

АЛГОРИТМЫ АВТОМАТИЧЕСКОГО ДИФФЕРЕНЦИРОВАНИЯ ДЛЯ МАТРИЧНЫХ ОПЕРАЦИЙ

Э.Ф.о. Ахмедов, аспирант

Ульяновский государственный педагогический университет им. И.Н. Ульянова
(Россия, г. Ульяновск)

DOI:10.24412/2500-1000-2025-2-3-100-104

Аннотация. В статье проводится анализ алгоритмов автоматического дифференцирования (АД), для вычисления производных функций, заданных в виде компьютерных программ, что особенно актуально в задачах, где аналитическое дифференцирование сопряжено со значительными трудностями или вовсе невозможно, обеспечивая при этом высокую точность, сопоставимую с аналитическим подходом, и исключая ошибки, характерные для численного дифференцирования. Особое внимание уделено предложенному А.В. Климовым алгоритму, который детально описывает как прямой, так и обратный проходы в перцептроне, предоставляя схему вычисления градиентов для обучения нейронных сетей, акцентируя внимание на четкой индексации, детальном описании операций в каждом узле, формализации вычисления градиентов посредством введения сопряженных узлов и учете доменов узлов для корректной генерации кода дифференцирования. В работе также рассмотрены особенности применения АД к матричным операциям, а именно, прямой и обратный режимы, с анализом их влияния на вычислительную эффективность, а также обоснование использования правила цепочки и преобразований функций для достижения композиционности дифференцирования. Проведен сравнительный анализ прямого и обратного режимов АД с точки зрения вычислительной сложности и затрат памяти, а также рассмотрены методы оптимизации, такие как накопление касательных в памяти и использование обратных распространителей.

Ключевые слова: автоматическое дифференцирование, матричные операции, прямой метод, обратный метод, градиент, оптимизация, вычислительная сложность.

Автоматическое дифференцирование – набор методов для вычисления производных функций, заданных компьютерными программами, важен в задачах, где аналитическое вычисление производных затруднительно или невозможно, обеспечивая высокую точность результатов, сравнимую с аналитическим дифференцированием, и исключая ошибки, свойственные численному дифференцированию [2].

А.В. Климов дает следующее определение «Автоматическое дифференцирование (АД) – это автоматическое преобразование программы на некотором языке, вычисляющей функцию, в программу на том же языке, вычисляющей производную этой функции, по заданному множеству числовых параметров, иначе говоря, ее градиент [1].

В области машинного обучения, где оптимизация параметров моделей является фундаментальной задачей, АД находит применение для вычисления градиентов функций потерь, необходимых для работы алгоритмов гради-

ентного спуска и его модификаций, позволяя эффективно обучать сложные модели, такие как нейронные сети с большим количеством параметров, и обеспечивая сходимость к оптимальным решениям [3]. Помимо машинного обучения, АД применяется в задачах оптимизации, возникающих в различных областях, включая проектирование, экономику и финансы, где требуется нахождение оптимальных значений параметров для достижения заданных целей, обеспечивая эффективный поиск экстремумов сложных целевых функций. Более того, в таких областях науки и техники, как вычислительная гидродинамика, вычислительная механика и робототехника, АД используется для анализа чувствительности решений к изменениям параметров, что позволяет проводить оптимизацию конструкций, управлять сложными системами и решать обратные задачи, требующие точного вычисления производных [4].

Предложенный А.В. Климовым алгоритм, представленный на рисунке 1, описывает как

прямой, так и обратный проходы в многослойном персептроне, предоставляя деталь-

ную схему вычисления градиентов для обучения нейронной сети [1].

