Квантизация языковых моделей с помощью дифференцируемого поиска архитектуры нейронной сети

Д. Н. Давыдова*

Поиск архитектуры нейронной сети – это метод, позволяющий автоматически подобрать наилучшую архитектуру нейронной сети для данной задачи на представленном алгоритму наборе данных. Одной из разновидностей поиска архитектуры нейронной сети является DARTS – differentiable neural architecture search. DARTS позволяет перейти от дискретного пространства поиска к непрерывному и использовать метод градиентного спуска для подбора параметров, отвечающих за архитектуру сети. В статье будет рассмотрена возможность использования метода DARTS для подбора схемы квантизации и разрядности различных компонентов генеративных языковых моделей и описаны результаты проведенных экспериментов. Код для обучения и оценки квантизованной модели доступен по ссылке: https://github.com/daria1d/Darts-QAT.

Ключевые слова: обработка естественного языка, поиск архитектуры нейронной сети, квантизация, DARTS.

1. Введение

Метод поиска архитектуры нейронной сети позволяет автоматизировать процесс подбора архитектуры различных нейросетевых моделей. Подбор оптимальной архитектуры нейронной сети вручную может занимать значительное количество времени и требовать проведения большого количества экспериментов, в то время как поиск архитектуры нейронной сети позволяет существенно ускорить создание наиболее эффективной модели для данной задачи.

Среди методов поиска архитектуры нейронной сети можно выделить эволюционные алгоритмы, методы обучения с подкреплением, дифференцируемый поиск архитектуры нейронной сети, поиск архитектуры с использованием больших языковых моделей.

*Давыдова Дарья Николаевна — аспирант каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: d.dawydowa2017@yandex.ru, ORCID: 0009-0009-4338-9734.

Davydova Daria Nikolaevna — PhD student, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

Опубликовано на условиях лицензии CC-BY 4.0
Published under CC-BY 4.0

Первые подходы поиска архитектуры нейронной сети были основаны на эволюционных алгоритмах. Подбор архитектурных параметров с помощью эволюционных алгоритмов осуществляется за счет модификации набора потенциальных решений (популяции) на основе ряда правил (операторов). На первом этапе эволюционный алгоритм инициализирует набор случайных сетей для создания первичной популяции. Далее каждая архитектура оценивается с помощью функции приспособленности, которая отражает её качество (например, точность классификации). По этой оценке отбираются наиболее подходящие архитектуры, к которым применяются итеративные процедуры мутации, скрещивания и селекции [1] для создания новых архитектур-кандидатов. Эволюционный процесс продолжается до тех пор, пока не будет достигнут критерий завершения, который обычно определяется максимальным количеством допустимых поколений.

Генетические алгоритмы требуют большого количества вычислительных ресурсов и не способны использовать информацию о градиенте, в то время как RL-алгоритмы более эффективны и могут использовать информацию с прошлых шагов для более точного поиска архитектуры. В основе поиска архитектуры нейронной сети с использованием RL [2] лежит наблюдение, что широкий класс архитектур нейронных сетей можно описать с помощью строк переменной длины. Ключевая идея метода – обучить модель-контроллер – агента, который пошагово генерирует архитектуру нейронной сети, выбирая компоненты из заданного пространства. Затем производительность модели, построенной по сгенерированному описанию, оценивается на валидационном наборе, и полученная метрика (например, точность) используется в качестве награды для обучения контроллера. Таким образом, генерируя все больше сетей, контроллер обучается строить более производительные модели.

С появлением больших языковых моделей (LLM) популярными стали методы поиска архитектуры нейронной сети, в которых для подбора различных компонентов архитектуры моделей используются LLM. В некоторых статьях [3] исследуются способности LLM в написании кода: LLMatic формирует описание архитектуры в виде кода или псевдокода с помощью LLM, используя заранее заданные шаблоны (промпты). Затем полученные архитектуры оцениваются по качеству и разнообразию с использованием принципов Quality-Diversity оптимизации: отбираются не только точные, но и структурно различные модели. Каждая архитектура оценивается на точность и производительность. Отобранные кандидаты далее служат основой для новых запросов к LLM, что позволяет итеративно улучшать архитектуру.

Другой метод LLaMA-NAS [4] находит оптимальные по Парето варианты архитектур на базе LLaMA2-7B. В основе метода лежит концепция one-shot суперсети – большой, переопределённой модели, которая содержит множество возможных подархитектур, например, с меньшим числом трансформерных слоёв или меньшими размерами скрытых слоёв. Во время обучения из этой суперсети совместно выбираются и дообучаются несколько подархитектур, что позволяет использовать общие параметры и эффективно оценивать модели. Поиск в пространстве потенциальных подархитектур в LLaMA-NAS осуществляется с помощью генетического алгоритма. В результате получаются оптимальные по Парето варианты архитектур – набор моделей, оптимальных по размеру, временным затратам и точности.

Дифференцируемый поиск архитектуры нейронной сети (DARTS) [5] позволяет достигать качества, сравнимого с SOTA-моделями, затрачивая при этом меньше вычислительных ресурсов. В то время как большинство методов поиска архитектуры нейронной сети предполагают поиск по дискретному набору архитектур-кандидатов, DARTS сокращает пространство поиска, делая его непрерывным. Это позволяет обучать параметры архитектуры с использованием градиентного спуска. Такой подход помогает существенно сократить вычислительные затраты при сохранении качества, сравнимого с лучшими подобранными вручную архитектурами.

Далее будут проведены эксперименты, адаптирующие метод DARTS для подбора наиболее оптимальной схемы квантизации и степени сжатия для каждого компонента модели и описаны полученные результаты.

2. Алгоритм дифференцируемого поиска архитектуры нейронной сети

Метод DARTS [5] был предложен в статье DARTS: Differentiable Architecture Search, опубликованной в 2018 году. Целью работы было сделать процесс автоматического поиска архитектур нейронных сетей более эффективным и доступным, заменив дорогие дискретные поисковые методы, требующие сотен GPU, на дифференцируемый, градиентно-оптимизируемый подход.

Алгоритм предполагает поиск ячейки архитектуры сети – повторяющегося базового элемента, служащего шаблоном для построения всей модели. Найденную ячейку можно укладывать последовательно для формирования итоговой архитектуры или сочетать с другими компонентами нейронной сети.

Каждая ячейка представляет собой ориентированный граф без циклов, содержащий N промежуточных вершин, две входных и одну выходную. Каждая вершина графа соответствует скрытому представлению $x(i)$ (например, карте признаков в случае сверточной нейронной сети), а каждое ребро – операции $o^{(i,j)}$, которая преобразует выход узла $x^{(i)}$ и передаёт результат в узел $x^{(j)}$. Выходные данные ячейки получаются путем применения операции сокращения (например, конкатенации) ко всем промежуточным узлам.

