Дата поступления рукописи в редакцию: 28.05.2026 г.
Дата принятия рукописи в печать: 19.06.2026 г.
DOI: 10.24412/2224-9133-2026-5-435-445

СЕМЕНОВ Александр Александрович,
научный сотрудник ФКУ НПО «СТиС» МВД России,
e-mail: mail@law-books.ru

АВЕТИСЯН Карэн Рафаелович,
Старший преподаватель кафедры информационных технологий и организации расследования киберпреступлений Московская академия Следственного комитета Российской Федерации имени А.Я. Сухарева,
e-mail: Karen-Avetisyan-1989@bk.ru

ОПТИМИЗАЦИЯ ВЫЧИСЛЕНИЙ В НЕЙРОННЫХ СЕТЯХ: АЛГОРИТМЫ, ПРОГРАММНЫЕ И АППАРАТНЫЕ РЕШЕНИЯ

Аннотация. *Исследуются различные подходы к оптимизации вычислений в нейронных сетях, включая алгоритмические, программные и аппаратные методы. Рассматриваются особенности оптимизации для полносвязных и сверточных архитектур, анализируются принципы работы специализированных процессоров (нейроморфных и TPU). Описываются механизмы квантования данных и их влияние на производительность. Особое внимание уделяется проблемам оптимизации памяти и эффективности вычислений в современных системах машинного обучения.*

Ключевые слова: *оптимизация вычислений, нейронные сети, алгоритмическая оптимизация, программная оптимизация, аппаратная оптимизация, квантование данных, нейроморфные процессоры, TPU, матричные вычисления, производительность систем.*

SEMENOV Alexander Alexandrovich,
Chief Researcher, Federal State Budgetary Institution "STIS" of the Ministry of Internal Affairs of Russia

AVETISYAN Karen Rafaelovich,
Senior Lecturer, Department of Information Technology and Cybercrime Investigation Organization Moscow Academy of the Investigative Committee of the Russian Federation named after A. Ya. Sukharev

OPTIMIZING COMPUTATIONS IN NEURAL NETWORKS: ALGORITHMS, SOFTWARE, AND HARDWARE SOLUTIONS

Annotation. *This paper explores various approaches to optimizing computations in neural networks, including algorithmic, software, and hardware methods. Optimization features for fully connected and convolutional architectures are discussed, and the operating principles of specialized processors (neuromorphic and TPU) are analyzed. Data quantization mechanisms and their impact on performance are described. Particular attention is paid to the problems of memory optimization and computational efficiency in modern machine learning systems.*

Key words: *computation optimization, neural networks, algorithmic optimization, software optimization, hardware optimization, data quantization, neuromorphic processors, TPUs, matrix computing, system performance.*

Оптимизация вычислений нейросетей подчиняется тем же законам, что и оптимизация математических вычислений в целом, хоть и имеет некоторые характерные особенности. Под оптимизацией понимается достижение максимальной скорости исполнения алгоритма программой на устройстве, соответственно, она бывает алгоритмическая (когда используя математические приемы пытаются заменить одни вычисления на другие, более

быстрые, но приводящие к тому же результату), программная (когда для выбранного устройства пытаются разработать наибыстрейшую программу) и аппаратная (когда для исполнения алгоритма разрабатываются специализированные вычислительные устройства).

Алгоритмические оптимизации не слишком широко распространены в машинном обучении, поскольку нейросети придумываются человеком для выполнения ровно тех вычислений, которые в них заложены, т. е. не подразумевают алгоритмических преобразований. В ситуациях, когда такая оптимизация востребована, чаще всего она лишь сводится к замене одних слоев другими. Тем не менее есть исключения, например, применение специальных преобразований, экономящих число операций умножения, к сверточным слоям (речь о преобразованиях Фурье и Винограда, позволяющих с помощью алгоритмических ухищрений уменьшить число операций в сверточном слое, сохранив результат. Однако применение этих преобразований приводит к появлению небольших арифметических погрешностей и, что важнее, значительно сужает возможность применения программной оптимизации. Преобразование Винограда более эффективно, проще реализуется и чаще встречается на практике)11F [9]. Наиболее распространенной алгоритмической оптимизацией нейросетей является использование типов данных меньшей разрядности, чем тот, в котором изначально хранятся веса нейросети, чаще всего пытаются применять 8-разрядный целочисленный тип вместо 32-разрядного float. Такой переход связан с существенной потерей точности в связи с уменьшением числа бит. Кроме того, числа, которые можно в точности представить в типах с плавающей точкой (например, float), расположены на числовой шкале неравномерно (это проиллюстрировано на *рисунке 1* голубым цветом), а числа, которые представляются целочисленными типами, наоборот находятся на равных интервалах друг от друга (на рис. 1 проиллюстрировано зеленым цветом). Переход от использования одного типа к другому, к тому же меньшей разрядности, очень нетривиален.

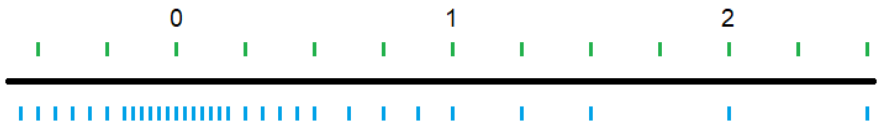

Рисунок 1. Числовая шкала

Квантование нейросети – преобразование нейросети, необходимое для обеспечения наименьшей потери точности нейросети при переходе к использованию для хранения значений и выполнения вычислений нейросети другого типа данных.

