УДК 004.85

Научная статья

https://doi.org/10.32603/2071-8985-2026-19-5-16-24

Особенности реализации потоковых рекуррентных нейронных сетей на графических процессорах

В. М. Тайц

Санкт-Петербургский федеральный исследовательский центр РАН,
Санкт-Петербург, Россия
taizvadim@gmail.com

Аннотация. Актуальность задачи оптимизации обработки потоков данных рекуррентными нейронными сетями обусловлена ростом объемов многомерных временных рядов в сложных динамических системах, где традиционные реализации не обеспечивают требуемой скорости прогноза в реальном времени и способности к непрерывному обучению без остановки системы. Целью исследований служит расширение возможностей обработки потоков многомерных временных рядов рекуррентными нейронными сетями (РНС) с управляемыми элементами за счет их эффективной реализации на графических процессорах (GPU). Предложен алгоритм такой реализации с учетом специфики данных сетей. В качестве материала исследований использовался сформированный датасет, содержащий 1000 элементов многомерных временных рядов, на котором обучались модели всех рассматриваемых архитектур; в качестве испытательной процедуры принят один цикл прогнозирования, включающий обучение на всей выборке и вычисление прогноза с горизонтом 72 такта. Проведены эксперименты по сравнению времени цикла обработки данных в РНС при реализациях на CPU- и GPU-платформах. Для каждого эксперимента использовались нейросеть с идентичными параметрами была реализована на перечисленных архитектурах. Было проведено множество экспериментов с РНС разного размера для оценки масштабируемости предложенной GPU-архитектуры. Результаты экспериментов показывают, что для малоразмерных РНС (около 650 нейронов в каждом слое) выигрыш в производительности предложенной GPU-архитектуры составляет порядка 10 раз по сравнению с CPU-реализацией, тогда как с увеличением размера сети ускорение нелинейно растет и для крупномасштабных конфигураций достигает 90 раз. Таким образом, реализация подобных РНС на GPU-платформе позволяет существенно расширить применимость этих сетей для решения задач прогнозирования временных рядов с непрерывным обучением.

Ключевые слова: рекуррентная нейронная сеть, графические процессоры, алгоритмы реализации, производительность

Для цитирования: Тайц В. М. Особенности реализации потоковых рекуррентных нейронных сетей на графических процессорах // Изв. СПбГЭТУ «ЛЭТИ». 2026. Т. 19, № 5. С. 16–24. doi: 10.32603/2071-8985-2026-19-5-16-24.

Original article

Features of the Implementation of Streaming Recurrent Neural Networks on Graphical Processing Units

V. M. Taits

Saint Petersburg Electrotechnical University, Saint Petersburg, Russia
taizvadim@gmail.com

Abstract. The relevance of the problem of optimizing data stream processing with recurrent neural networks (RNNs) is driven by the growing volumes of multivariate time series in complex dynamic systems, where traditional implementations fail to provide the required real-time prediction speed and the ability for continuous

learning without system interruption. The aim of this work is to expand the capabilities of processing multivariate time series streams with recurrent neural networks (RNNs) with controlled elements through their efficient implementation on graphics processing units (GPUs). The paper proposes an algorithm for such an implementation, taking into account the specific features of these networks. As the research material, a constructed dataset containing 1,000 elements of multivariate time series was used, on which models of all considered architectures were trained; the test procedure adopted is one prediction cycle, including training on the entire sample and computing a forecast with a horizon of 72 time steps. Experiments comparing the data processing cycle time in RNNs for CPU- and GPU-based implementations were conducted. For each experiment, a neural network with identical parameters was implemented on the aforementioned architectures. A series of experiments with RNNs of different sizes was carried out to evaluate the scalability of the proposed GPU architecture. The experimental results show that for small-scale RNNs (approximately 650 neurons in each layer), the performance gain of the proposed GPU architecture is about 10 times compared to the CPU implementation, whereas, with increasing network size, the acceleration grows nonlinearly and reaches 90 times for large-scale configurations. Thus, implementing such RNNs on a GPU platform significantly expands the applicability of these networks for solving time series forecasting problems with continuous learning.

Keywords: recurrent neural network, graphics processors, implementation algorithms, performance

For citation: Taits V. M. Features of the Implementation of Streaming Recurrent Neural Networks on Graphical Processing Units // LETI Transactions on Electrical Engineering & Computer Science. 2026. Vol. 19, no. 5. P. 16–24. doi: 10.32603/2071-8985-2026-19-5-16-24.