Каждая промежуточная вершина вычисляется с учетом всех своих предков:

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$$

Специальная нулевая операция указывает на отсутствие ребра между двумя вершинами. Задача подбора ячейки таким образом сводится к подбору операций на ребрах.

Для перехода от дискретного к непрерывному пространству поиска выбор операции определяется как функция Softmax над множеством всех операций-кандидатов:

$$\overline{o}^{(i,j)}(x) = \sum_{o \in O} \frac{exp(\alpha_o^{(i,j)})}{\sum_{o' \in O} exp(\alpha_{o'}^{(i,j)})} o(x),$$

где веса смешанной операции для пары вершин (i, j) определяются вектором $\alpha^{(i,j)}$ размерности $|O|$.

Задача поиска архитектуры тогда сводится к обучению множества непрерывных переменных $\alpha = \{\alpha^{(i,j)}\}$. После процедуры поиска дискретную архитектуру можно получить заменой каждой смешанной операции $o'^{(i,j)}$ на наиболее вероятную операцию, то есть, $o^{(i,j)} = argmax_{o \in O} \alpha_o^{(i,j)}$.

В процессе поиска архитектуры DARTS решает задачу двухуровневой оптимизации, где параметры весов w и архитектуры α обучаются попеременно на разных подмножествах данных. На внутреннем уровне минимизируется ошибка на обучающей выборке:

$$w^*(\alpha) = \arg\min_w \mathcal{L}_{train}(w, \alpha).$$

На внешнем уровне оптимизируются архитектурные параметры по ошибке на валидации:

$$\min_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha).$$

На каждой итерации алгоритм обновляет w градиентным спуском по $\mathcal{L}_{train}$, фиксируя α, затем обновляет α по $\mathcal{L}_{val}$, фиксируя w. После завершения поиска полученная архитектура дискретизируется (выбираются операции с наибольшими коэффициентами α), и сеть обучается заново, оптимизируя только веса w.

$$\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$w^*(\alpha) = argmin_w \mathcal{L}_{train}(w, \alpha)$$

.

3. DARTS для квантизации языковых моделей

В [6] было проведено исследование по применению метода DARTS для подбора оптимальной степени сжатия различных компонентов модели в сверточных нейронных сетях. Архитектура ячейки сети (количество слоев, размерность фильтра свертки для каждого слоя) является фиксированной для всех вариантов квантизации. Слои ячейки связаны несколькими параллельными ребрами, представляющими операцию свертки, параметры и активации которой приводятся к разным разрядностям с помощью DARTS. Как и в оригинальном DARTS, для обновления параметров используется двухуровневая оптимизация, а в функцию потерь для архитектурных параметров включен также штраф за выбор высокой разрядности в двух вариантах. В первом случае авторы делают акцент на сжатии модели, а во втором – на ускорение и оптимизацию энергопотребления:

$$\mathcal{L} = \mathrm{CE} + C \cdot \sum_{e_{ij} \in E_k} m_{ij} \, \#\mathrm{PARAM}(e_{ij}) \, \mathrm{bit}_w(e_{ij}),$$

$$\mathcal{L} = \mathrm{CE} + C \cdot \sum_{e_{ij} \in E_k} m_{ij} \, \#\mathrm{FLOP}(e_{ij}) \, \mathrm{bit}_w(e_{ij}) \, \mathrm{bit}_a(e_{ij}).$$

где CE – кросс-энтропия, e_{ij} – ребро в графе ячейки, m_{ij} – маска выбора ребра, $m_{ij} \in [0, 1]$, C – коэффициент, определяющий вклад штрафа за вычислительную стоимость в функцию потерь, E_k – множество ребер-кандидатов рассматриваемой ячейки.

Использование DARTS для моделей ResNet [7], MobileNet [8], VGG-small [9] на задаче классификации изображений позволило превзойти результаты SOTA-методов квантизации и существенно сжать модель.

Авторы исследования [10] также используют DARTS для подбора схем квантизации для сверточных нейронных сетей MobileNet [8] и ResNet

[7]. Метод сохраняет идею двухуровневой оптимизации из оригинального DARTS и позволяет избегать использования прокси-задачи на больших моделях. В исследовании параметры квантизации (тип операции и битность) и параметры модели оптимизируются совместно по одной функции потерь, в которую включен штраф за высокую разрядность:

$$L[F] = CE + \eta \sum_{f \in F} \mathbb{E}[b_f]\, \mathbb{E}[b_a]\, \frac{|f|\, w\, h}{s^2},$$

$$\mathbb{E}[b_f] = \sum_i \pi_i\, \alpha_i^{b_f}, \qquad \mathbb{E}[b_a] = \sum_j \pi_j\, \beta_j^{b_a},$$

где CE – кросс-энтропия, f – операция (сверточный или линейный фильтр), w, h – ширина и высота выхода фильтра, s – шаг операции f, b_f – битность весов, b_a – битность активаций, $\mathbb{E}[b_f]$ – ожидаемая битность весов фильтра, $\mathbb{E}[b_a]$ – ожидаемая битность активаций, π_i – вероятность выбора i-й операции, η – коэффициент, определяющий вклад штрафа за вычислительную стоимость, $|f|$ – число параметров операции f, i и j – индексы кандидатов разрядности весов и активаций соответственно.

В результате удалось сжать исходные модели в несколько раз с точностью, близкой к исходным 32-битным моделям.

В статье [11] предложен метод ∇QDARTS, в котором с помощью дифференциируемого поиска подбирается не только архитектура, но и разрядность для различных компонентов сети. Исследования проводились на ResNet, MobileNet [8] и EfficientNet [12] для задач классификации изображений. В методе ∇QDARTS процесс обучения происходит на двух уровнях: на внешнем уровне оптимизируются архитектурные параметры, а на внутреннем – параметры разрядности и параметры самой сети. Для поощрения модели к снижению битности на внешнем уровне вводится штраф за высокую разрядность, что позволяет добиться баланса между точностью и эффективностью модели.

$$\mathcal{L}[F] = R_E(F) + \nu \sum_{i=1}^{N_{\mathrm{OP}}} \mathrm{OP}_i \cdot \mathrm{BitA}_i \cdot \mathrm{BitW}_i$$

$$\mathrm{BitA}_i = \gamma_i^p a_i, \quad \mathrm{BitW}_i = \gamma_i^q w_i,$$

где $R_E(F)$ – эмпирическая ошибка модели F, $\nu \geqslant 0$ – обучаемый коэффициент при штрафе за битность, OP_i – вычислительная стоимость i-й операции, N_{OP} – число элементарных вычислений слоя, a_i, w_i – активации и веса, γ_i^p, γ_i^q – параметры, отвечающие за отбор битности.