Для выполнения квантования нейросети необходимо правильно определить диапазоны данных, возникающих внутри нейросети, а это можно сделать только выполнив на примерах данных инференс уже обученной нейросети. Соответственно, квантование нейросети требует наличия некоторого количества примеров входных данных. После того, как инференс на этих примерах выполнен, подсчитываются границы диапазонов и средние значения возникающих на всех слоях нейросети чисел, после чего веса нейросети изменяются так, чтобы результат работы нейросети не изменился, но значения весов и возникающих во время вычислений чисел насколько возможно лучше соответствовали применяемому типу данных. Существуют более продвинутые техники квантования, заключающиеся в квантовании сразу во время обучения, обучении нейросетей, лучше поддающихся квантованию, и т. д., но применение этих техник в большей или меньшей степени ограничивает потенциал получаемых нейросетей по сравнению с ситуацией, при котором квантуют уже обученные нейросети.

Перед тем, как разбираться в аппаратных и программных оптимизациях, следует в общих чертах ознакомиться с тенденциями современных вычислительной техники и высокопроизводительного программирования:

– законы физики не позволяют до бесконечности ускорять частоту процессоров, поэтому мощность современных процессоров максимизируется за счет добавления в них как можно большего числа элементов, способных выполнять вычисления. Подразумеваются так называемые арифметико-логи-

ческие устройства – АЛУ (далее по тексту будет использоваться именно эта аббревиатура);
- любой математический алгоритм содержит в себе определенный потенциал распараллеливания, какие-то вычисления в алгоритме можно выполнять в любом порядке, а, значит, и параллельно, а какие-то – только последовательно;
- чем больше вычислений выполняются в процессоре параллельно, тем больше поток данных из памяти в процессор и обратно и, следовательно, выше требования к объему и скорости памяти;
- чем больше физический элемент памяти, тем сложнее организовать в него программный доступ, поэтому память с большим объемом обычно медленнее, чем память с меньшим объемом, откуда возникает невозможность обеспечить одновременно нужные скорость и объем. Эта проблема известна под названием «стена памяти», она проявляется множеством способов, поэтому существует масса равнозначных ее формулировок;
- как следствие предыдущих свойств, оптимизация алгоритма (аппаратная или программная) сводится к тому, чтобы минимизировать обращения в большие элементы памяти за счет максимального использования элементов памяти меньшего размера.

Обратим внимание на последний тезис. Возможность чаще использовать данные из меньшей памяти существует только тогда, когда одни и те же данные участвуют в вычислениях несколько раз. Сравним две операции: умножение матрицы A с размерами 1000x1000 на вектор x из 1000 значений и перемножение двух матриц U и V с размерами 100x100 (*рисунок 2*).

$$y = Ax, N_{\text{выч}} = 1000000, N_{\text{знач}} = 1002000,$$
$$W = U \cdot V, N_{\text{выч}} = 1000000, N_{\text{знач}} = 30000$$

Рисунок 2. Операции с матрицами.

На первый взгляд кажется, что вторая операция сложнее оптимизируется, однако в действительности во второй операции каждое входное значение используется 100 раз, в первой операции 100 раз используются только значения вектора x, а значения матрицы A, составляющие 99 % всех значений в алгоритме, используются по одному разу каждое. Поэтому, если представить себе, что в компьютере, выполняющем эти операции, есть медленная большая память на 2000000 значений и быстрая маленькая память на 200 значений, то оптимизировать вычисления второй процедуры за счет использования маленькой памяти теоретически как-то возможно, а вот оптимизировать вычисления первой процедуры никак нельзя.

Если некоторый алгоритм хорошо поддается оптимизации, то для него, скорее всего, можно разработать специализированный процессор, устройство которого будет подобрано оптимально для этого типа алгоритмов. В противном случае никакой специализированный процессор для типа алгоритмов изобрести попросту невозможно. В машинном обучении разные типы нейросетей представляют собой разные типы алгоритмов, соответственно, для некоторых нейросетей можно придумать специализированные процессоры, а для некоторых – нельзя. В действительности в настоящее время есть только два типа специализированных процессоров для нейросетей, соответствующих двум группам нейросетей, для которых оказалось возможным разработать специализированные процессоры.

Тензорный процессор/tensor processor unit/TPU – специализированный процессор, предназначенный для выполнения матричных умножений, чаще всего конкретно вычислений сверточный нейросетей.

Нейроускоритель/нейропроцессор/ускоритель ИИ/ускоритель машинного обучения – научно-популярные термины, обозначающие в целом специализированные ускорители для нейросетей.

Нейроморфный процессор/neuromorphic processor – специализированный процессор, предназначенный для выполнения вычислений (инференса и в некоторых случаях обучения) полносвязных и спайковых нейросетей. Они являются своего рода более сложной версией полносвязных нейросетей и потому с точки зрения оптимизации вычислений в целом обладают схожими свойствами.

Разберемся с тем, какие свойства нейросетей позволили этим классам иметь место.

