Автоматическое дифференцирование в потоковом языке на примере задачи глубокого обучения

Арк.В. Климов

Институт проблем проектирования в микроэлектронике РАН, г. Москва, arkvklimov@yandex.ru

Аннотация — Описывается метод преобразования заданной программы в программу вычисления производных по параметрам в потоковом языке параллельного программирования UPL. Метод иллюстрируется на примере многослойного персептрона, где из алгоритма прямого вычисления выходов по входам автоматически порождается алгоритм вычисления градиента методом обратного распространения, который затем используется в процессе обучения

Ключевые слова — автоматическое дифференцирование, обратное распространение, глубокое обучение потоковый язык, графическое программирование.

I. ВВЕДЕНИЕ

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. Эта задача имеет давнюю историю [1]. В машинном обучении обычно используется ее специальный случай, рассчитанный на исходные функции определенного класса, заданные как многослойные нейронные сети с весами на связях в качестве параметров (Torch [2], TensorFlow [3]). Существуют, однако, и системы АД общего вида, допускающие программы на обычном языке программирования, как Python – PyTorch [4]. Их использование позволяет существенно расширить арсенал способов задания входных прямых алгоритмов, избавляя от кропотливой и склонной к ошибкам работы по разработке соответствующей программы вычисления производных.

Существующие методы АД рассчитаны на обычные языки последовательного программирования (Фортран, C++, Java, Python). Дифференцирование с обратным накоплением обычно включает предварительное вычисление заданной функции прямым алгоритмом с запоминанием (обычно в стеке) некоторой промежуточной информации. Затем она используется при вычислении производных исходной функции на обратном проходе, в котором для каждого оператора исходного кода выполняется соответствующий adjoint оператор, причем в обратном порядке [5].

Мы производим АД в рамках нашего графического языка UPL[6,7] потокового типа, являющегося одним из входных языков архитектуры параллельной потоковой

вычислительной системы (ППВС), разрабатываемой в ИППМ РАН [8]. В отличие от TF и Torch программа в UPL обрабатывает не тензоры или массивы, а скалярные значения, во многом подобно CUDA, где в рамках ядра (kernel) мы имеем дело с кодом, обрабатывающим отдельные элементы. Однако в UPL массивы или тензоры не возникают и «между» вызовами ядер – просто потому, что никакого «между» не существует. А данные существуют всегда виртуально «в россыпь», но каждый скаляр снабжен полным тегом (индексом), определяющим его место среди других.

Статья имеет следующую структуру. Сначала (раздел 2) дается краткое описание средств графического языка UPL. Затем (раздел 3) приведен (как пример) алгоритм многослойного персептрона на этом языке: рисунок и пояснения к нему. В разделе 4 описаны общие правила построения программы дифференцирования (методом обратного распространения) в UPL. Правила иллюстрируются на примере многослойного персептрона: по описанию прямого алгоритма выполнения персептрона по описанным формальным правилам строится алгоритм вычисления производных по параметрам (весам). Для получения обучаемого персептрона остается добавить схему коррекции весов (раздел 5).

II. ОПРЕДЕЛЕНИЕ ЯЗЫКА

В UPL порядок между операторами (узлами) определяется зависимостями по данным: узел срабатывает, когда на всех его входах имеются токены, несущие аргументы для узла. В исполняющей системе (ППВС) имеется процессор сопоставлений (ПС), основанный на ассоциативной памяти, который выявляет все случаи готовности. В результате активации узла выполняется записанная в нем небольшая программа (в стандартном языке программирования), дополненная операторами отправки новых токенов на входы других узлов.

Программа на UPL содержит обычно несколько *типовых* узлов. Типовой узел в графической форме изображается прямоугольником (бледно-розовым), помеченным именем (см. пример на Рис.1). На его границе (обычно левой) помещаются *входы* в форме овалов (желтых). Внутри узла задан список переменных (полей) *индекса* (в ярко-розовом шестиугольнике), которые при активации должны принять конкретные значения (обычно целые). Имя типового узла и

значения полей индекса вместе определяют адрес конкретного *виртуального* узла. Входы снабжены именами и типами. На рисунке типы опущены, подразумеваются вещественные. Внутри узла может быть задано вычисляемое *выражение*, определяющее выходное значение. Оно вычисляется при активации. В нем могут использоваться входы, поля индекса и константы. Значение может присваиваться новой переменной.

Из одного узла во вход другого могут вести стрелки, по которым перемещаются токены. На стрелке в общем случае находятся:

- в прямоугольнике (сером) – передаваемое значение (данное),
- в шестиугольнике (голубом) – индекс целевого узла,
- у основания – условие выхода.