На задачах классификации изображений метод обеспечил до 2–3% прироста точности при снижении вычислительных затрат и размера модели на 30–50% по сравнению с методом, представленным в [6].

В исследовании [13] авторы использовали метод DARTS для поиска оптимальной схемы квантизации трансформерной модели BERT [14]. В [13] архитектурные параметры α отвечают за подбор разрядности весов и активаций линейных компонентов модели. Параметры α подбираются для различных групп весов и активаций (для отдельной матрицы, для всего механизма внимания или для целых блоков модели). Авторы используют двухуровневую оптимизацию – параметры модели оптимизируются с помощью кросс-энтропии, а параметры подбора разрядности оптимизируются с помощью функции потерь, сочетающей кросс-энтропию и штраф за высокую битность:

$$\mathcal{L} = CE + \lambda\, L_{\text{size}},$$

где λ – коэффициент штрафа за размер модели, L_{size} – штраф за размер модели.

Штраф за размер модели должен обеспечить низкую разрядность модели, сохранив при этом достаточно высокую точность. Для этого вводится желаемый размер модели, и сеть штрафуется как за слишком высокую разрядность для группы параметров, так и за слишком низкую:

$$L_{\text{size}} = \begin{cases} 0, & C_O \in [(1-\epsilon)V,\, (1+\epsilon)V] \\ \log \mathbb{E}[CO_{i,j}], & C_O > (1+\epsilon)V \\ -\log \mathbb{E}[CO_{i,j}], & C_O < (1-\epsilon)V, \end{cases}$$

где C_O – фактический размер модели с распределением разрядностей O, V – желаемый размер модели, $\mathbb{E}[CO_{i,j}]$ – ожидаемый вклад группы параметров в размер модели, i – индекс слоя, j – индекс группы параметров внутри слоя.

Эксперименты на модели BERT с задачами из GLUE [15] показали, что предложенный метод сохраняет до 99% точности исходной модели при заметном сокращении вычислительной сложности и объёма памяти, демонстрируя, что принципы DARTS могут быть эффективно адаптированы для задач смешанной квантизации и сжатия трансформеров.

4. Эксперимент по применению DARTS для квантизации генеративной трансформерной модели

В статье будет описан эксперимент по подбору схемы квантизации и разрядности для весов и активаций различных компонентов генеративной трансформерной модели с помощью метода DARTS и представлены полученные результаты.

В отличие от прошлых исследований [13], метод DARTS будет адаптирован для генеративной трансформерной модели, состоящей только из блоков декодера. Метод подбирает битность для каждого линейного слоя модели из блоков декодера и учитывает ошибку квантизации для уменьшения потерь в качестве работы модели при снижении разрядности.

4.1. Подбираемые схемы квантизации

Для эксперимента в качестве кандидатов были выбраны следующие схемы квантизации: PACT – квантизация активаций с обучаемым порогом отсечения [16], LSQ+ – квантизация с обучаемым шагом [17] и равномерная симметричная квантизация [18].

Равномерная симметричная квантизация [18] – это представление непрерывных значений весов и активаций дискретными значениями с фиксированным шагом. Диапазон значений квантизации при этом симметричен относительно 0. Таким образом, нулевая точка квантизации всегда находится в 0, а все уровни квантизации имеют один и тот же шаг.

Квантизованное целочисленное значение веса или активации вычисляется по формуле:

$$\bar{x} = \text{clamp}\Big(\text{round}\big(x/s\big), n, p\Big),$$

где s – шаг квантизации, n и p – границы диапазона квантизации. Восстановленное деквантизованное значение вычисляется по формуле:

$$\hat{x} = \bar{x} \times s$$

Для обратного прохода по сети используется straight-through-estimator (STE) [19].

$$\frac{\partial \hat{x}}{\partial x} = s\frac{\partial \bar{x}}{\partial x} \simeq \begin{cases} 1, & ns < x < ps, \\ 0, & \text{иначе.} \end{cases}$$

Квантизация методом LSQ+ [17] – усовершенствованная версия метода LSQ с обучаемым шагом квантизации и пересчетом градиента с учётом

диапазона квантизации, что позволяет лучше адаптировать метод для квантизации к низким разрядностям.

Для квантизации активаций предлагается асимметричная схема, в которой обучается не только шаг квантизации, но и сдвиг, что позволяет корректно обрабатывать смещённое относительно 0 распределение активаций.

$$\bar{x} = round\Big(\text{clamp}\Big(\frac{x-\beta}{s},\, n,\, p\Big)\Big)$$

$$\hat{x} = \bar{x} \times s + \beta,$$

где s – шаг квантизации, β – сдвиг квантизации, n, p – границы диапазона квантизации, $\hat{x}$ – восстановленные (деквантизованные) активации, $\bar{x}$ – квантованные активации.

Обновление градиента s вычисляется по следующей формуле:

$$\frac{\partial \hat{x}}{\partial s} = \frac{\partial \bar{x}}{\partial s} s + \bar{x} \simeq \begin{cases} -\frac{x-\beta}{s} + round\Big(\frac{x-\beta}{s}\Big), & \text{если } n < \frac{x-\beta}{s} < p \\ n \text{ или } p, & \text{иначе} \end{cases}$$

Обновление градиента β вычисляется по формуле:

$$\frac{\partial \hat{x}}{\partial \beta} = \frac{\partial \bar{x}}{\partial \beta}\, s + 1 \simeq \begin{cases} 0, & \text{если } n < \dfrac{x-\beta}{s} < p \\ 1, & \text{иначе} \end{cases}$$

В обоих случаях для вычисления $\frac{\partial \bar{x}}{\partial \beta}$ и $\frac{\partial \bar{x}}{\partial s}$ используется straight-through-estimator (STE) [19].

Для весов используется симметричная квантизация со знаком, поскольку на практике распределение весов слоя обычно симметрично относительно нуля. Асимметричная квантизация активаций не вносит дополнительной вычислительной стоимости на этапе инференса, так как сдвиг квантизации может быть предварительно вычислен и включён в значение смещения слоя при запуске.

$$\hat{w}\hat{x} = (\bar{w} \times s_w)\,(\bar{x} \times s_x + \beta) = \bar{w}\bar{x}\, s_w s_x + \underbrace{\beta s_w \bar{w}}_{\text{смещение слоя}},$$

где $\hat{w}$ – восстановленные (деквантизованные) веса, $\bar{w}$ – квантизованные веса.