В полносвязных (и спайковых) нейросетях каждый вес полносвязного слоя используется для единственной операции, в которой он умножается на конкретное входное значение, в результате чего получается конкретное выходное значение (в противоположность, например, сверточным нейросетям, где каждый вес используется многократно). Это позволяет как бы прибить гвоздями все значения весов к определенному месту в памяти и выполнять все вычисления прямого и обратного хода, связанные с конкретным весом, рядом с ним же в памяти. При этом для каждого веса можно выделить свой элемент в памяти, для выполнения каждой вычислительной операции – свое АЛУ, а необходимость в организации большой общей памяти вообще отпадает. Структура процессора в этом случае словно полностью повторяет архитектуру нейросети. На самом деле обычно для описания структуры процессора также используется термин «архитектура», но мы будем здесь избегать его для того, чтобы не путать с архитектурой нейросети. Отпадает также необходимость в сложном устройстве системы команд, достаточно «зашить» в каждое АЛУ действия прямого и обратного хода. Такое чрезвычайно примитивное строение процессора позволяет уместить в нем много весов и соответствующих АЛУ, т. е. приспособить для достаточно больших полносвязных нейросетей. Однако возникает необходимость прокладывания в микросхеме физических связей между местом, где возникает слагаемое при вычислении полносвязного слоя, и местом, где это слагаемое складывается с другими. Если не ограничить количество слагаемых в одной сумме, потребуется проложить в процессоре огромное количество пересекающихся связей, что проблематично. По этой причине число слагаемых ограничивают, а сами веса и связанные с ними АЛУ группируют в ядра процессора, это позволяет ввести некоторый порядок пересылки данных от одной группе другой. С точки зрения строения нейросети, это означает, что ее архитектура состоит из множества полносвязных слоев, расположенных в нескольких параллельных ветвях, между которыми есть связи. Устройство нейроморфного процессора проиллюстрировано на рисунке 3. Весь процессор состоит из решетки ядер, между которыми проложены специальные коммуникационные линии (бежевым цветом), пересылки по ним регулируются специальными элементами (синие крестики). На рисунке каждое ядро реализует полносвязный слой с тремя входными и тремя выходными значениями, т. е. с девятью весами.

Каноническим примером нейроморфного процессора является один из первых процессоров данного класса – IBM True North. Этот процессор представляет собой решетку 64×64 из 4096 ядер, каждое из которых позволяет вычислять до 256 выходных значений одного слоя полносвязной или спайковой нейросети. По-настоящему увлеченные разработчики спайковых нейросетей продолжают отстаивать их аналогию с устройством мозга, в связи с чем каждому весу отводят относительно незначительную роль, грубо говоря, в их представлении каждый вес должен просто определять, будет ли нейрон реагировать на ненулевое входное значение или нет. Отсюда в процессоре True North каждый вес представляется всего-навсего 2-битным значением (для других нейроморфных процессоров также характерно использование очень малых разрядностей). В России разработкой нейроморфорных процессоров занимается только фирма «Мотив Нейроморфные Технологии», которая выпустила прототип своего процессора «Алтай» в 2022 г.

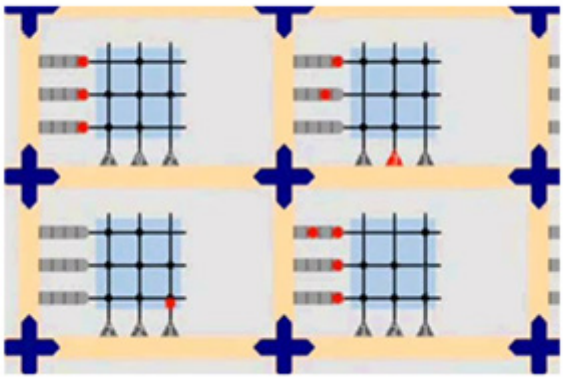

Рисунок 3. Устройство нейроморфного процессора.

Примечательно, что разработкой нейроморфных процессоров занимаются многие крупные разработчики микросхем, но до их практического применения не дошла пока ни одна. Это связано с ограниченностью сферы применимости полносвязных и спайковых нейросетей, в тех же задачах, в которых они все же применимы, их применение пока не оказалось достаточно экономически целесообразным, чтобы мог развиться какой-то существенный спрос, который бы подтолкнул к производству соответствующей продукции. Можно ожидать, что через некоторое время все же массовое производство таких процессоров начнется, но их применение будет ограничено некоторыми прикладными областями. При этом представители разработчиков нейроморфных процессоров и солидарные с ними исследователи пытаются приблизить этот момент, занимаясь поиском прикладных задач, которые могут быть решены с помощью полносвязных нейросетей. В частности, развилось направлений исследований, заключающееся в приведении нейросетей сложных типов (например, сверточных), к полносвязным. Практический потенциал этого подхода ограничен физическими и математическими законами, потому ценность таких исследований минимальна.

Теперь разберемся со сверточными нейросетями. Если рассмотреть вычисления сверточного слоя, можно понять, что они представляют собой совокупность матричных умножений. В примере на рисунке 2, операция умножения матриц легко поддается оптимизации, поскольку в ней все значения используются многократно. При этом умножение матриц всегда можно разбить на совокупность умножений меньших матриц. Наиболее экономной является схема, проиллюстрированная на *рисунке 4*. Из левой матрицы-множителя выбираются N строк (выделены розовым), из правой матрицы – N столбцов (выделены голубым).

