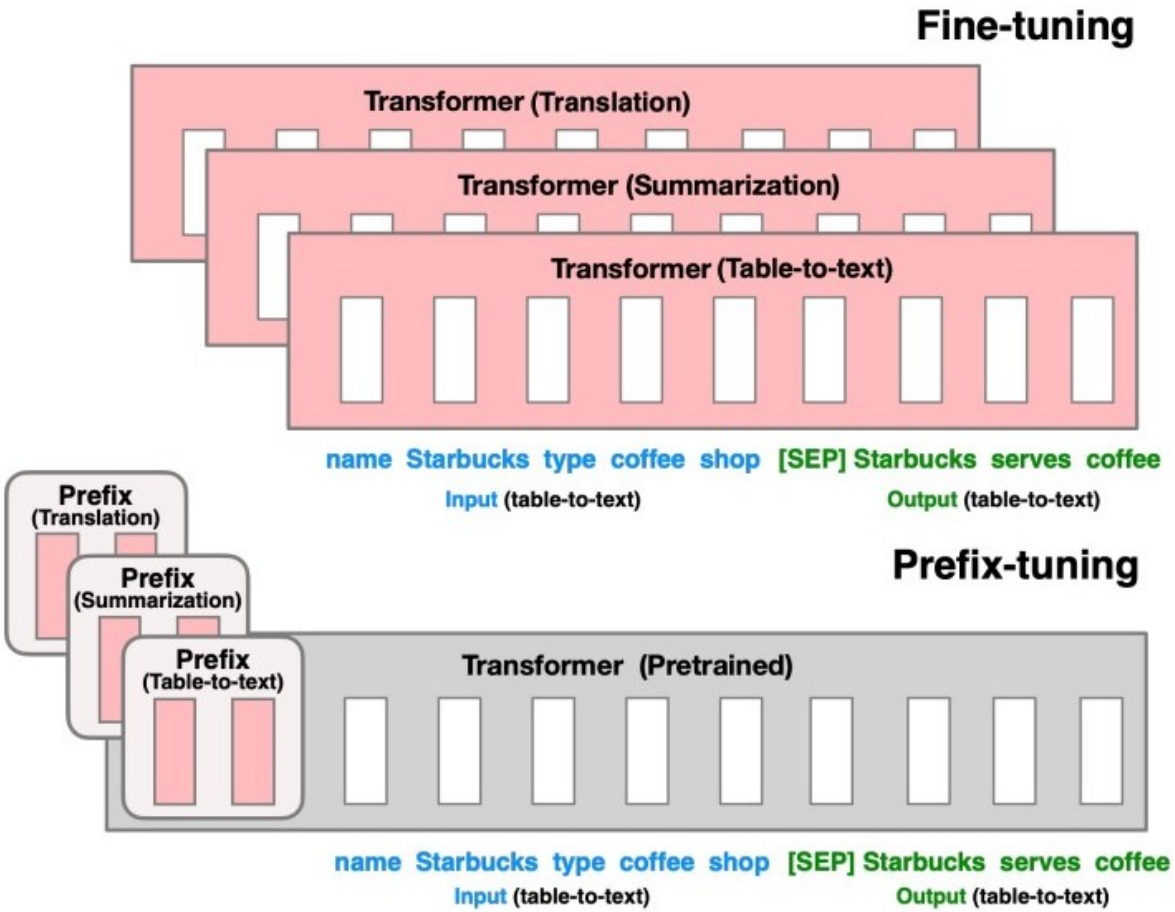
Можно ли придумать такой контекст в виде префикса, дописываемого перед каждыми входными данными, который будет указывать модели не просто генерируемый результат, а саму задачу, которая модель должна решать?

Solution

Да!