Все эти элементы задаются в виде выражений, зависящих от входов или полей индекса узла-источника, а также констант, задаваемых в отдельной декларации (на Рис.1 вверху справа). Если на стрелке отсутствует значение, то оно по умолчанию равно значению выражения в теле узла-источника или значению его единственного входа (или одноименной с целевым входом переменной).

В индексе на месте некоторых полей может стоять знак «*», указывающий, что значение этого поля в токене не определено. (Оно должно быть определено в токене на другой вход.) Такой токен может участвовать в активации виртуального узла с любым значением в индексе на месте «*». Если отсутствует весь индекс, он по умолчанию составляется из имен целевого индекса, определенных в узле-источнике, а на позиции не определенных в источнике имен ставится «*». При активации узла по стрелкам передаются сформированные токены (при выполнении условия выхода).

Если узел имеет единственный вход и не имеет выражения, он может изображаться отдельным желтым овалом с индексом.

В токене может быть задана кратность – выражение после значения, отделенное знаком #. Оно показывает число активаций, в которых данный токен может участвовать. Кратность может быть бесконечной (##). По умолчанию кратность 1.

Некоторые входы могут быть собирающими. Имя такого входа имеет префикс вида @($\square$), где $\square$ – знак ассоциативной и коммутативной операции, и суффикс вида [n], где n – ожидаемое число слагаемых (оно может быть или константой или именем другого входа или знаком «#», если число слагаемых задается в виде «кратности» каждого из токенов-слагаемых).

III. СХЕМА ПЕРСЕПТРОНА

На Рис.1 вверху показан алгоритм работы прямого прохода многослойного персептрона. Здесь поля индекса следующие: d – номер примера (от 1), k –

номер слоя, i – номер нейрона в слое, j – номер нейрона в следующем слое (для весов и связей). Константный массив $N[0:K-1]$ задает число нейронов в каждом слое, включая входной. Результирующая структура сети показана на Рис. 2.

Узел Link умножает значение r на вес w . На его вход w подаются токены с полным индексом $\langle d, k, i, j \rangle$, а на вход r – с неполным индексом $\langle d, k, i, * \rangle$ и кратностью $N[k]$. Токен r спаривается с $N[k]$ токенами w и всякий раз произведение подается на суммирующий узел Sum с индексом $\langle d, k, j \rangle$, то есть на место индекса i узла Sum ставится индекс j узла Link. Число слагаемых задается в форме кратности в каждом слагаемом, здесь это $N[k-1]$, то есть размер предыдущего слоя. Результат суммирования s подается на вход функции активации AF_ReLU. Она вычисляет логическое значение $b = (s > 0)$ и посылает на выход произведение $b * s$ (понимая b как 0 или 1 по правилам языка C). Выходной узел Res здесь «склеен» с узлом Res на входе, то есть это в «реальности» один и тот же узел (что обозначено двойным пунктиром). При $k < K$ токены из него поступают на узел Link, а при $k = K$ – на выход сети. Начальные входные значения $X_{d,i}$ поступают на узел Res с индексом $\langle d, 1, j \rangle$. Синие (на белом фоне) стрелки и тексты на Рис. 1 являются комментариями.

IV. ПОРОЖДЕНИЕ ПРОГРАММЫ ДИФФЕРЕНЦИРОВАНИЯ

Введем функцию потерь, которая сравнивает выходы из сети $R_{d,i}$ с эталонными значениями $Y_{d,i}$ и вырабатывает неотрицательное скалярное значение L . Нам будут нужны производные этой величины по всем (весовым) параметрам. Будем использовать цепное правило вычисления производной сложной функции, накапливая произведения в обратном порядке. Производную потери L по промежуточной величине Z будем обозначать как δZ . Если в качестве функции потерь взять полу-сумму квадратов разностей $R_{d,i} - Y_{d,i}$, то первым шагом будет вычисление производных по самим выходам сети, $\delta R_{d,i}$, которые в этом случае будут в точности равны разностям $R_{d,i} - Y_{d,i}$, то есть ошибкам сети (errors). Чтобы не усложнять схему, с них и начнем.

Будем строить граф вычисления производных для всех промежуточных значений исходного графа. Для каждого исходного узла введем параллельный сопряженный (adjoint) узел с тем же индексом. Пусть исходный узел N принимает аргументы $s1$ и $s2$ и выдает результат r . Тогда сопряженный узел DN будет принимать производную δr (и возможно, что-то еще из прямого прохода) и выдавать производные $\delta s1$ и $\delta s2$. Таким образом, внутренние стрелки сопряженного графа получаются обращением стрелок исходного графа. Индексные выражения на стрелках пересчитываются на контекст нового целевого узла.

Для вычислений в узле DN нам понадобятся частные производные выхода узла N по его входам. Они в общем случае зависят от самих входов узла N, поэтому некоторые значения придется передать от узла N к узлу DN с тем же индексом. Дифференцирование

внутренней программы узла N выполняется классическими методами. На каждый выход δs_i узла DN выдается частная производная выхода r узла N по соответствующему входу s_i , умноженная на δr .