Далее итеративно берут подстолбец из значений выделенных строк левой матрицы (выделен красным) и подстроку из значений выделенных столбцов правой матрицы (выделена синим). Каждое взятое значение из левой матрицы умножается на каждое взятое значение из правой, полученные произведения добавляются в качестве слагаемых к соответствующим значениям результирующей матрицы (выделены зеленым), в следующей итерации выбираются следующие подстолбец левой и подстрока правой матриц, и действия повторяются. Когда цикл по всем подстолбцам и подстрокам закончен, квадрат N×N значений результирующей матрицы оказываются подсчитанными полностью. В описанной схеме на каждой итерации выполняется N2 вычислений, а для их выполнения из памяти необходимо взять только 2N значений.

В конце цикла в память выгружаются N2 значений выходной матрицы, но если ширина левой перемножаемой матрицы (и, соответственно, высота правой) достаточно велики, то на фоне цикла вычислений выгрузка посчитанных значений оказывается незначительной.

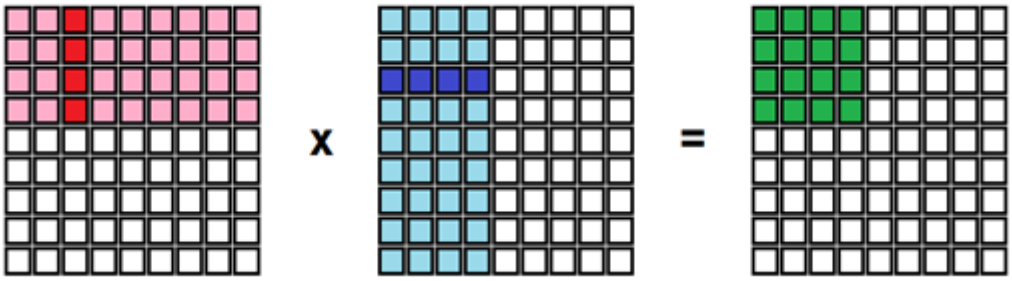

Рисунок 4. Схема матриц.

В описанной схеме чем больше N, тем больше вычислений выполняется с одним значением матрицы одновременно, т. е. после того, как оно один раз взято из памяти. Другими словами, чем больше N, тем меньше раз понадобится взять каждое значение перемножаемых матриц из памяти. Для хранения всех данных для одной итерации нужно только N2 + 4N места в памяти, соответственно, следует выбрать N таким, чтобы значение данного выражения было как можно больше, но меньше объема самой быстрой памяти. Этим будет достигнуто теоретически максимальное сокращение числа обращений в более медленные уровни памяти.

Именно описанный выше принцип эксплуатируется в TPU, но используется «перпендикулярная» схема. Если сопоставить вычисления прямого хода сверточного слоя и схему на рисунке 4, станет ясно, что размеры результирующей матрицы соответствуют числу каналов выходной карты признаков и числу пикселей (либо в строке, либо во всей карте признаков) в ней. Это не очень удобно, так как у разных сверток разные ядра, страйды и паддинги, поэтому в TPU квадрат из N2 элементов выбирается из тензора весов. В одной итерации цикла этот квадратный фрагмент умножается (как матрица) на вектор из N значений одного пикселя входной карты признаков, в результате чего получается вектор-слагаемое, соответствующий N значениям одного пикселя выходной карты признаков в следующей итерации аналогичные действия выполняются для следующего пикселя. Одна такая итерация в TPU реализуется в одно действие с помощью огромной решетки из N2 АЛУ. Эта решетка АЛУ представляет собой главный элемент любого TPU, часто называемый матричным блоком. Поскольку в сверточных нейросетях, кроме сверточных слоев, есть еще другие, в первую очередь ReLU и пулинг, помимо решетки АЛУ, в TPU предусмотрен еще один блок, в противоположность первому часто называемый линейным блоком, который реализует слои, отличные от сверточного. Их вычисления могут быть существенно ускорены только за счет увеличения скорости работы памяти (объяснение к примеру на рисунке 2), что затруднительно. С другой стороны, число операций в этих слоях обычно примерно в 100 раз меньше, чем число операций в сверточных. Поэтому придерживаются схемы, в которой линейный блок перехватывает постепенно поступающие из матричного блока фрагменты пикселей выходной карты признаков и выполняет над ними требуемые операции. Для выполнения пулинга 3×3, например, блоку достаточно иметь собственный буфер памяти чуть более чем на две строки входной карты признаков (*рисунок 5*).

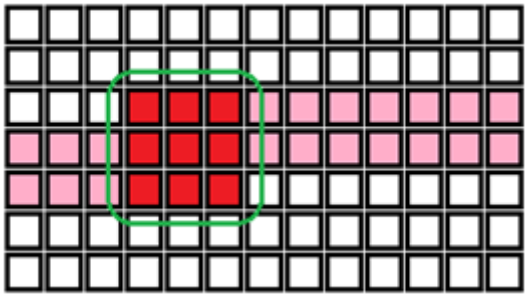

Рисунок 5. Пулинг 3×3.

Поскольку на линейный блок ложится больше разного функционала (как минимум учет паддингов, страйдов, размеров ядра и типа пулинга), чем на матричный, он, как правило, по своему внутреннему устройству сложнее, поэтому у разных разработчиков получается разным. Как правило, линейный блок содержит от N до 9N АЛУ. Устройство разных TPU хоть и может отличаться деталями, принципиально оно у всех одинаковое.