Также в качестве кандидата схем квантизации рассматривается метод квантизации активаций PACT (Parameterized Clipping Activation

for Quantized Neural Networks) [16]. Проблема, которую решает PACT – потеря точности при квантизации активаций в связи с тем, что при квантизации выходов таких функций, как ReLU, большинство значений близки к нулю, и появляется несколько очень больших значений-выбросов. В статье используется асимметричный вариант PACT с обучаемыми нижней и верхней границами. При квантизации с использованием PACT вводятся обучаемые параметры α_l, α_h, задающие нижнюю и верхнюю границу активации:

$$y = \mathrm{PACT}(x) = \mathrm{clip}(x, \alpha_l, \alpha_h) = \begin{cases} \alpha_l, & \text{если } x < \alpha_l, \\ x, & \text{если } \alpha_l \leqslant x < \alpha_h, \\ \alpha_h, & \text{если } x \geqslant \alpha_h. \end{cases}$$

После обрезки значения равномерно квантуются к k бит в диапазоне $[\alpha_l, \alpha_h]$:

$$y_q = \mathrm{round}\Big(\frac{y - \alpha_l}{s}\Big) \cdot s + \alpha_l, \quad \text{где } s = \frac{\alpha_h - \alpha_l}{2^k - 1}.$$

В процессе обратного распространения ошибки градиенты $\frac{\partial y_q}{\partial \alpha_l}$, $\frac{\partial y_q}{\partial \alpha_h}$ вычисляются с использованием суррогатной производной: градиент распространяется через операцию отсечения: $y = \mathrm{clip}(x, \alpha_l, \alpha_h)$, а остальные зависимости квантизованного значения y_q от границ α_l и α_h в обратном проходе не учитываются.

$$\frac{\partial y_q}{\partial \alpha_l} \simeq \frac{\partial y_q}{\partial y}\frac{\partial y}{\partial \alpha_l} = \begin{cases} 1, & x < \alpha_l \\ 0, & x \geqslant \alpha_l \end{cases}$$

$$\frac{\partial y_q}{\partial \alpha_h} \simeq \frac{\partial y_q}{\partial y}\frac{\partial y}{\partial \alpha_h} = \begin{cases} 0, & x < \alpha_h \\ 1, & x \geqslant \alpha_h \end{cases}$$

4.2. Смешанная операция линейного слоя

При поиске схемы квантизации с помощью DARTS подбираются разрядности для всех линейных слоев модели, за исключением слоя эмбеддингов и выходного слоя логитов, которые наиболее чувствительны к понижению разрядности.

Для каждого линейного слоя ℓ задаётся набор квантизаторов-кандидатов для весов $\{Q_i^w\}_{i=1}^{N_w}$ и активаций $\{Q_j^a\}_{j=1}^{N_a}$ с битностями b_i^w

и b_j^a соответственно. Подбор квантизатора и разрядности для весов и активаций осуществляется с помощью параметров $\boldsymbol{\alpha}^w, \boldsymbol{\alpha}^a$, обучаемых градиентным спуском.

Для перехода к непрерывному пространству поиска параметров, отвечающих за подбор схемы квантизации и битности, выбран Softmax с температурой, позволяющий во время обучения исследовать различные сочетания квантизаторов и разрядностей, а затем перейти к более жесткому выбору битности для каждого слоя.

$$p_i(T) = \frac{\exp\left(\frac{\alpha_i}{T}\right)}{\sum\limits_{j=1}^{N} \exp\left(\frac{\alpha_j}{T}\right)},$$

где α_i – параметры квантизаторов, T – температура.

Активации и веса квантизуются выбранными кандидатами-квантизаторами с заданными битностями по формулам:

$$\tilde{x} = \sum_{j=1}^{N_a} p_j^a \cdot Q_j^a(x)$$

$$\tilde{W} = \sum_{i=1}^{N_w} p_i^w \cdot Q_i^w(W)$$

Таким образом, полная смешанная операция на линейных слоях модели задается следующей формулой:

$$Y = \left(\sum_{i=1}^{N_a} p_i^a \, Q_i^a(X)\right) \cdot \left(\sum_{j=1}^{N_w} p_j^w \, Q_j^w(W)\right)^{\mathsf{T}} + b$$

4.3. Оптимизация и функция потерь

Как и в классическом DARTS, оптимизация параметров модели и параметров квантизаторов проводится на двух уровнях. На внутреннем уровне оптимизируются параметры сети с помощью кросс-энтропии, на внешнем – параметры квантизаторов.

Цель поиска схемы квантизации и разрядности с помощью DARTS – уменьшить размер и вычислительные затраты модели, сохранив при этом ее качество. Поэтому функция потерь внешнего уровня состоит из кросс-энтропии и функции, отвечающей за вычислительные затраты, которые необходимо минимизировать.

Для определения вычислительных затрат необходимо оценить среднюю разрядность слоя и количество операций, выполняемых в данном слое. Для подсчета средней разрядности вычисляется ожидаемая битность весов и активаций при смешанной квантизации на данном линейном слое:

$$\mathbb{E}[b_w] = \sum_{i=1}^{N_w} p_i^w b_i^w, \qquad \mathbb{E}[b_a] = \sum_{j=1}^{N_a} p_j^a b_j^a$$

Количество операций подсчитывается как количество MAC (Multiply–Accumulate) – базовых вычислительных операций, состоящих из двух действий: умножения двух чисел и последующего прибавления результата к накопленной сумме.

$$a = a + (b \cdot c)$$

Тогда количество операций для линейного слоя внутри многослойного персептрона будет вычисляться по формуле:

$$MACs_L = d_{in} \cdot d_{out} \cdot L,$$

где d_{in}, d_{out} – входная и выходная размерность слоя соответственно, L – средняя длина последовательности в корпусе.

Итоговая вычислительная стоимость линейного слоя может быть определена формулой:

$$Cost_L = MACs_L \cdot \mathbb{E}[b_w] \cdot \mathbb{E}[b_a]$$

Стоимость вычислений линейных преобразований для матриц Q, K, V и выходной проекции в механизме внимания можно вычислить по формуле:

$$MACs_{Att} = L \cdot d_{model}^2$$

Стоимость вычислений для операций QK^T и $Softmax(QK^T)V$ определяется следующим образом:

$$MACs_{Att} = L^2 \cdot d_{model}$$

Тогда итоговая вычислительная стоимость слоя внимания может быть определена формулой:

$$Cost_{Att} = L \cdot d_{model}^2 \cdot \Big(\mathbb{E}[b_{w_Q}] \cdot \mathbb{E}[b_{a_Q}] + \mathbb{E}[b_{w_K}] \cdot \mathbb{E}[b_{a_K}] + \mathbb{E}[b_{w_V}] \cdot \mathbb{E}[b_{a_V}] + \\ + \mathbb{E}[b_{w_{out}}] \cdot \mathbb{E}[b_{a_{out}}]\Big) + 2L^2 \cdot d_{model}$$

Суммарная вычислительная стоимость по всем слоям определяется следующим образом:

$$\mathcal{L}_{cost} = \sum_{\ell=1}^{N} \mathrm{Cost}_{L_\ell} + \sum_{\ell=1}^{N} \mathrm{Cost}_{Att_\ell},$$

где N - число трансформерных блоков модели.