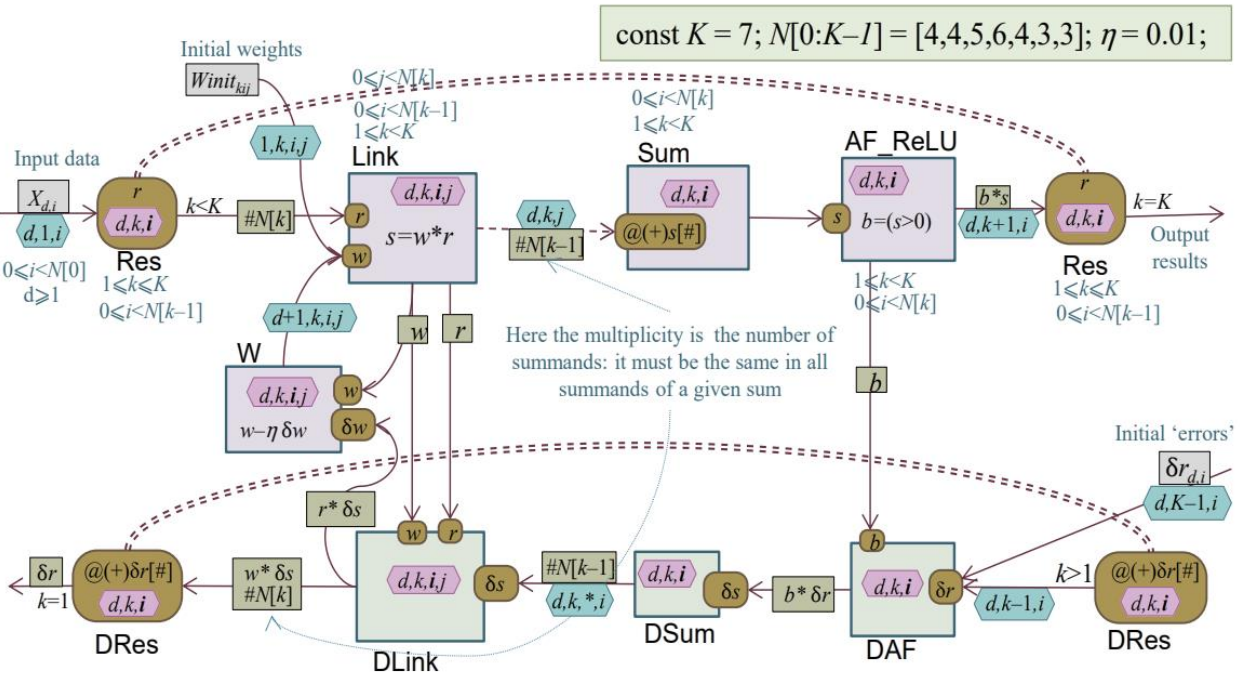


Рис. 1. Алгоритмы прямого (вверху) и обратного (внизу) проходов [1]

Автор акцентирует внимание на четкой индексации. Использование индексов d, k, i и j обеспечивает однозначную адресацию нейронов, весов и промежуточных результатов. Алгоритм подробно описывает операции в каждом узле (Link, Sum, AF_ReLU, Res), что облегчает его понимание и реализацию. Введение сопряженных узлов и правила обращения стрелок помогает формализовать вычисление градиентов. Учет доменов узлов важен для корректной генерации кода дифференцирования. Такой подход поможет автоматизировать процесс дифференцирования и генерации кода для обучения нейронных сетей, что является

важным преимуществом. Представленная схема наглядно демонстрирует взаимосвязь прямого и обратного проходов, а также роль каждого элемента в процессе обучения.

В алгоритме автоматического дифференцирования (AD), основное внимание уделяется их применению к матричным операциям. В основном рассматриваются как прямой, так и обратный режимы AD и их влияние на эффективность вычислений. AD использует программные преобразования для прямого ($\rightarrow D$) и обратного ($\leftarrow D$) режимов. Правило цепочки является основополагающим [5]:

$$(f \times g)'(x) = f'(g(x)) \times g'(x) \tag{1}$$

В данном случае, дифференциация не является непосредственно композиционной. Композиционность достигается путем преобразо-

вания функции $f: R \rightarrow R$ функцию $\rightarrow D(f): R^2 \rightarrow R^2$:

$$\rightarrow D(f): \langle z, z \rangle \mapsto \langle f(z), f'(z) \cdot z \rangle \tag{2}$$

Здесь z (первичный) - это результат g, а ž (касательный) - результат g'. Преобразование является композиционным:

$$\rightarrow D(f \circ g) = \rightarrow D(f) \circ \rightarrow D(g) \tag{3}$$

Оно обобщается на n -арные функции в оценке $\rightarrow Dn$ и является основой прямого режима AD. «Прямой» означает, что первичные и касательные вычисления выполняются в со-

ответствии с потоком ввода-вывода; $f'(z)$ вычисляется после накопления производной от ее входной функции в z . Если $|M|$ представляет размер члена M , то:

$$|\rightarrow Dn(\phi(M))| = O(n) + |\rightarrow Dn(M)| \quad (4)$$

Если F содержит только функциональные символы и переменные, вычисление F и $\rightarrow Dn(F)$ требует операций, примерно пропорциональных их размеру. Поскольку $|\rightarrow Dn(F)| = O(n|F|)$, вычисление $\rightarrow Dn(F)$ в n раз функциональнее, чем вычисление F . Если рассматривать, где F - функция потерь, а n - количество (потенциально массивных) параметров отображения, вызывает серьезную озабоченность.