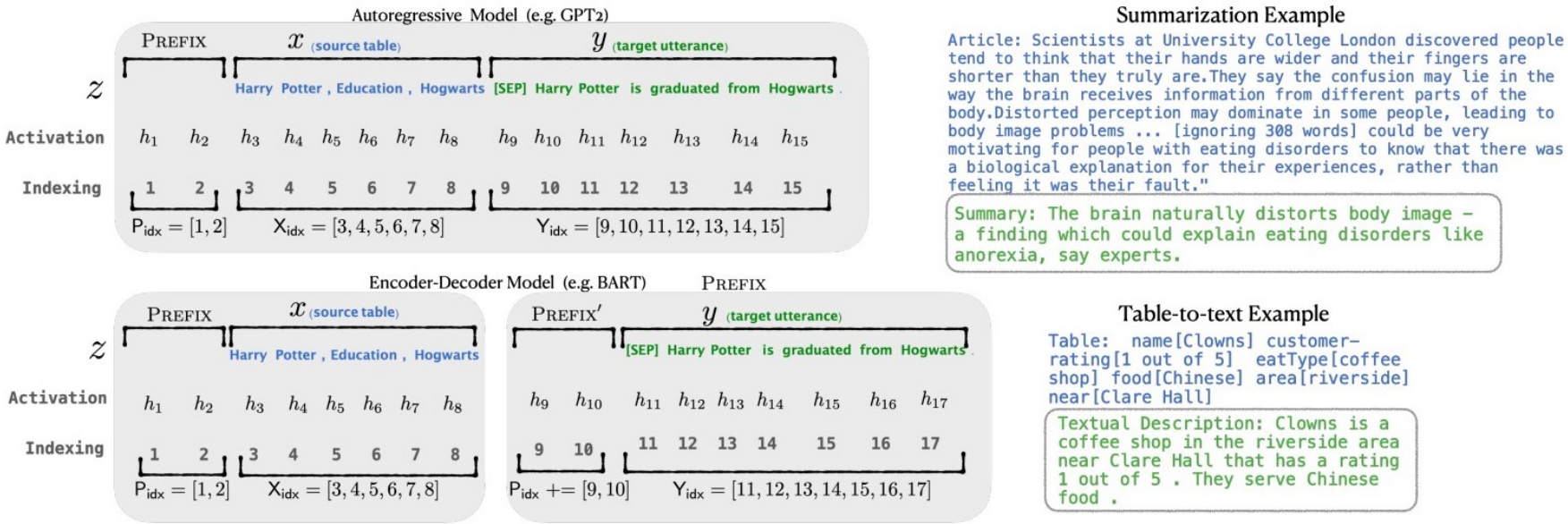
Красивая картинка

Рис. 2: Сравнение количества обучаемых параметров в полном fine-tuning и prefix-tuning



Ещё одна красивая картинка

Рис. 3: Куда пихать перфикс



Формулки

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.

Метод

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.

Эксперименты

Датасеты

Для table-to-text задачи использовались датасеты E2E, WebNLG и DART, а для summarization использовался XSUM.

Архитектуры

Для table-to-text использовались GPT-2_{MEDIUM} и GPT-2_{LARGE}. Для summarization использовался $\text{BART}_{\text{LARGE}}$.

Результаты

Table-to-text

Как видно по таблице 18, prefix-tuning обходит другие бейзлайны[a] добавляя всего лишь 0.1% параметров и имеет качество, сравнимое с полным fine-tuning на всех трех датасетах. Если уменьшить количество параметров для метода adapter до тех же 0.1%, его качество станет сильно хуже prefix-tuning.

[a] Adapter и FT-TOP2

Summarization

Судя по таблице 18, prefix-tuning уже уступает полному fine-tuning в силу более сложного датасета.

Таблица

Рис. 4: Table-to-text

	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	-	-	-	-	-	-	-	-	-

Рис. 5: 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

LoRA

Последняя рассматриваемая техника PEFT это LoRA [3], то есть Low-Rank Adaptation of Large Language Models. Как и ранее, мы пытаемся придумать эффективный способ дообучения исходной модели под новые задачи. Adapter tuning и prefix-tuning уже решили многие проблемы полного fine-tuning, но данные методы[1] имеют ряд недостатков:

- Увеличение inference latency, то есть дополнительные расходы по времени инференса из-за дополнительных слоев модели;
- Методы сложно комбинируются друг с другом;
- Используют много VRAM;
- Плохо распараллеливаются.

[1] а также другие придуманные на текущий момент

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!