Введение. В последние годы искусственные нейронные сети (ИНС) активно развиваются, что связано с их способностью эффективно решать задачи машинного обучения, обработки больших данных и автоматизации сложных процессов [1], [2]. Особое внимание уделяется архитектурам, обладающим расширенными возможностями, с трансферным и непрерывным обучением [3], [4]. Такие подходы позволяют модели адаптироваться к новым задачам и данным без необходимости полного переобучения, что особенно важно для динамических и изменяющихся сред.

Несмотря на значительный прогресс в области архитектур ИНС, многие из них изначально реализуются и тестируются на центральных процессорах (CPU), что обеспечивает гибкость и универсальность, но ограничивает производительность при работе с большими объемами данных и сложными вычислениями. В то же время, графические процессоры широко применяются для значительного ускорения вычислений в искусственных нейронных сетях, прежде всего за счет эффективной реализации матричных умножений, составляющих основу их работы [5].

В данной статье рассматривается архитектура рекуррентной нейронной сети (РНС) с управляющими элементами [6], [7], обладающая преимуществами трансферного и непрерывного обучения, однако до настоящего времени не реализованная на GPU-платформе. Основное внимание уделяется процессу адаптации этой архитектуры для эффективной работы на GPU, анализу возникших технических и алгоритмических сложностей, а также сравнению производительности CPU- и GPU-реализаций. Полученные результаты демонстрируют потенциал использования GPU для ускорения работы данной архитектуры и расширяют возможности ее применения в задачах обработки и анализа данных в реальном времени.

Общая характеристика потоковых РНС и постановка задачи. Рассмотрим двухслойную рекуррентную нейронную сеть с управляемыми синаптическими связями, в которую подаются сигналы, предварительно разложенные на пространственно-частотные составляющие. Каждая из этих составляющих преобразуется в последовательность единичных образов (СЕО), частота следования которых определяется амплитудой соответствующей компоненты. Оба слоя сети идентичны и содержат по N нейронов, причем каждый нейрон связан со всеми нейронами противоположного слоя, но не с нейронами своего слоя. Передача единичных образов между слоями осуществляется с управляемыми пространственными сдвигами, что позволяет разбивать слои на логические поля и организовывать продвижение совокупностей СЕО вдоль слоев по определенной схеме. Пример спиральной структуры слоев РНС приведен на рис. 1.

В [6]–[8] показано, что данная архитектура демонстрирует высокую эффективность в решении творческих задач на основе ассоциаций, особенно при использовании спиральной схемы с однонаправленным или встречным продвижением еди-

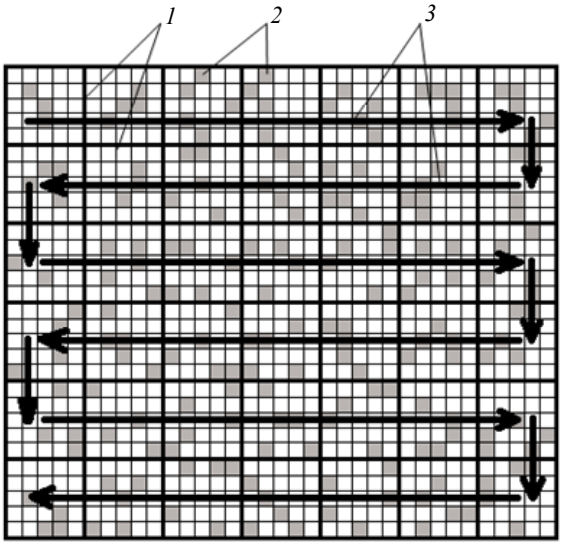

Рис. 1. Спиральная структура первого слоя РНС, идентичная структуре второго слоя: *1* – линии разбивки слоев на логические поля за счет реализуемых пространственных сдвигов СЕО при передаче от слоя к слою; *2* – возбужденные нейроны; *3* – направления продвижения СЕО вдоль слоев

Fig. 1. The spiral structure of the first layer of the RNS, identical to the structure of the second layer: *1* – lines of division of layers into logical fields due to the realized spatial shifts of the SSP during transmission from layer to layer; *2* – excited neurons; *3* – directions of SSP advancement along layers

ничных образов. Однако основным недостатком такой реализации служит низкая производительность, обусловленная выполнением вычислений на CPU. Это существенно ограничивает возможности масштабирования и практическое применение сети для задач с большими объемами данных.