Обратный режим AD (обратное распространение) обеспечивает эффективный подход. Он накапливает касательные в порядке, обратном порядку простых чисел. Используя обозначение (1), $\leftarrow D(f)$ сначала вычисляет $f'(g(x))$, а затем использует $g'(x)$ для умножения. Как отмечают В.А.Pearlmutter, J.M. Siskind [5], естественным образом выражается в функциональном программировании заменой касательных переменных на обратные распространители z^{**} , которые являются функциями (продолжениями):

$$\leftarrow D(f): \langle z, z^* \rangle \mapsto \langle f(z), \lambda a. z^* (f'(z) \cdot a) \rangle \quad (5)$$

Обратный режим преобразует \mathbb{R} в \mathbb{R}^n , ожидая действительного числа (производной следующей функции) для вычисления градиента всей функции. В (5) второй компонент является обратным распространителем функции f , и z^{**} - обратным распространителем входной функции f (в (1)). В обобщенном виде равно $\leftarrow Dn$.

$D(F)$ также имеет первый порядок, но $\leftarrow D(F)$ имеет более высокий порядок. В функциональном языке программирования, где распространено обратное распространение, накопление касательных в памяти помогает избежать дублирования. Функциональный язык программирования требуют таких методов, как замыкающие преобразования или обращения к памяти и продолжения с разделителями [6].

Если проверить, что $|\leftarrow Dn(\phi(M))| = O(1) + |\leftarrow Dn(M)|$, и если F содержит только функциональные символы и переменные, вычисление $\leftarrow Dn(F)$ асимптотически так же затратно, как и вычисление F . Однако при совместном использовании (например, при многократном вызове подпрограммы) $\leftarrow D(F)$ может излишне дублировать вычисления обратного распространения, что приводит к экспоненциальному увеличению. Ключевое отличие: если F имеет первый порядок, то \rightarrow

Данное обсуждение следует из мнения A.Brunel, D.Mazza, M.Pagani, в котором используются типы линейной логики - обратные распространители имеют тип $\mathbb{R} \perp^n$ (линейные отображения от \mathbb{R} до \mathbb{R}^N). Правило факторинга в операционной семантике показывает, что необходимо совместно использовать оценки обратных распространителей [3]:

$$zM + zN \rightarrow z * (M + N) \quad (6)$$

В данном случае основное внимание уделяется обоснованности, а не эффективности, при использовании $\mathbb{R} \perp^n := \mathbb{R} \rightarrow \mathbb{R}^n$ без правила линейного разложения на множители (6), которое только ускоряет вычисления без внесения ошибок. Обозначение $(-)\perp$ служит напоминанием о предполагаемой линейной стрелке. Преобразование [Brunel et al., 2020] оста-

ется хорошо типизированным при таком «нелинейном» определении отрицания.

Преобразования AD были объяснены для примитивных функций («элементарных блоков»). Как формализовано F.Wang et al., они распространяются на произвольные программы по функциональности $\rightarrow D$ и $\leftarrow D$ заменяются программными конструкциями. Аб-

структное синтаксическое дерево в основном сохранено, благодаря двум переменным, но изменены только типы переменных, чтобы учесть основные значения и тангенсы или обратным распространителям. Часто описывается AD, как «перегрузка оператора», но «функциональность» технически более точна [6].

$$D(\text{if}(P, M, N)) := \text{if}(D'(P), D(M), D(N)) \quad (7)$$

где D' – вспомогательное преобразование. Не имеет решающего значения для результатов и не имеет отношения к данному уровню абстракции.

Примеры двух режимов D приведены на рисунке 3. Вычитание (рисунок 3b) показывает, как размер члена прямого преобразования увеличивается с увеличением размера гради-

ента n , в то время как он остается постоянным при $\leftarrow D_n$.

Учитывая $\Gamma \vdash M: A$ с $\Gamma = {}_1^A 1x, \dots, x_n^{An}$
то $d(\Gamma) \vdash D(M): D(A)$

где $d(\Gamma) = x_1^{D(A_1)}, \dots, x_n^{D(A_n)}$

Если M - программа, то
 $x_1^{R \times R^n}, \dots, x_n^{R \times R^n} \vdash \rightarrow D_n(M): R \times R^n$

$$x_1^{R \times (R^n)^\perp}, \dots, x_n^{R \times (R^n)^\perp} \vdash \leftarrow D_n(M): \times R (R^n)^\perp \quad (8)$$

Если M является простым, то вычислительное поведение $\rightarrow D$ и $\leftarrow D$ определяется обоснованностью AD для простых терминов, которое связывает оценку преобразованного члена с градиентом исходного члена.