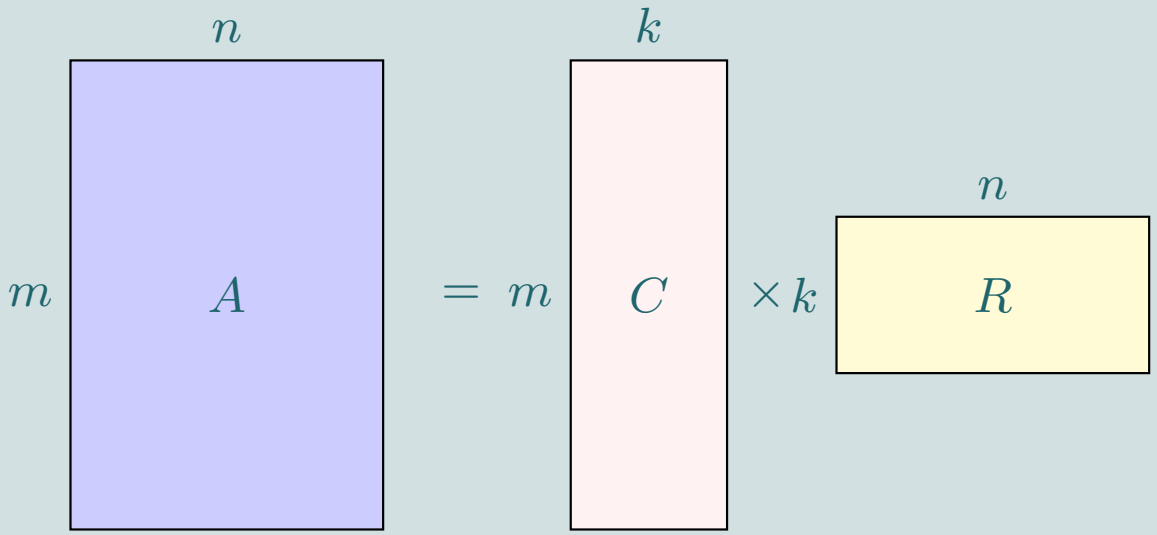
Перепараметризация

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.

Ранг матрицы

Definition

Рангом матрицы $A \in \mathbb{R}^{m \times n}$ будем называть минимальное неотрицательное целое число k такое, что A можно представить в виде произведения $A = CR$ двух матриц[a] $C \in \mathbb{R}^{m \times k}$ и $R \in \mathbb{R}^{k \times n}$.



[a] картинку я рисовал честно в texe

Махания руками

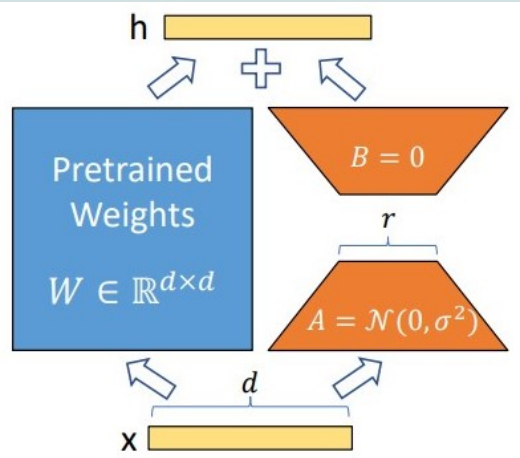
Кукарек[a]

[a] на самом деле не такой уж и кукарек

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:

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

Как это делать?



В каждом attention-слое трансформера есть четыре матрицы (W_q, W_k, W_v, W_o). Зафиксируем какое-то подмножество этих матриц (например, (W_q, W_v)) и для каждого из них добавим в модель два слоя как на картинке слева. В процессе обучения будем обновлять только матрицы A и B и получим искомое.

Преимущества

Пунктики

- 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.

Дополнительно

- Можно подбирать к какому подмножеству матриц attention-слоев применять такие разложения и далее обучать;
- Можно подбирать ранги r этих разложений.

Эксперименты

Сеттинг

LoRA применялся к архитектурам RoBERTa, DeBERTa и GPT-2, а также расширение до GPT-3. В качестве бенчмарков брались GLUE, WikiSQL и SAMSum.

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**

Эксперименты 2.0

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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.85}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{71.8}_{\pm.1}$	$\mathbf{2.53}_{\pm.02}$
GPT-2 L (FT)*	774.03M	68.5	8.78	46.0	69.9	2.45
GPT-2 L ($\text{Adapter}^{\text{L}}$)	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 ($\text{Adapter}^{\text{L}}$)	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	$\mathbf{70.4}_{\pm.1}$	$\mathbf{8.89}_{\pm.02}$	$\mathbf{46.8}_{\pm.2}$	$\mathbf{72.0}_{\pm.2}$	$2.47_{\pm.02}$

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 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	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

Список литературы I

[1] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan и Sylvain Gelly. "Parameter-Efficient Transfer Learning for NLP". В: *CoRR* abs/1902.00751 (2019). arXiv: 1902.00751. URL: http://arxiv.org/abs/1902.00751.

[2] Xiang Lisa Li и Percy Liang. "Prefix-Tuning: Optimizing Continuous Prompts for Generation". В: *CoRR* abs/2101.00190 (2021). arXiv: 2101.00190. URL: https://arxiv.org/abs/2101.00190.

[3] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang и Weizhu Chen. "LoRA: Low-Rank Adaptation of Large Language Models". В: *CoRR* abs/2106.09685 (2021). arXiv: 2106.09685. URL: https://arxiv.org/abs/2106.09685.