Дополнительно вводится штраф за ошибку квантизации, предотвращающий слишком резкий переход модели к низким битностям. Он определяется среднеквадратической ошибкой между исходными весами и активациями и их квантизованными версиями, домноженными на соответствующие параметры квантизации:

$$\mathcal{L}_{\mathrm{err}} = \sum_{\ell=1}^{N_L} \left(\sum_{i=1}^{N_w} p_{w,i}^{(\ell)} \frac{1}{|W_\ell|} \left\| W_\ell - Q_i(W_\ell) \right\|^2 + \sum_{j=1}^{N_a} p_{a,j}^{(\ell)} \frac{1}{|X_\ell|} \left\| X_\ell - Q_j(X_\ell) \right\|^2 \right).$$

Итоговая функция потерь, учитывающая точность модели и ее вычислительную сложность, задается формулой:

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda_{cost} \cdot \mathcal{L}_{cost} + \lambda_{err} \cdot \mathcal{L}_{\mathrm{err}}$$

где $\mathcal{L}_{CE}$ кросс-энтропия для задачи языкового моделирования, а $\lambda_{cost}, \lambda_{err}$ – коэффициенты, балансирующие точность модели и ее вычислительную стоимость.

4.4. Результаты эксперимента

Эксперимент проводился на модели GPT-Neo-125m [20] – трансформерной декодерной модели, содержащей стандартный механизм внимания и многослойный персептрон, состоящий из 2 линейных слоев. Модель была предобучена на задачу генерации текста на корпусе The Pile [21], содержащем около 800 ГБ текста, включая научные статьи, код, книги, энциклопедии, форумы.

Квантизация проводилась на входных активациях и весах матриц Q, K, V и выходной проекции механизма внимания, а также входного и выходного полносвязного линейного слоя многослойного персептрона. Были выбраны следующие квантизаторы-кандидаты и разрядности для весов: квантизатор LSQ+ и симметричный равномерный квантизатор с разрядностями 32, 16, 8 и 4 бит, и LSQ+, PACT и симметричный равномерный квантизатор с разрядностями 32, 16, 8, и 6 бит для активаций.

Размер батча был задан равным 16, длина последовательности 512, использовался оптимизатор AdamW с шагом обучения $2 \cdot 10^{-5}$ для оптимизации параметров модели, AdamW с шагом обучения 10^{-4} для архитектурных параметров. На каждой итерации сначала обновлялись параметры модели по обучающему батчу, затем архитектурные параметры по отдельному валидационному батчу. В эксперименте использовались значения $\lambda_{cost} = 0.02$, $\lambda_{err} = 0.05$. В ходе обучения температура Softmax линейно уменьшалась от 1.5 до 1 по шагам оптимизации. Для токенизации текстов использовался оригинальный токенизатор модели GPT-Neo-125m. Модель обучалась в течение 1 эпохи на подмножестве корпуса The Pile [21], содержащем 20000 текстов. Для внутреннего уровня оптимизации использовалось 60% выборки, для внешнего - 40%. Полное распределение разрядностей и квантизаторов по слоям, полученное в результате поиска DARTS, приведено в приложении.

Качество исходной и квантизованной модели оценивалось с помощью перплексии – обратной вероятности последовательности токенов, нормализованной по ее длине:

$$PPL(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, ..., w_N)}},$$

где W – упорядоченная последовательность токенов тестовой коллекции, N – количество токенов в этой последовательности.

Перплексия замерялась на следующих корпусах:

- wikitext-2 [22], содержащем англоязычные статьи из Википедии размером около 2 миллионов токенов, для оценки использовалась подвыборка test;
- wikitext-103 [22], содержащем статьи из Википедии размером 103 миллиона токенов, для оценки использовалась подвыборка validation;
- FineWeb-EDU [23], содержащем извлечённые из Common Crawl и отфильтрованные по качеству тексты образовательной направленности размером 1.3 триллионов токенов, для оценки использовалась подвыборка train;
- OpenWebText [24] – аналоге корпуса WebText [25], на котором обучалась модель GPT-2 [25], размером 9 миллиардов токенов, для оценки использовалась подвыборка train;
- PG-19 [26], содержащем длинные художественные тексты из Project Gutenberg [27] размером около 2 миллиардов токенов, для оценки использовалась подвыборка validation.

Далее метод поиска схемы квантизации с помощью DARTS сравнивается с другими методами квантизации. Первые 3 из них предполагают квантизацию в ходе обучения (QAT), оставшиеся 2 – квантизацию после обучения (PTQ).

- Квантизация весов по равномерной симметричной схеме к 8 бит;
- Квантизация весов по равномерной симметричной схеме, а активаций по асимметричной схеме к 8 бит по подходу LLM-QAT [28];
- Квантизация весов методом nf4 [29], используемым в QLoRA, адаптированным для QAT, к 4 бит;
- Квантизация весов методом LLM.int8(), введенным в [30], к 8 бит;
- Квантизация весов методом nf4 [29] к 4 бит.

Для оценки степени сжатия модели рассчитывается средняя битность параметров модели с учетом как квантизуемых параметров, так и параметров, сохраняемых в полной разрядности:

$$\text{AvgBits} = \frac{32N_{\text{fp}} + \sum_{i=1}^{N_q} N_i b_i}{N_{\text{fp}} + \sum_{i=1}^{N_q} N_i},$$

где N_{fp} - количество элементов матриц весов, оставшихся в разрядности 32 бит, N_q - количество квантизуемых матриц весов, N_i - число элементов в i-й квантизуемой матрице, b_i - ее разрядность.

Смещения линейных слоев и параметры нормализации при вычислении AvgBits не учитываются, разделяемая матрица весов эмбеддингов и выходного слоя учитывается при подсчете один раз.

Оценка вычислительной сложности модели определяется следующим образом:

$$BitOps = \sum_{i=1}^{n} MACs_i * b_{w_i} * b_{a_i} + 32^2 \sum_{j=1}^{m} MACs_j,$$

где n – количество квантизуемых линейных слоев, b_{w_i} – битность весов i-го слоя, b_{a_i} – битность активаций i-го слоя, m – количество слоев, оставляемых в полной разрядности.

При оценке вычислительной сложности учитывались линейные проекции Q, K, V, O в механизме внимания, два линейных слоя в многослойном персептроне, матричные умножения $QK^\top$ и AV, где A - матрица весов внимания, а также выходной линейный слой.