Для произвольной программы M с арностью n и кратностью l следующие программы вычисляют $\nabla JMK\Gamma$ при $r = (r_1, \dots, r_n) \in R^n$ (если определено):

$$\rightarrow \text{grad}_n(M)(P) := \pi^2 2(\rightarrow D_n(M)\{r_1, l_1^n 1/x_1\} \dots \{r_n, l_n^n 1/x_n\}), \quad (9)$$

$$\leftarrow \text{grad}_n(M)(P) := \pi^2 2\left(\leftarrow D_n(M)\left\{r^1, \frac{l^{1n}}{x^1}\right\} \dots \left\{r_n, \frac{l^n}{x^n}\right\}\right) 1 \quad (10)$$

Такие примеры, иллюстрируют поведение вычисления градиента, выделяя случаи, когда вычисленный градиент совпадает с аналитическим градиентом только почти везде из-за точек недифференцируемости. Различные стратегии сокращения влияют на область сходимости и дифференцируемости, но если доказана обоснованность максимальной стратегии, то это справедливо и для других стратегий.

В отличие от численного дифференцирования, которое вносит ошибку аппроксимации, АД обеспечивает вычисление градиентов с точностью до ошибок округления, что способствует стабильному и эффективному обучению. На практике, для оценки производительности и точности обычно используются такие метрики, как время обучения на эпоху, скорость сходимости функции потерь, точность на валидационном и тестовом наборах

данных, а также другие специфические метрики, зависящие от конкретной задачи. Важно отметить, что выбор метода оптимизации и его параметров, таких как скорость обучения и размер батча, также оказывает существенное влияние на производительность и точность обучения, и часто требует экспериментального подбора.

Выводы. Проведенное исследование, посвященное алгоритмам автоматического дифференцирования (АД) для матричных операций, позволило углубить понимание особенностей применения различных методов АД, включая прямой и обратный методы, к операциям с матрицами, что является важным шагом в развитии вычислительных методов для решения широкого круга задач в науке и технике.

В качестве предложений по улучшению существующих методов можно отметить

необходимость дальнейшей разработки и оптимизации алгоритмов АД для разреженных матриц, что позволит существенно сократить вычислительные затраты и объем требуемой памяти при работе с матрицами большого размера, содержащими большое количество нулевых элементов.

Актуальным направлением является исследование и разработка гибридных методов АД, комбинирующих преимущества прямого и обратного методов для достижения опти-

мальной вычислительной эффективности в зависимости от структуры конкретной задачи. Перспективным направлением является разработка специализированных программных библиотек, реализующих разработанные алгоритмы АД для матричных операций, что позволит сделать их доступными для широкого круга пользователей и способствовать их активному применению в различных областях науки и техники.

Библиографический список

1. Климов А.В. Автоматическое дифференцирование в потоковом языке на примере задачи глубокого обучения / А.В. Климов // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). – 2021. – № 3. – С. 94-98.
2. Baydin A., Pearlmutter B., Radul A., Siskind J. 2018. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*. – 2018. – Vol. 18. – P. 1-43.
3. Brunel A., Mazza D., Pagani M. Backpropagation in the simply typed lambda-calculus with linear negation. *PACMPL* 4, POPL. 2020. Vol. 64. p.1-27.
4. Elliott C. The simple essence of automatic differentiation. *PACMPL* 2, ICFP. – 2018. – Vol. 70. – P.1-29.
5. Pearlmutter B.A., Siskind J.M. Reverse-mode AD in a Functional Framework: Lambda the Ultimate Backpropagator. *ACM Trans. Program. Lang. Syst.* – 2008. – Vol. 30,2. – P. 7:1-36.
6. Wang F., Zheng D., Decker J.M., Wu X., Essertel G.M., Rompf T. Demystifying differentiable programming: shift/reset the penultimate backpropagator. *PACMPL* 3, ICFP. – 2019. – Vol. 96. – P. 1-31.

AUTOMATIC DIFFERENTIATION ALGORITHMS FOR MATRIX OPERATIONS

E.F.o. Ahmedov, *Postgraduate Student*

Ulyanovsk State Pedagogical University named after I.N. Ulyanov
(Russia, Ulyanovsk)

Abstract. *The article analyzes automatic differentiation (AD) algorithms for calculating derivatives of functions specified in the form of computer programs, which is especially important in problems where analytical differentiation is associated with significant difficulties or is completely impossible, while ensuring high accuracy comparable to the analytical approach and excluding errors typical of numerical differentiation. Particular attention is paid to the algorithm proposed by A.V. Klimov, which describes in detail both the forward and backward passes in the perceptron, providing a scheme for calculating gradients for training neural networks, focusing on clear indexing, a detailed description of operations in each node, formalization of gradient calculation by introducing conjugate nodes and taking into account node domains for correct generation of differentiation code. The paper also considers the features of applying AD to matrix operations, namely, direct and reverse modes, with an analysis of their impact on computational efficiency, as well as a justification for using the chain rule and function transformations to achieve compositionality of differentiation. A comparative analysis of the direct and inverse modes of AD is carried out in terms of computational complexity and memory costs, and optimization methods such as accumulation of tangents in memory and the use of back propagators are considered.*

Keywords: *automatic differentiation, matrix operations, direct method, inverse method, gradient, optimization, computational complexity.*