Simplex LoRA: Expand important adapters

Давыденко Григорий, **Шалыгин Игорь**

Научный руководитель: Безносиков А. Н.
Научный консультант: Веприков А. С.

Московский физико-технический институт

18 мая 2025 г.

Содержание доклада

1. Введение и мотивация
2. Постановка задачи
3. Основная идея SimplexLoRA
4. Методы изменения рангов матриц
5. Результаты экспериментов
6. Заключение и дальнейшие планы

Введение и мотивация

Low-rank adaptation (LoRA):

- Тюнинг линейных слоев больших моделей через низкоранговые матрицы $A \in \mathbb{R}^{n \times r}$ и $B \in \mathbb{R}^{r \times m}$ $(r \ll m, n)$:

$$W_i \longrightarrow W_i + A_i \cdot B_i$$

- Эффективность за счет снижения числа обучаемых параметров

Проблема:

- В данном алгоритме мы не регулируем ранги адаптеров
- Хотим ответить на вопрос:

  Выходы каких слоев вносят наибольшее отклонение от оптимума и должны быть скорректированы сильнее?

Идея и алгоритм SimplexLoRA

Добавим к каждому адаптеру вес ω.

Ограничения: $\sum_{i=1}^{n} \omega_i = n$ и $\omega_i \geq 0$ (масштабированный симплекс).

За время обучения мы совершаем раз в несколько шагов изменения рангов LoRA в соответствии с их обучаемыми весами:

1. Сначала все ранги равны гиперпараметру $r_i^{(0)} = r^0$, веса - $\omega_i^{(0)} = 1$
2. После шага в оптимизаторе $\omega^{(k)} = proj_{simplex}(\omega^{(k)})$
3. Через фиксированное количество шагов x меняем ранги LoRA согласно новым весам:

$$r_i^{(u)} = ceil(r^0 \cdot \omega_i^{(u \cdot x)})$$

4. Пункты 2 и 3 повторяем несколько раз.
5. Запускаем стандартное обучение LoRA с подобранными рангами.

Расширение рангов

Пусть есть LoRA адаптер с $A_{old} \in \mathbb{C}^{n \times r_{old}}$ и $B_{old} \in \mathbb{C}^{r_{old} \times m}$, который мы хотим расширить до ранга $r > r_{old}$

- **Наивный подход:**
  Дополнение обученных матриц A и B нормальной и нулевой

$$A = [A_{old} \quad N_{n \times (r - r_{old})}] \qquad B = [B_{old} \quad 0_{m \times (r - r_{old})}]$$

- **Через QR-разложение:**
  QR-разложения: $A_{old} = Q_A \cdot R_A$

$$A = [Q_A \quad (I - Q_A \cdot Q_A^*) \cdot N_{n \times (r - r_{old})}] = [Q_A \quad N^{new}_{n \times (r - r_{old})}]$$

$$B = [B_{old} \cdot R_A^* \quad 0_{m \times r_{old}}]$$

В обоих случаях у нас соблюдается равенство $A_{old} \cdot B_{old} = A \cdot B$

Сжатие рангов

Пусть есть LoRA адаптер с $A_{old} \in \mathbb{C}^{n \times r_{old}}$ и $B_{old} \in \mathbb{C}^{r_{old} \times m}$, который мы хотим уменьшить до положительного ранга $r < r_{old}$

Ставим оптимизационную задачу:

$$||A_{old} \cdot B^*_{old} - A \cdot B^*||_F \longrightarrow min_{A \in \mathbb{C}^{n \times r}, B \in \mathbb{C}^{r \times m}}$$

Решение: эффективно обрезать SVD разложение $A_{old} \cdot B^*_{old}$.

$$A_{old} = Q_A \cdot R_A \quad B_{old} = Q_B \cdot R_B \quad U \cdot \Sigma \cdot V^* = SVD(R_A \cdot R_B^*)$$