Чтобы правильно выставить условия на стрелках, транслятору надо знать *домены* узлов — множества

допустимых значений индексов. На схеме они показаны рядом с каждым исходным узлом синим цветом в виде комментариев. Здесь это многогранники, заданные как наборы линейных условий. Возможно, пользователю придется указать только некоторые из доменов так, чтобы транслятор мог вывести остальные сам.

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

На основе доменов расставляются условия на стрелках. Так, на стрелке между DLink и DRes условие теперь не нужно, но требуется условие $k < K$ на стрелке от DRes к DAF.

суммирующий узел (с одной входящей стрелкой) перейдет в узел, порождающий обратный токэн с кратностью, равной исходному числу слагаемых.

В задании на автоматическое дифференцирование следует указать, по каким переменным будут нужны производные: в нашем случае это весовые входы $\text{Link.w}\langle d,k,i,j \rangle$. Токэн δw порождается узлом $\text{DLink}\langle d,k,i,j \rangle$. Постпроцессор «вычислит» вычисления тех производных, которые не используются для вычисления указанных производных.

V. КОРРЕКЦИЯ ВЕСОВ

Осталось использовать вычисленные производные δw для коррекции весов. Соответствующий узел W добавим «руками». Он принимает копию использованного веса w и производную δw , и вычисляет новый вес по формуле $w = w - \eta \delta w$, где η — скорость обучения (константа). Данный вариант соответствует стохастическому методу обучения, в котором веса корректируются после прогона каждого примера. Несложно переделать его так, чтобы выполнялась корректировка после прогона каждой группы из m примеров.

Рис. 2. Структура многослойной нейросети, имеющей 1 входной, 1 выходной и 5 промежуточных слоев ($K = 7$, $N[K] = [4,4,5,6,4,3,3]$)

Разветвления в исходном графе переходят в сложения в сопряженном графе. Если разветвление неявное (то есть представлено одним токеном-стрелкой с множественным индексом и кратностью), то при его обращении появится суммирующий вход. При этом кратность исходного токэна остается в позиции кратности обратного токэна, только теперь она выражает количество слагаемых. И наоборот, исходный

VI. ЗАКЛЮЧЕНИЕ

Мы описали графический язык UPL и определили правила преобразования программы вычисления функции в программу вычисления ее производных по заданному множеству параметров. Конечно, не для любой такой программы это возможно. В частности, требуется, чтобы выражения для индексов в токенах были обратимыми, чтобы домены узлов были выразимы как квазилинейные многогранники и чтобы каждый конкретный виртуальный узел исполнялся не более одного раза.

Метод был продемонстрирован на примере стандартной многослойной нейросети — персептрона. Для него по приведенным общим формальным правилам порождается классический алгоритм глубокого обучения методом обратного распространения ошибок. Пока эти правила существуют только «на бумаге», их реализация — вопрос ближайшего будущего. И тогда, варьируя определения сетей как функций с параметрами, мы можем автоматически синтезировать эффективные программы дифференцирования по заданным параметрам, чтобы использовать их в процессах обучения настройкой данных параметров. Также возможны и другие виды использования АД в задачах нелинейной оптимизации.

Подобные инструменты существуют уже давно для традиционных языков фон-неймановского типа. Отметим достаточно свежую систему Tangent для языка Python [9]. Может ли на основе нашего потокового языка UPL быть построена конкурентная практическая система для этой области, сказать трудно, но представляется, что она может иметь ценность как инструмент для исследований — как самих определяемых функций, так и методов их дифференцирования. В рамках данного языка UPL эти методы отличаются разительной простотой и наглядностью. Отметим также высокую степень распределенного параллелизма, присущего программам на этом языке [10].

ЛИТЕРАТУРА

[1] Baydin A.G., Pearlmutter B.A., Radul A.A., Siskind J.M. Automatic Differentiation in Machine Learning: a Survey.

Journal of Machine Learning Research. 2018. Vol. 18, N. 153. pp. 1 – 43.

[2] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-like environment for machine learning. In BigLearn, NIPS Workshop, number EPFL-CONF192376, 2011. URL: https://cs.nyu.edu/~koray/files/2011_torch7_nipsw.pdf.

[3] Mart'ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016. URL: <https://arxiv.org/pdf/1603.04467.pdf>.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In NIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques, Long Beach, CA, US, December 9, 2017, 2017.

[5] Bartholomew-Biggs M., Brown S., Christianson B., Dixon L. Automatic differentiation of algorithms. Journal of Computational and Applied Mathematics. 2000. Vol. 124, N. 12. pp. 171–190.