Каноническим примером TPU является первый процессор этого класса, разработанный компанией Google в 2015 г.12F [11], схема его устройства приведена на рисунке 6. Матричный блок Google TPU-v1 состоит из 65536 АЛУ и, как и у большинства других TPU, выполняет вычисления над 8-битными целыми. Процессор работает на частоте 0.7 ГГц и обладает внутренней памятью 28 Мбайт. В России разработкой TPU занимается компаний «IVA Technologies», первые образцы были получены в 2021 г.

В отличие от нейроморфных процессоров, TPU не только разрабатываются многими компаниями, но также активно применяются на практике и как самостоятельные процессоры, и как составные части других процессоров (например, в смартфонах).

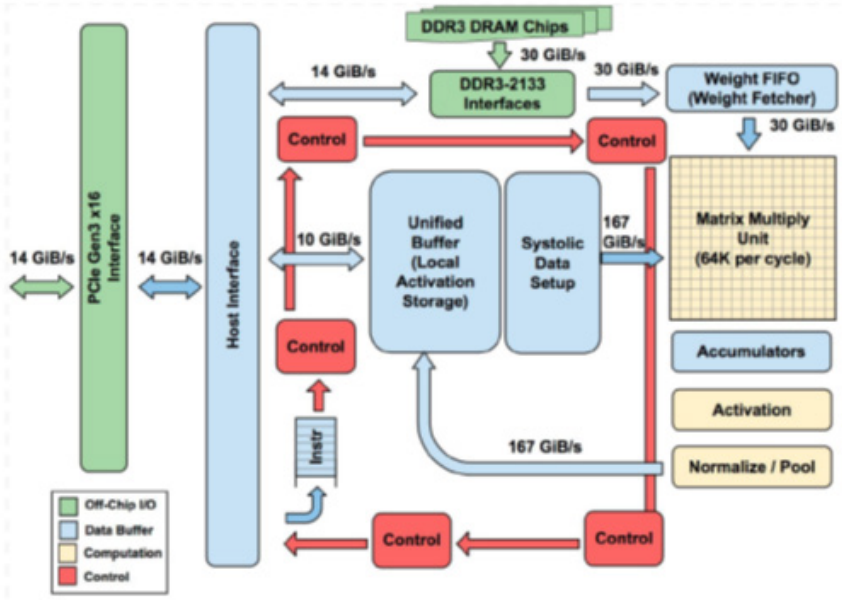

Рисунок 6. Схема устройства первого процессора класса TPU, разработанного компанией Google в 2015 г.

Казалось бы, раздутием размера матричного блока можно достигать все большей и большей оптимизации, однако два фактора не дают это сделать.

Во-первых, чтобы матричный блок был эффективен, его необходимо загрузить соответствующими вычислениями, и если его размеры оказываются больше размеров тензора на конкретном слое, то используется уже не весь блок, соответственно, эффект от его использования оказывается меньше. По этой причине наиболее мощные TPU содержат не одну, а несколько решеток АЛУ.

Внесем ясность в том, что следует понимать под эффективностью процессора. Каждый процессор имеет конечное число АЛУ и известную частоту, на которой они могут работать. Соответственно, перемножив два эти значения можно получить предельное число операций, которое процессор теоретически может выполнить в единицу времени, эта величина называется пиковой производительностью.

С другой стороны, особенности процессора и алгоритма в целом не позволяют загружать все АЛУ процессора в каждый момент времени, часть времени теряется на загрузку данных, посторонние вычисления и т. д., и на практике число операций, выполняемых за единицу времени, оказывается меньше, это число называется реальной производительностью. Для каждого процессора и каждого алгоритма теоретически существует некоторая предельно достижимая реальная производительность, однако в зависимости от навыков программиста в оптимизации может быть достигнуто как 99 % от этого предела, так и 1 %, при чем чаще встречается второе. Соответ-

ственно, когда речь идет об оптимизации, наиболее удобной величиной, дающей представление о том, насколько эффективно реализуются вычисления, является средняя загрузка АЛУ на протяжении рассматриваемого периода, ее и называют эффективностью (что-то вроде КПД). Если из-за размеров сверточного слоя мы можем загрузить, например, не более 25 % матричного блока, это уже означается, что эффективность использования процессора как минимум в четыре раза ниже пиковой.

Во-вторых, чем больше решетка АЛУ, тем больше разница между производительностью матричного и линейного блоков. Напомним, что матричный блок выполняет вычисления, поддающиеся оптимизации, а линейный блок – вычисления, оптимизации не поддающиеся. Представим себе, что самолеты стали летать в 20 раз быстрее, в таком случае большую часть времени любого путешествия будет составлять дорога в аэропорт, и дальнейшее увеличение скорости самолетов уже не будет оказывать существенного влияния на суммарное время. То же самое происходит со сверточными нейросетями. Отсюда возникает даже такой парадокс: нейросеть, состоящая из поканальных сверток и сверток 1×1 и содержащая в 10 раз меньше вычислений, чем другая нейросеть, состоящая из сверток 3×3, может на TPU работать в 10 раз медленнее, чем вторая. По этим причинам TPU на практике никогда не загружены полностью, чаще на 10–20 %.

Эффективность TPU высока на матричных умножениях (сверточных слоях) и низка на остальных операциях, а в целом на нейросети она, как правило, в несколько раз меньше пиковой. Чем больше в нейросети матричных умножений (сверточных слоев) и чем больше размеры тензоров в них, тем выше эффективность на этой нейросети.