После проведения экспериментов получены следующие значения перплексии исходной модели GPT-Neo-125m и квантизованных версий на корпусах:

Таблица 1: Результаты экспериментов на модели GPT-Neo-125m

	Wikitext-2	Wikitext-103	FineWeb-EDU	OpenWebText	PG-19	AvgBit	BitOps (G)
GPT-Neo-125m	24.11	25.50	23.28	21.75	22.31	32	68477.21
QAT w8-a32	54.86	50.75	24.81	22.14	23.43	15.72	35079.55
QAT w8-a8	63.52	58.82	28.37	25.32	27.99	15.72	26730.13
QAT w4-a32	129.17	114.55	51.67	44.75	47.18	13.0	29513.27
PTQ w8-a32	60.38	43.28	24.55	22.43	23.59	15.72	35079.55
PTQ w4-a32	86.71	67.30	34.13	31.31	31.73	13.0	29513.27
DARTS+QAT	54.54	50.34	24.80	22.13	23.68	15.64	28141.03

Эксперимент показал сохранение работоспособности модели GPT-Neo-125m несмотря на ее небольшой размер при сжатии модели в 2.05 раза. Результаты показывают высокий потенциал использования методов архитектурного поиска для подбора схем квантизации, в том числе, для моделей больших размеров, таких как LLaMA-7B, Qwen3-8B и других. Кроме того, полученная схема квантизации интерпретируема и показывает, какие из слоев оказываются более чувствительными к ошибке квантизации, а какие допускают более сильное сжатие: для большинства матриц весов выбирается разрядность 8 бит, тогда как отдельные проекции механизма внимания допускают более сильное сжатие до 4 бит.

Список литературы

[1] A. E. Eiben, J. E. Smith, *Introduction to evolutionary computing*, Springer, Berlin, Heidelberg, 2015, ISBN: 9783662448731,3662448748. DOI: 10.1007/978-3-662-44874-8, 287 pp.

[2] B. Zoph, Q. V. Le, "Neural architecture search with reinforcement learning", *International Conference on Learning Representations (ICLR)*, 2017 https://arxiv.org/abs/1611.01578.

[3] M. U. Nasir, S. Earle, J. Togelius, S. James, C. Cleghorn, "LLMatic: Neural architecture search via large language models and quality diversity optimization", *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Association for Computing Machinery, New York, NY, USA, 2024, 1110–1118. DOI: 10.1145/3638529.3654017.

[4] A. Sarah, S. N. Sridhar, M. Szankin, S. Sundaresan, "LLaMA-NAS: Efficient neural architecture search for large language models", *European Conference on Computer Vision (ECCV)*, Springer Nature Switzerland, Cham, Switzerland, 2024, 67–74. DOI: 10.48550/arXiv.2405.18377.

[5] H. Liu, K. Simonyan, Y. Yang, “DARTS: Differentiable architecture search”, *International Conference on Learning Representations (ICLR)*, 2019 https://arxiv.org/abs/1806.09055.

[6] B. Wu, Y. Wang, P. Zhang, Y. Tian, “Mixed precision quantization of convnets via differentiable neural architecture search”, *arXiv preprint*, 2018 https://arxiv.org/abs/1812.00090.

[7] K. He, X. Zhang, S. Ren, J. Sun, “Deep residual learning for image recognition”, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, 770–778. DOI: 10.48550/arXiv.1512.03385.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, H. Adam, “MobileNets: Efficient convolutional neural networks for mobile vision applications”, *arXiv preprint*, 2017 https://arxiv.org/abs/1704.04861.

[9] K. Simonyan, A. Zisserman, “Very deep convolutional networks for large-scale image recognition”, *International Conference on Learning Representations (ICLR)*, 2015 https://arxiv.org/abs/1409.1556.

[10] Z. Cai, N. Vasconcelos, “Rethinking differentiable search for mixed-precision neural networks”, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, 2349–2358. DOI: 10.48550/arXiv.2004.05795.

[11] P. Behnam, U. Kamal, S. Vijay Ganesh, Z. Li, M. A. Jurado, A. Khare, I. Fedorov, G. Liu, A. Tumanov, “∇QDARTS: Quantization as an Elastic Dimension to Differentiable NAS”, *Transactions on Machine Learning Research*, 2025 https://openreview.net/forum?id=ubrOSWyTS8.

[12] M. Tan, Q. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks”, *International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, 2019, 6105–6114 https://arxiv.org/abs/1905.11946.

[13] C. Zhao, T. Hua, Y. Shen, Q. Lou, H. Jin, “Automatic mixed-precision quantization search of BERT”, *arXiv preprint*, 2021 https://arxiv.org/abs/2112.14938.

[14] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019 https://arxiv.org/abs/1810.04805.

[15] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding", *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, 2018, 353–355. DOI: 10.18653/v1/W18-5446.

[16] J. Choi, Z. Wang, S. Venkataramani, P. Chuang, V. Srinivasan, K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks", *arXiv preprint*, 2018 https://arxiv.org/abs/1805.06085.

[17] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, N. Kwak, "LSQ+: Improving low-bit quantization through learnable offsets and better initialization", *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, 696–697. DOI: 10.1109/CVPRW50498.2020.00356.

[18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, 2704–2713. DOI: 10.1109/CVPR.2018.00286.

[19] Y. Bengio, N. Léonard, A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation", *arXiv preprint*, 2013 https://arxiv.org/abs/1308.3432.

[20] S. Black, G. Leo, P. Wang, C. Leahy, S. Biderman, "GPT-Neo: Large scale autoregressive language modeling with mesh-tensorflow", *Zenodo*, 2021. DOI: 10.5281/zenodo.5297715.

[21] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, C. Leahy, "The Pile: An 800GB dataset of diverse text for language modeling", *arXiv preprint*, 2020 https://arxiv.org/abs/2101.00027.

[22] S. Merity, C. Xiong, J. Bradbury, R. Socher, "Pointer sentinel mixture models", *arXiv preprint*, 2016 https://arxiv.org/abs/1609.07843.

[23] "FineWeb-EDU dataset", *Hugging Face Datasets*, 2024 https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

[24] "OpenWebText Corpus", *GitHub repository*, 2019 https://github.com/skylion007/OpenWebTextCorpus.

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, "Language models are unsupervised multitask learners", *OpenAI Technical Report*, 2019 https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[26] J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, "Compressive transformers for long-range sequence modelling", *International Conference on Learning Representations (ICLR)*, 2020 https://arxiv.org/abs/1911.05507.

[27] M. Hart, "Project Gutenberg", 1971 https://www.gutenberg.org.

[28] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, V. Chandra, "LLM-QAT: Data-free quantization aware training for large language models", *Findings of the Association for Computational Linguistics: ACL 2024*, 2024, 467–484. DOI: 10.18653/v1/2024.findings-acl.26.