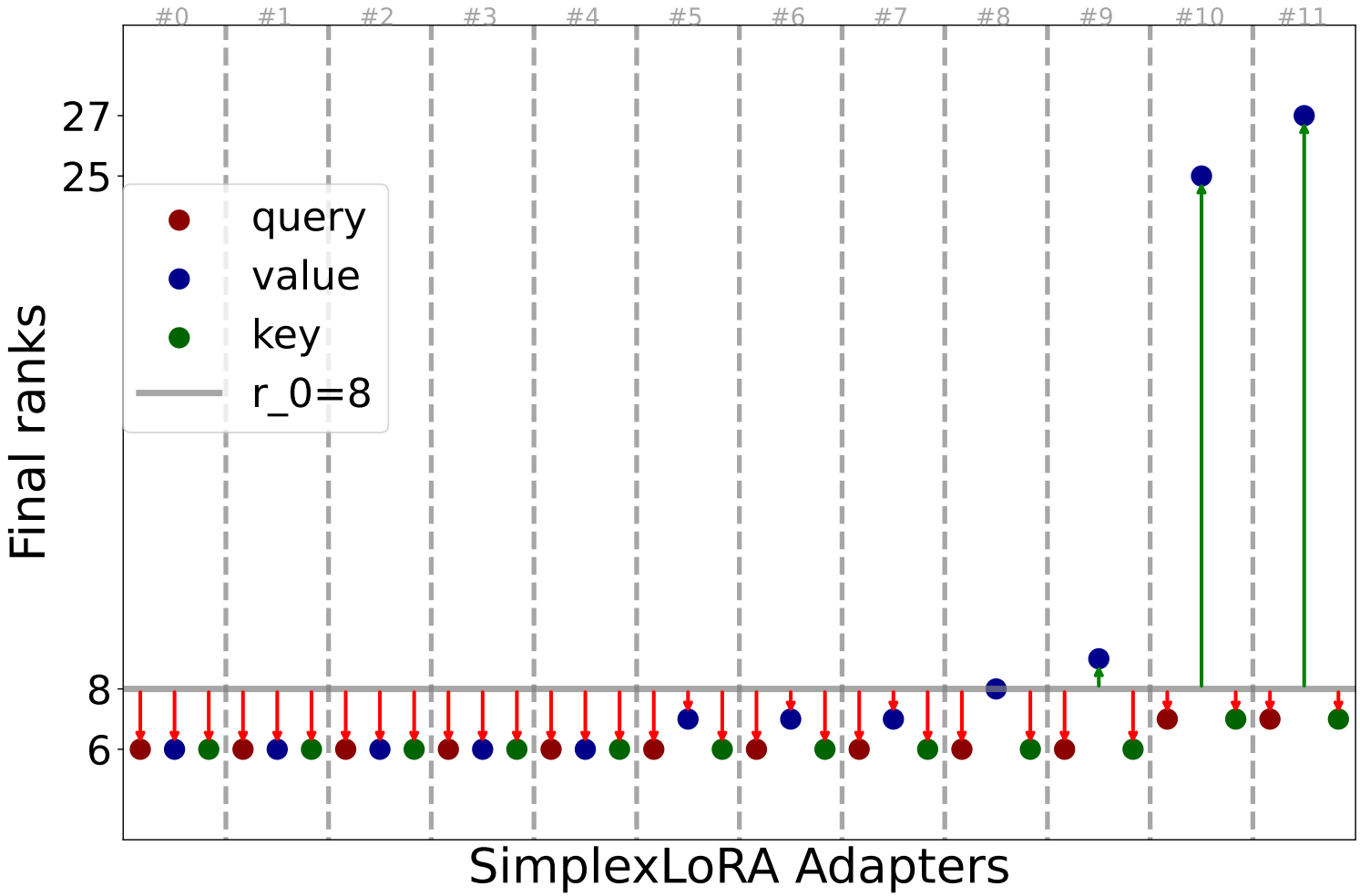
$$\begin{cases} U \in \mathbb{C}^{r_{old} \times r_{old}} \\ V \in \mathbb{C}^{r_{old} \times r_{old}} \\ \Sigma \in \mathbb{C}^{r_{old} \times r_{old}} \end{cases} \longrightarrow \begin{cases} U_r \in \mathbb{C}^{r_{old} \times r} \\ V_r \in \mathbb{C}^{r_{old} \times r} \\ \Sigma_r \in \mathbb{C}^{r \times r} \end{cases}$$

Итоговые матрицы A и B:

$$A = Q_A \cdot U_r \qquad B = \Sigma_r \cdot V_r^* \cdot Q_B^*$$

Результаты экспериментов

Рис.: Итоговое распределение рангов после 3 шагов проекции.



Результаты экспериментов

Таблица: Тюнинг гиперпараметров на примере Roberta(185M) GLUE RTE

Метод	Ранг r_0			
	1	2	4	8
LoRA	0.79422 1e-4	0.78700 8e-5	0.83032 5e-4	0.84477 5e-4
SimplexLoRA	0.81588 1e-5	0.82310 5e-5	0.81588 5e-5	0.81949 1e-4

Обращаем внимание на следующие параметры:

1. learning rate
2. warmup steps
3. количество операций изменений рангов

Во время проведения экспериментов выявлено, что данные параметры имеют наибольшее влияние на целевые метрики.

Заключение и дальнейшие планы

1. Досчитываем эксперименты и метрики, ведем учет в таблицах
2. Дописываем статью на конференцию ACL
3. Готовим pull request в библиотеку PEFT

Источники

- LoRA: Low-Rank Adaptation of Large Language Models
- AdaLoRA: Adaptive Budget Allocation for Parameter-Efficient Fine-Tuning
- Large-scale Multiclass Support Vector Machine Training via Euclidean Projection onto the Simplex