Разработчики других классов процессоров для увеличения эффективность своих процессоров на сверточных нейросетях без потери функционала добавляют в структуру процессоров небольшие решетки АЛУ, называемые тензорными ядрами.

Тензорное ядро (в контексте процессора, отличного от TPU) – небольшая решетка АЛУ, подобная матричному блоку TPU. Как правило, характерные размеры решетки не превышают 8, при этом решетка не всегда квадратная.

Впервые тензорные ядра были добавлены в GPU Nvidia поколения Volta, по 1 тензорному ядру из 64 АЛУ на 8 стандартных АЛУ. Таким образом, использование тензорных ядер вместо стандартных номинально позволило увеличить пиковую производительность в восемь раз, однако в силу тех же обстоятельств, что ограничивают реальную производительность TPU, использование тензорных ядер увеличивает скорость вычислений нейросетей не более чем в 1.2–1.4 раза. Тем не менее, непонимание средним обывателем этих свойств и единиц измерения производительности позволяет компании Nvidia бесконечно путать потребителей, жонглируя значениями пиковой и реальной производительностей при разных обстоятельствах, снова и снова заявляя ускорение в десятки раз. Производительность процессоров и содержащих их устройств по умолчанию измеряется во флопсах (от англ. floating point operation per second – FLOPS/s), одним «флопом» считается либо одно умножение, либо одно сложение. К величине применяют стандартные префиксы-множители «гига», «тера» и т. д. В некоторых задачах вычисления выполняются в целых числах (в том числе в нейросетях, для которых выполнено соответствующее квантование), их измеряют также, но из названия величины убирают буквы «фл» (в англ., соответственно, FL). В современных процессорах, как правило, вычисления реализуются через смешанную объединенную операцию умножения со сложением, и, чтобы не забывать постоянно умножать и на два число операций, а заодно не добавлять и убирать буквы «фл», в последнее время стали использовать универсальную единицу исчисления «операция умножения с накоплением», обозначаемую «MAC» (от

англ. multiply accumulate). Таким образом, 1 MAC/s = 2 FLOPS/S, 1 MAC/s = 2 OPS/S, хотя FLOPS и OPS приравнять друг к другу нельзя. Один и тот же процессор может быть способен выполнять разное число операций за единицу времени для разных типов данных, соответственно, у него может быть одновременно несколько показателей производительности. В лучшем случае единицу данных указывают рядом, например «1.45 GFLOPS/s fp32, 2.9 GFLOPS/s fp16». В худшем случае, когда у процессора есть разные режимы для работы с числами одного типа, могут возникать особенно путанные обозначения с каким-нибудь эпитетом, заменяющим тип данных, например «23 trillion MAC AI performance».

Вернемся теперь к программной оптимизации. В силу того, что полносвязные и некоторые похожие на них нейросети не поддаются оптимизации (рисунок 2), никаких существенных программных оптимизаций для них применить нельзя. Что же касается сверточных нейросетей, то в силу их востребованности и оптимизируемости соответствующие программные библиотеки в настоящее время разработаны практически для всех классов процессоров. Именно оптимизированные функции для сверточных нейросетей составляют большую часть программного кода всех библиотек машинного обучения вместе взятых, поскольку больше оптимизировать особо и нечего. В тех же случаях, когда для какого-то процессора такая библиотека еще не разработана или не применима, может надобиться разработать ее самостоятельно, что чрезвычайно трудно для программиста, не имеющего опыта низкоуровневой разработки, но вполне по силу сделать за несколько месяцев (в зависимости от поддерживаемого функционала) человеку с таким опытом.

Разработка такой библиотеки должна быть подчинена следующим простым соображениям:

– в зависимости от требуемого функционала продумывается поддерживаемый библиотекой формат описания нейросетей;
– для выполнения преобразования нейросетей (например, объединения сверточных слоев и ReLU) используются либо функции внешнего фреймворка, либо разрабатываются свои собственные;
– детально изучается устройство целевого процессора;
– реализуются (как правило, преимущественно на ассемблере) оптимальные функции вычисления сверточных слоев, использующие, насколько возможно, трюк, приведенный в формуле на рисунке 4, с учетом иерархии памяти, конфигураций АЛУ и других свойств процессора для разных параметров сверточных слоев;
– реализуются насколько возможно оптимальные функции для остальных слоев, помимо сверточных.
– Что касается программной оптимизации нейросетей в общем случае, то по сравнению с оптимизацией математических алгоритмов в целом она обладает лишь несколькими особенностями:
– параллельное выполнение вычислений для нескольких единиц входных данных в рамках батча позволяет в некоторых случаях более плотно загрузить АЛУ и, таким образом, повысить эффективность;
– на инференсе веса становятся фиксированными значениями, что позволяет размещать их в памяти перед выполнением вычислений заранее удобным образом;
– вычисления прямого и обратного хода используют общие данные, в связи с чем при использовании нескольких параллельно работающих вычислительных устройств во время обучения необходимо продумать распределение между ними соответствующих вычислений и наборов весов и учесть возникающие при этом обмены данными (в особенности в процессе прибавления градиента к весам);
– характерное для большинства нейросетей большое количество разных математических операций в некоторых случаях позволяет перегруппировать эти операции так, чтобы стали возможными хотя бы некоторые программные оптимизации.