[29] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs", *Advances in Neural Information Processing Systems*, **36**, 2023, 10088–10115 https://arxiv.org/abs/2305.14314.

[30] T. Dettmers, M. Lewis, Y. Belkada, L. Zettlemoyer, "LLM.int8(): 8-bit matrix multiplication for transformers at scale", *arXiv preprint*, 2022. DOI: 10.48550/arXiv.2208.07339.

Приложение А. Результаты подбора разрядностей для трансформерной генеративной модели

Автоматически подобранные битности весов и активаций для линейных проекций модели GPT-Neo-125m:

```
module.transformer.h.0.attn.attention.k_proj: W=8 (uniform) | A=8 (
    uniform)
module.transformer.h.0.attn.attention.v_proj: W=8 (uniform) | A=32 (
    identity)
module.transformer.h.0.attn.attention.q_proj: W=8 (uniform) | A=8 (
    lsq)
module.transformer.h.0.attn.attention.out_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.0.mlp.c_fc: W=8 (lsq) | A=8 (uniform)
module.transformer.h.0.mlp.c_proj: W=8 (uniform) | A=8 (lsq)
```

```
module.transformer.h.1.attn.attention.k_proj: W=8 (lsq) | A=8 (lsq)
module.transformer.h.1.attn.attention.v_proj: W=4 (uniform) | A=8 (
    pact)
module.transformer.h.1.attn.attention.q_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.1.attn.attention.out_proj: W=4 (uniform) | A=8
    (lsq)
module.transformer.h.1.mlp.c_fc: W=8 (lsq) | A=8 (pact)
module.transformer.h.1.mlp.c_proj: W=8 (uniform) | A=8 (uniform)

module.transformer.h.2.attn.attention.k_proj: W=8 (uniform) | A=32 (
    identity)
module.transformer.h.2.attn.attention.v_proj: W=8 (uniform) | A=32 (
    identity)
module.transformer.h.2.attn.attention.q_proj: W=8 (lsq) | A=32 (
    identity)
module.transformer.h.2.attn.attention.out_proj: W=8 (uniform) | A=8
    (uniform)
module.transformer.h.2.mlp.c_fc: W=8 (uniform) | A=8 (lsq)
module.transformer.h.2.mlp.c_proj: W=8 (uniform) | A=8 (uniform)

module.transformer.h.3.attn.attention.k_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.3.attn.attention.v_proj: W=8 (lsq) | A=8 (lsq)
module.transformer.h.3.attn.attention.q_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.3.attn.attention.out_proj: W=8 (uniform) | A=8
    (uniform)
module.transformer.h.3.mlp.c_fc: W=8 (uniform) | A=8 (uniform)
module.transformer.h.3.mlp.c_proj: W=8 (uniform) | A=8 (uniform)

module.transformer.h.4.attn.attention.k_proj: W=8 (uniform) | A=32 (
    identity)
module.transformer.h.4.attn.attention.v_proj: W=8 (lsq) | A=32 (
    identity)
module.transformer.h.4.attn.attention.q_proj: W=8 (uniform) | A=8 (
    lsq)
module.transformer.h.4.attn.attention.out_proj: W=8 (uniform) | A=8
    (uniform)
module.transformer.h.4.mlp.c_fc: W=8 (lsq) | A=8 (uniform)
module.transformer.h.4.mlp.c_proj: W=8 (lsq) | A=32 (identity)

module.transformer.h.5.attn.attention.k_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.5.attn.attention.v_proj: W=8 (lsq) | A=8 (lsq)
```

```
module.transformer.h.5.attn.attention.q_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.5.attn.attention.out_proj: W=8 (uniform) | A=8
    (uniform)
module.transformer.h.5.mlp.c_fc: W=8 (lsq) | A=8 (uniform)
module.transformer.h.5.mlp.c_proj: W=8 (uniform) | A=8 (uniform)

module.transformer.h.6.attn.attention.k_proj: W=8 (lsq) | A=8 (pact)
module.transformer.h.6.attn.attention.v_proj: W=8 (lsq) | A=32 (
    identity)
module.transformer.h.6.attn.attention.q_proj: W=8 (uniform) | A=8 (
    lsq)
module.transformer.h.6.attn.attention.out_proj: W=4 (uniform) | A=8
    (uniform)
module.transformer.h.6.mlp.c_fc: W=8 (uniform) | A=8 (uniform)
module.transformer.h.6.mlp.c_proj: W=8 (lsq) | A=8 (lsq)

module.transformer.h.7.attn.attention.k_proj: W=8 (uniform) | A=32 (
    identity)
module.transformer.h.7.attn.attention.v_proj: W=8 (uniform) | A=32 (
    identity)
module.transformer.h.7.attn.attention.q_proj: W=8 (uniform) | A=8 (
    pact)
module.transformer.h.7.attn.attention.out_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.7.mlp.c_fc: W=8 (lsq) | A=8 (uniform)
module.transformer.h.7.mlp.c_proj: W=8 (uniform) | A=8 (lsq)

module.transformer.h.8.attn.attention.k_proj: W=8 (lsq) | A=8 (lsq)
module.transformer.h.8.attn.attention.v_proj: W=8 (lsq) | A=32 (
    identity)
module.transformer.h.8.attn.attention.q_proj: W=8 (lsq) | A=8 (pact)
module.transformer.h.8.attn.attention.out_proj: W=4 (uniform) | A=8
    (lsq)
module.transformer.h.8.mlp.c_fc: W=8 (lsq) | A=8 (uniform)
module.transformer.h.8.mlp.c_proj: W=8 (lsq) | A=8 (pact)

module.transformer.h.9.attn.attention.k_proj: W=8 (uniform) | A=32 (
    identity)
module.transformer.h.9.attn.attention.v_proj: W=8 (lsq) | A=32 (
    identity)
module.transformer.h.9.attn.attention.q_proj: W=8 (lsq) | A=32 (
    identity)
module.transformer.h.9.attn.attention.out_proj: W=8 (uniform) | A=8
    (lsq)
module.transformer.h.9.mlp.c_fc: W=8 (lsq) | A=8 (uniform)
```

```
module.transformer.h.9.mlp.c_proj: W=8 (lsq) | A=8 (uniform)

module.transformer.h.10.attn.attention.k_proj: W=8 (lsq) | A=8 (lsq)
module.transformer.h.10.attn.attention.v_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.10.attn.attention.q_proj: W=8 (lsq) | A=8 (pact)

module.transformer.h.10.attn.attention.out_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.10.mlp.c_fc: W=8 (lsq) | A=8 (uniform)
module.transformer.h.10.mlp.c_proj: W=8 (uniform) | A=8 (uniform)

module.transformer.h.11.attn.attention.k_proj: W=8 (lsq) | A=8 (pact)

module.transformer.h.11.attn.attention.v_proj: W=8 (lsq) | A=8 (pact)

module.transformer.h.11.attn.attention.q_proj: W=8 (lsq) | A=8 (pact)

module.transformer.h.11.attn.attention.out_proj: W=8 (lsq) | A=8 (
    uniform)
module.transformer.h.11.mlp.c_fc: W=8 (lsq) | A=32 (identity)
module.transformer.h.11.mlp.c_proj: W=8 (lsq) | A=32 (identity)
```

Статья поступила 26 апреля 2026 г.