[6] А.В. Климов. О парадигме универсального языка параллельного программирования // Языки программирования и компиляторы-2017. Труды конференции, PLC–2017 (3–5 апреля 2017), ЮФУ, Ростов-на-Дону, 2017, с. 141–146. URL: <http://plc.sfedu.ru/files/PLC-2017-proceedings.pdf> (14.10.2020).

[7] Климов А.В., Окунев А.С. Графический потоковый метаязык для асинхронного распределенного программирования // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. № 2. С. 151–158. URL: <http://www.mes-conference.ru/data/year2016/pdf/D149.pdf>.

[8] Стемпковский А.Л., Левченко Н.Н., Окунев С.А., Цветков В.В. Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов // Информационные технологии, 2008, №10. С. 2–7.

[9] URL: <https://github.com/google/tangent>.

[10] А.В. Климов. Средства верификации распределения вычислений в потоковой архитектуре ППВС «Буран» // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 4. С. 236–243.

Automatic Differentiation in a Dataflow Language on the Example of a Deep Learning Problem

Ark.V. Klimov

Institute of Design Problems in Microelectronics of RAS, Moscow, arkvklimov@yandex.ru

Abstract — We describe a method for automatic converting a given program into a program for calculating derivatives with respect to designated parameters. The original program as well as the converted one is represented in our dataflow parallel programming language UPL. A program in UPL is graphically represented as a graph of nodes connected with arcs. A node has several input ports and calculates a function, sending result to other node's ports. A program node denotes a family of activation nodes (or node instances), each with its own index, or tag. An arc carries a result value expression and a target tag expression, which are calculated in the activation. Having almost arbitrary program graph in UPL that calculates a single scalar result value the hypothetical converter builds another program graph, which calculates the set of derivatives of the result of original program w.r.t. its parameters. The converter works on per node basis adding an adjoint node for each source node. Inverted adjoint arcs, each carrying a derivative of the result with respect to original value, connect these adjoint nodes. The arc's tag expressions are rewritten to the other node's context (it must be feasible). A special kind of arcs — multicast fan-out arcs and summation fan-in arcs — are transformed one into another. The constructed adjoint graph calculates the required derivatives in backward direction. As an example, a program of a multilayer neural network (perceptron) is considered, for which a classical backpropagation algorithm is built according to the described rules. It calculates derivatives of the loss function with respect to weight parameters, which can be used further for weights correction in the frame of deep learning process.

Keywords — automatic differentiation, back propagation, deep learning, dataflow language, graphical programming.

REFERENCES

- [1] Baydin A.G., Pearlmutter B.A., Radul A.A., Siskind J.M. Automatic Differentiation in Machine Learning: a Survey. *Journal of Machine Learning Research*. 2018. Vol. 18, N. 153. pp. 1 – 43.
- [2] Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF192376, 2011. URL: https://cs.nyu.edu/~koray/files/2011_torch7_nipsw.pdf.
- [3] Mart'ın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. URL: <https://arxiv.org/pdf/1603.04467.pdf>.
- [4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques*, Long Beach, CA, US, December 9, 2017, 2017.
- [5] Bartholomew-Biggs M., Brown S., Christianson B., Dixon L. Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics*. 2000. Vol. 124, N. 12. pp. 171–190.
- [6] Klimov Ark.V. O paradigme universal'nogo yazyka parallel'nogo programmirovaniya (On the paradigm of the universal language for parallel programming) // *Yazyki programmirovaniya i kompilyatory (Programming Languages and Compilers)* — 2017. *Trudy konferencii*. Rostov-na-Donu: YuFU, 2017. P. 141–146. URL: <http://plc.sfedu.ru/files/PLC-2017-proceedings.pdf> (access date: 14.10.2020).
- [7] Klimov A.V., Okunev A.S. A graphical dataflow meta-language for asynchronous distributed programming. *Problemy razrabotki perspektivnykh mikro- i nanoelektronnykh sistem (MES)* 2016. N.2, pp. 151–158. URL: <http://www.mes-conference.ru/data/year2016/pdf/D149.pdf>.
- [8] Stempkovskij A.L., Levchenko N.N., Okunev A.S., Cvetkov V.V. Parallelnaya potokovaya vychislitelnaya sistema — dalnejshee razvitiye arkhitektury i strukturnoj organizacii vychislitelnoj sistemy s avtomaticheskimi raspredeleniem resursov (Parallel dataflow computing system — further development of the architecture and structural organization of the computing system with automatic allocation of resources). *Zhurnal «INFORMACIONNYE TEHNOLOGII»*. 2008, № 10, pp. 2–7 (In Russian).
- [9] URL: <https://github.com/google/tangent>.
- [10] Klimov A.V. Tools for verification of computation distribution in the Parallel Dataflow Computing System (PDCS) “Buran” // *Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020*. Issue 4. P. 236-243. doi:10.31114/2078-7707-2020-4-236-243.