Можно ожидать, что в будущем появятся новые типы нейросетей, для которых станет возможна разработка новых классов специализированных процессоров. Однако на данный момент, как видно, за вычетом оптимизации сверточных слоев, тема оптимизации вычислений в машинном обучении достаточно бедна. С другой стороны, именно широкое распространение доступных и мощных вычислительных устройств явилось предпосылкой развития машинного обучения. При чем основную массу разработчиков нейросетей составляют математиками, а не низкоуровневые программистами, способные писать свои библиотеки на ассемблере. Вычислительные средства нужны для инференса в конечных устройствах, для обучения нейросетей на серверах и для разработки на персональных компьютерах. Таким образом, потребность человечества в использовании машинного обучения диктует потребность в вычислительных средствах, которые были бы одновременно доступными в больших количествах для любого кошелька, достаточно мощными и легко осваиваемыми (чтобы программист, который будет их использовать,

мог относительно небольшими стараниями достигать удовлетворительного уровня эффективности).

Наилучший баланс между этими характеристиками обеспечивают GPU. Это объясняется тем, что GPU базируются на принципе так называемой массовой мультипоточности, использование которого на порядки упрощает оптимизацию программ.

Одновременно с этим среди производителей GPU доминирующую позицию занимает компания Nvidia: она первая разработала и стала активно продвигать удобный язык программирования CUDA для GPU, который стал де-факто стандартом параллельного программирования среди программистов всего мира. В России разработкой GPU, программируемого на CUDA, занимается только компания «Байкал Электроникс», выход первого GPU ожидается в 2026 г.

Список литературы:

[1] Chen Zhang, Joohee Kim Video object detection with two-path convolutional LSTM pyramid. 2016. – URL: https://www.researchgate.net/publication /343702579_Video_Object_Detection_With_ Two-Path_ Convolutional_LSTM_ Pyramid/fulltext/5f3b2658928 51cd302013482/Video-Object-Detection-With-Two-Path-Convolutional-LSTM-Pyramid.pdf/ (дата обращения: 16.01.2026).

[2] Hanson A., Koutilya PNVR, Sanjukta Krishnagopal, Larry Davis Bidirectional Convolutional LSTM for the Detection of Violence in Videos. 2018. – URL: https://openaccess.thecvf.com/content_eccv_2018_workshops/w10 /html/Hanson_Bidirectional_Convolutional_LSTM_or_the_ Detection_of_Violence_in_ Videos_ ECCVW_2018_paper.html (дата обращения: 15.01.2026).

[3] Howard A.G., Menglong Zhu, Bo Chen, Kalenichenko D. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. – URL: https://arxiv.org/pdf/1704.04861/ (дата обращения: 11.01.2026).

[4] Iandola F.N., Han S., Moskewicz M.W., Ashraf K., Dally W.J., Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size Recognition. 2016. Published as a conference paper at ICLR. 2015. – URL: https://www. researchgate.net/publication/ 301878495_SqueezeNet_AlexNet-level_accuracy_ with_50x_fewer_ parameters_ and_05MB_model_size (дата обращения: 10.01.2026).

[5] Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. – URL: https://arxiv.org/pdf/1502.03167/ (дата обращения: 13.01.2026).

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Deep Residual Learning for Image Recognition. 2015. – URL: https://arxiv.org/pdf/1512.03385 (дата обращения: 11.01.2026).

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Spatial pyramid pooling in deep convolutional networks for visual recognition. 2015. – URL: https://arxiv.org/pdf/1406.4729/ (дата обращения: 11.01.2026).

[8] Krizhevsky I., Sutskever I., Hinton G.E. Imagenet classification with deep convolutional neural networks. 2012. – URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/
c399862d3b9d6b76c8436e 924a68c45b-Paper.pdf (дата обращения: 12.01.2026).

[9] Lavin A., Scott Gray Fast Algorithms for Convolutional Neural Networks. 2015. – URL: https://arxiv.org/abs/1509.09308 (дата обращения: 11.01.2026).

[10] Mikolov T., Kai Chen, Corrado G., Jeffrey Dean Efficient Estimation of Word Representations in Vector Space. 2013. – URL: https://arxiv.org/abs/1301.3781 (дата обращения: 15.01.2026).

[11] Norman P. Jouppi, Cliff Young, Nishant Patil and others In-Datacenter Performance Analysis of a Tensor Processing Unit. 2017. – URL: https://arxiv.org/pdf/1704.04760 (дата обращения: 11.01.2026).

[12] Ronneberger O., Fischer Ph., Brox Th. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. – URL: https://arxiv.org/pdf/1505.04597/ (дата обращения: 13.01.2026).

[13] Szegedy C., Wei Liu, Chapel Hill, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich Going Deeper with Convolutions. 2014. – URL: https://arxiv.org/pdf/1409.4842 (дата обращения: 12.01.2026)

[14] Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015. Published as a conference paper at ICLR 2015. – URL: https://arxiv.org/pdf/1409.1556 (дата обращения: 10.01.2026).

[15] Wei Liu, Anguelov D., Erhan D., Szegedy C., Reed S., Cheng-Yang Fu, Berg A.C. SSD: Single Shot MultiBox Detector. 2016. – URL: https://arxiv.org/pdf/1512.02325 (дата обращения: 11.01.2026).

[16] Официальный сайт ONNX. – URL: https://onnx.ai/.

[17] Основы технологий искусственного интеллекта: учебное пособие/ А.А. Семенов, В.В. Гончар, А.С. Эрдниев. – Московский университет МВД России имени В.Я, Кикотя, 2025. – 190с.