Quantization of language models via differentiable neural architecture search

D. N. Davydova

Neural architecture search is a method for automatic selection of an optimal neural network architecture for a given task based on a provided dataset. One of the approaches to neural architecture search is Differentiable architecture search (DARTS). DARTS transforms the discrete search space into a continuous one, enabling the use of gradient-based optimization to learn the architectural parameters of the network. In this paper, we investigate the applicability of DARTS for selecting quantization schemes and bit-widths for different components of generative language models, and present the results of the conducted experiments. The code for training and evaluation of the quantized model is available at: https://github.com/daria1d/Darts-QAT.

Keywords: natural language processing, neural architecture search, quantization, DARTS.

References

[1] A. E. Eiben, J. E. Smith, *Introduction to evolutionary computing*, Springer, Berlin, Heidelberg, 2015, ISBN: 9783662448731,3662448748. DOI: 10.1007/978-3-662-44874-8, 287 pp.

[2] B. Zoph, Q. V. Le, "Neural architecture search with reinforcement learning", *International Conference on Learning Representations (ICLR)*, 2017 https://arxiv.org/abs/1611.01578.

[3] M. U. Nasir, S. Earle, J. Togelius, S. James, C. Cleghorn, "LLMatic: Neural architecture search via large language models and quality diversity optimization", *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Association for Computing Machinery, New York, NY, USA, 2024, 1110–1118. DOI: 10.1145/3638529.3654017.

[4] A. Sarah, S. N. Sridhar, M. Szankin, S. Sundaresan, "LLaMA-NAS: Efficient neural architecture search for large language models", *European Conference on Computer Vision (ECCV)*, Springer Nature Switzerland, Cham, Switzerland, 2024, 67–74. DOI: 10.48550/arXiv.2405.18377.

[5] H. Liu, K. Simonyan, Y. Yang, "DARTS: Differentiable architecture search", *International Conference on Learning Representations (ICLR)*, 2019 https://arxiv.org/abs/1806.09055.

[6] B. Wu, Y. Wang, P. Zhang, Y. Tian, "Mixed precision quantization of convnets via differentiable neural architecture search", *arXiv preprint*, 2018 https://arxiv.org/abs/1812.00090.

[7] K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, 770–778. DOI: 10.48550/arXiv.1512.03385.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications", *arXiv preprint*, 2017 https://arxiv.org/abs/1704.04861.

[9] K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *International Conference on Learning Representations (ICLR)*, 2015 https://arxiv.org/abs/1409.1556.

[10] Z. Cai, N. Vasconcelos, "Rethinking differentiable search for mixed-precision neural networks", *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, 2349–2358. DOI: 10.48550/arXiv.2004.05795.

[11] P. Behnam, U. Kamal, S. Vijay Ganesh, Z. Li, M. A. Jurado, A. Khare, I. Fedorov, G. Liu, A. Tumanov, "∇QDARTS: Quantization as an Elastic Dimension to Differentiable NAS", *Transactions on Machine Learning Research*, 2025 https://openreview.net/forum?id=ubrOSWyTS8.

[12] M. Tan, Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks", *International Conference on Machine Learning (ICML)*, Proceedings of Machine Learning Research, 2019, 6105–6114 https://arxiv.org/abs/1905.11946.

[13] C. Zhao, T. Hua, Y. Shen, Q. Lou, H. Jin, "Automatic mixed-precision quantization search of BERT", *arXiv preprint*, 2021 https://arxiv.org/abs/2112.14938.

[14] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019 https://arxiv.org/abs/1810.04805.

[15] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding", *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP*, 2018, 353–355. DOI: 10.18653/v1/W18-5446.

[16] J. Choi, Z. Wang, S. Venkataramani, P. Chuang, V. Srinivasan, K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks", *arXiv preprint*, 2018 https://arxiv.org/abs/1805.06085.

[17] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, N. Kwak, "LSQ+: Improving low-bit quantization through learnable offsets and better initialization", *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, 696–697. DOI: 10.1109/CVPRW50498.2020.00356.

[18] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, 2704–2713. DOI: 10.1109/CVPR.2018.00286.

[19] Y. Bengio, N. Léonard, A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation", *arXiv preprint*, 2013 https://arxiv.org/abs/1308.3432.

[20] S. Black, G. Leo, P. Wang, C. Leahy, S. Biderman, "GPT-Neo: Large scale autoregressive language modeling with mesh-tensorflow", *Zenodo*, 2021. DOI: 10.5281/zenodo.5297715.

[21] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, C. Leahy, "The Pile: An 800GB dataset of diverse text for language modeling", *arXiv preprint*, 2020 https://arxiv.org/abs/2101.00027.

[22] S. Merity, C. Xiong, J. Bradbury, R. Socher, "Pointer sentinel mixture models", *arXiv preprint*, 2016 https://arxiv.org/abs/1609.07843.

[23] "FineWeb-EDU dataset", *Hugging Face Datasets*, 2024 https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

[24] "OpenWebText Corpus", *GitHub repository*, 2019 https://github.com/skylion007/OpenWebTextCorpus.

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, "Language models are unsupervised multitask learners", *OpenAI Technical Report*, 2019 https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[26] J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, "Compressive transformers for long-range sequence modelling", *International Conference on Learning Representations (ICLR)*, 2020 https://arxiv.org/abs/1911.05507.

[27] M. Hart, "Project Gutenberg", 1971 https://www.gutenberg.org.

[28] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, V. Chandra, "LLM-QAT: Data-free quantization aware training for large language models", *Findings of the Association for Computational Linguistics: ACL 2024*, 2024, 467–484. DOI: 10.18653/v1/2024.findings-acl.26.

[29] T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs", *Advances in Neural Information Processing Systems*, **36**, 2023, 10088–10115 https://arxiv.org/abs/2305.14314.

[30] T. Dettmers, M. Lewis, Y. Belkada, L. Zettlemoyer, "LLM.int8(): 8-bit matrix multiplication for transformers at scale", *arXiv preprint*, 2022. DOI: 10.48550/arXiv.2208.07339.

Received on April 26, 2026