Теоретические основы AdaBERT и реализация кастомной архитектуры

1.1 Идея и ожидания

BERT и его производные модели стали стандартом в обработке естественного языка (NLP), но их масштаб — сотни миллионов параметров — делает такие модели непригодными для задач с ограниченными вычислительными ресурсами. Например, BERT-base содержит около 110 миллионов параметров. Для реального времени это слишком тяжело: медленно, много памяти и очень долго обучать

Решение — сжать модель. Но большинство методов делают это независимо от задачи, сжимая модель один раз и применяя ко всем задачам одинаково. Но по сути существуем множество различных задача и мести все под одну гребенку неправильно. Поэтому было принято решение реализовать adaBert, адаптивный способ сжатия BERT, который подбирает архитектуру под конкретную задачу. Для этого используется дифференцируемый поиск архитектур (Differentiable Neural Architecture Search), и два типа потерь:

- Task-oriented distillation loss — помогает передавать полезную для конкретной задачи информацию от большой teacher-модели к маленькой student-модели.
- Efficiency loss — штрафует за громоздкие архитектуры и помогает искать компромисс между точностью и компактностью.

1.2 Архитектура поиска: SearchCell

`SearchCell` — это отдельная ячейка архитектуры, в которой определяется множество возможных операций:

- `conv1`, `conv3`, `conv5` — свёртки с ядрами разного размера (1, 3, 5)
- `dilconv3` — расширенная свёртка
- maxpool, avgpool — пулинг
- `skip` — пропуск (identity).

Каждая операция применяется к входу x, и результат всех операций складывается с весами α_i:

$$\mathrm{h} = \sum_{i=1}^{n} \alpha_i \cdot \mathrm{op}_i(x)$$

α_i — это веса архитектуры. Внутри каждой `SearchCell` происходит перемножение выходов операций на их веса и суммирование. Таким образом, выход ячейки представляет собой взвешенное среднее по всем операциям.

Также я использую Gumbel-Softmax для более мягкой версии выбора одной операции. По сути, гумбель-софтмакс это способо выбрать одну операцию или слой, но при этом сохранить дифференцируемость, чтобы продолить обучение с помощью градиентов. Грубо говоря, вместо того, чтобы выбирать один вариант мы вычисляем вероятности и складываем полуившиеся резы. Благодаря Gumbel-Softmax мы можем обучать выбор архитектуры одновременно с обучением весов модели, что делает поиск архитектуры встроенным в обучение. Это намного эффективнее, чем перебирать архитектуры вручную или запускать отдельный внешний NAS-процесс

1.3 Кастомная модель AdaBERT

Модель `AdaBERT` состоит из:

- Эмбеддингов — `nn.Embedding`, кодирующих токены в векторы.
- Множественных SearchCell-ячеек — ячейки каждая выбирает оптимальную операцию из набора.
- Классификатора — простой линейный слой.

Общий выход вычисляется как среднее по последнему скрытому состоянию:

$$\text{output} = \text{Linear}\left(\frac{1}{T}\sum_{t=1}^{T} h_t\right)$$

1.4 Функция потерь эффективности (Efficiency Loss)

Наша цель найти не только точную, но и быструю, компактную модель. Для этого вводится функция эффективность-потеря:

$$\mathscr{L}_{\text{eff}} = \frac{\sum_i \theta_i \cdot FLOPs(op_i)}{FLOPs_{\max}} + \frac{\sum_i \theta_i \cdot Params(op_i)}{Params_{\max}} \tag{1}$$

FLOPs и число параметров рассчитываются как ожидание от распределения архитектурных весов

1.5 Distillation loss: передача знаний от учителя

Функция `custom_kd_loss` реализует мягкую передачу знаний:

1. Выбираются соответствия между слоями teacher и student (например, 0-0, 1-3, 2-6).

2. Для каждого слоя считается MSE между скрытыми представлениями.

3. Если размерности различаются, представление ученика проектируется линейно в размерность учителя.

$$\mathscr{L}_{\text{KD}} = \frac{1}{|M|} \sum_{(s,t)\in M} \|h_s^{\text{student}} - h_t^{\text{teacher}}\|^2 \tag{2}$$

1.6 Финальная функция потерь

Модель обучается по объединённой функции потерь:

$$\mathscr{L} = \lambda_1 \cdot \mathscr{L}_{\text{CE}} + \lambda_2 \cdot \mathscr{L}_{\text{KD}} + \lambda_3 \cdot \mathscr{L}_{\text{eff}} \tag{3}$$

Я распределили веса 0.2, 0.8, 1.5. Когда повышал коэффициенты при λ_3 модель переставала обучаться и застревала

1.7 Заморозка и экспорт структуры

После тренировки мы замораживаем архитектур и выбираем те слои и операции, которые получили наибольшие веса. Это даёт статическую, эффективную модель, пригодную для продакшена.

Финальный вариант модели называется `FrozenAdaBERT` — он сохраняет только выбранные операции и обученные веса, отброшены все лишние ветви. Используя Gumbel Softmax, efficiency loss и передачу знаний от учителя, я создал архитектуру, которая умеет адаптироваться и работать эффективно даже на ограниченных ресурсах.

Архитектура была выбрана на всех этапах avgpoll, скорее всего это произошло из-за маленького количетсво эпох при подборе архитектуры. Я использовал 15, в то время как в различных статьях используется от 50до 90 эпох. Если прогнать на таком большом количестве, я думаю подберется более точная архитектура, но мне кажется сильного буста по точности это не даст. Ниже я приведу сравнительный анализ моей полученной модели с помощью adaBERT и обычной классической берт

Таблица 1: Сравнение характеристик моделей

Модель	Параметры (млн)	Вес (МБ)	Время эпохи (мин)	Точность (%)
BERT-base	110	430	55-58	94.12
AdaBERT (моя)	7.8	1 - 1.2	12	89.79