[18] Финансовый мониторинг: учебное пособие / В.И. Глотов, А.У. Альбеков, О.Н. Тисен и др. – КноРус, 2022. – 198 с.

[19] Тисен О.Н. Противодействие современным приемам и способам вербовки в международные террористические организации // Российская юстиция. 2018, №9. С. 51-54.

[20] Гончар В.В. О важности формирования единообразного понятийного аппарата необходи-

мого для расследования преступлений в сфере компьютерной информации// Вестник Экономической безопасности. 2018. №1. С. 225-230.

[21] Орлова А.А., Гончар В.В. Особенности расследования преступлений, совершаемых в сфере информационных технологий // Безопасность бизнеса. 2021. № 4. С. 25-29.

References:

[1] Chen Zhang, Joohee Kim Video object detection with two-path convolutional LSTM pyramid. 2016. – URL: https://www.researchgate.net/publication /343702579_Video_Object_Detection_With_ Two-Path_ Convolutional_LSTM_ Pyramid/fulltext/5f3b2658928 51cd302013482/Video-Object-Detection-With-Two-Path-Convolutional-LSTM-Pyramid.pdf/ (дата обращения: 16.01.2026).

[2] Hanson A., Koutilya PNVR, Sanjukta Krishnagopal, Larry Davis Bidirectional Convolutional LSTM for the Detection of Violence in Videos. 2018. – URL: https://openaccess.thecvf.com/content_eccv_2018_workshops/w10 /html/Hanson_Bidirectional_Convolutional_LSTM_or_the_ Detection_of_Violence_in_ Videos_ ECCVW_2018_paper.html (дата обращения: 15.01.2026).

[3] Howard A.G., Menglong Zhu, Bo Chen, Kalenichenko D. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. - URL: https://arxiv.org/pdf/1704.04861/ (access date: 11.01.2026).

[4] Iandola F.N., Han S., Moskewicz M.W., Ashraf K., Dally W.J., Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size Recognition. 2016. Published as a conference paper at ICLR. 2015. – URL: https://www. researchgate.net/publication/ 301878495_SqueezeNet_AlexNet-level_accuracy_ with_50x_fewer_ parameters_ and_05MB_model_size (дата обращения: 10.01.2026).

[5] Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. – URL: https://arxiv.org/pdf/1502.03167/ (access date: 13.01.2026).

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Deep Residual Learning for Image Recognition. 2015. – URL: https://arxiv.org/pdf/1512.03385 (access date: 11.01.2026).

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Spatial pyramid pooling in deep convolutional networks for visual recognition. 2015. - URL: https://arxiv.org/pdf/1406.4729/ (access date: 11.01.2026).

[8] Krizhevsky I., Sutskever I., Hinton G.E. Imagenet classification with deep convolutional neural networks. 2012. - URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e 924a68c45b-Paper.pdf (accessed on: 12.01.2026).

[9] Lavin A., Scott Gray Fast Algorithms for Convolutional Neural Networks. 2015. - URL: https://arxiv.org/abs/1509.09308 (access date: 11.01.2026).

[10] Mikolov T., Kai Chen, Corrado G., Jeffrey Dean Efficient Estimation of Word Representations in Vector Space. 2013. – URL: https://arxiv.org/abs/1301.3781 (access date: 15.01.2026).

[11] Norman P. Jouppi, Cliff Young, Nishant Patil and others In-Datacenter Performance Analysis of a Tensor Processing Unit. 2017. – URL: https://arxiv.org/pdf/1704.04760 (access date: 11.01.2026).

[12] Ronneberger O., Fischer Ph., Brox Th. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. – URL: https://arxiv.org/pdf/1505.04597/ (access date: 13.01.2026).

[13] Szegedy C., Wei Liu, Chapel Hill, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich Going Deeper with Convolutions. 2014. – URL: https://arxiv.org/pdf/1409.4842 (accessed on: 12.01.2026)

[14] Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015. Published as a conference paper at ICLR 2015. - URL: https://arxiv.org/pdf/1409.1556 (access date: 10.01.2026).

[15] Wei Liu, Anguelov D., Erhan D., Szegedy C., Reed S., Cheng-Yang Fu, Berg A.C. SSD: Single Shot MultiBox Detector. 2016. – URL: https://arxiv.org/pdf/1512.02325 (access date: 11.01.2026).

[16] ONNX official website. – URL: https://onnx.ai/.

[17] Fundamentals of artificial intelligence technologies: a textbook/A.A. Semenov, V.V. Gonchar, A.S. Erdniev. - Moscow University of the Ministry of Internal Affairs of Russia named after V.Ya, Kikotya, 2025. - 190s.

[18] Financial monitoring: training manual/V.I. Glotov, A.U. Albekov, O.N. Tisen et al. - KnoRus, 2022. - 198 s.

[19] Tisen O.N. Countering modern methods and methods of recruitment to international terrorist organizations//Russian Justice. 2018, №9. S. 51-54.

[20] Gonchar V.V. On the importance of forming a uniform conceptual apparatus necessary for investigating crimes in the field of computer information// Bulletin of Economic Security. 2018. №1. S. 225-230.

[21] Orlova A.A., Gonchar V.V. Features of the investigation of crimes committed in the field of information technology//Business security. 2021. № 4. S. 25-29.