В связи с этим актуальна разработка архитектуры РНС, адаптированной для реализации на GPU, а это позволяет существенно повысить производительность обработки и обеспечить эффективную работу сети при больших размерах слоев. Предлагаемая архитектура должна учитывать особенности параллельной обработки данных на GPU, а также сохранять ключевые ассоциативные свойства исходной схемы. Экспериментальное подтверждение эффективности такой реализации заключается в сравнении времени обработки и качества ассоциативных связей при переходе с CPU на GPU, что позволит оценить преимущества предложенного подхода для дальнейшего применения в реальных задачах.

Требуется найти целесообразный способ реализации S_0 РНС на графических процессорах, при котором будет достигаться минимум времени цикла обработки входной информации $T_0\left(S_0\right) = \min T_i\left(S_i\right)$.

При ограничениях: $V_i\left(S_i\right) \le V_{\text{зад}}$; $F_i\left(S_i\right) = F_{\text{зад}}$; $M_i\left(S_i\right) \le M_{\text{зад}}$, где $i\{1, 2, \ldots, K\}$ – номер рассматриваемого варианта реализации (разные стратегии распараллеливания, разные размеры планок/блоков, разные форматы хранения синапсов); K – количество технологически возможных и ресурсно допустимых вариантов реализации; S_i – i-й вариант реализации РНС на GPU; S_0 – оптимальный вариант реализации; $T_i(S_i)$ – время полного цикла обработки входного сигнала (такта работы РНС) для i-го варианта; $T_0(S_0)$ – оптимальное время полного цикла обработки входного сигнала; $V_{\text{зад}}$ – время полного цикла обработки входного сигнала при реализации на CPU; $V_i(S_i)$ – время полного цикла обработки входного сигнала i-го варианта реализации на GPU; $F_{\text{зад}}$ – результат работы при реализации на CPU; $F_i(S_i)$ – результат работы i-го варианта реализации на GPU; $M_{\text{зад}}$ – память, потребляемая программой при реализации на CPU; $M_i\ (S_i)$ – память, потребляемая программой, реализующей i-й вариант на GPU.

Особенности GPU-процессоров. GPU представляет собой массивно-параллельный многоядерный процессор, содержащий тысячи вычислительных ядер, оптимизированных для одновременной обработки множества потоков данных в отличие от последовательной архитектуры CPU [9]. Это позволяет вместо одного потока данных разбить вычисления на множество потоков. Однако существует ряд ограничений при реализации алгоритмов на GPU, которые нужно преодолеть для успешной работы алгоритма:

– каждый вычислительный модуль GPU исполняет один поток, однако синхронизация блоков возможна только на уровне CPU-программы;

– данные должны копироваться между CPU- и GPU-памятью, что требует дополнительных операций и может стать узким местом;

– не все типы данных и операции, доступные на CPU, поддерживаются на GPU. Например, нет прямой поддержки булевых переменных.

Архитектуры с широкой одноинструкционной многопоточной архитектурой (SIMT) часто требуют статического распределения групп потоков, которые выполняются синхронно на протяжении всего ядра приложения. Например, если код содержит условный оператор и хотя бы один поток

группы зашел в ветку его выполнения, все остальные, которые в нее не зашли, все равно вынуждены дожидаться момента, когда активные потоки завершат выполнение этого блока. Только после синхронного окончания всех ветвей группа продолжает выполнение дальше. Хотя для максимизации эффективности процессора во время операций ветвления используются алгоритмы сходимости, приложения, требующие сложного потока управления, часто приводят к низкой эффективности процессора из-за большой длины и количества путей управления [10]. Поэтому при проектировании алгоритмов на GPU важно стремиться к тому, чтобы потоки внутри одной группы выполняли одну и ту же последовательность инструкций.

Адаптация под GPU. Благодаря принятой организации слоев РНС и передачи сигналов между ними [6] становится возможным параллельно вычислять новые состояния нейронов принимающего слоя на каждом этапе обработки. Это достигается за счет независимости вычислений для каждого нейрона – суммарный входной потенциал для каждого нейрона формируется только на основе текущих состояний нейронов предыдущего слоя и соответствующих весов синапсов, без необходимости ожидания результатов других нейронов. Таким образом, все нейроны слоя могут обрабатываться одновременно, что особенно эффективно при реализации на GPU.

Кроме того, параллельная обработка позволяет одновременно пересчитывать веса синапсов между слоями. Для каждого синапса, соединяющего возбужденный нейрон передающего слоя с нейроном принимающего слоя, вес корректируется в зависимости от текущего состояния принимающего нейрона (ожидание, возбуждение, временная невосприимчивость (рефракторность)).

Рассмотрим обобщенную схему передачи сигнала от первого слоя ко второму с оптимизацией под GPU:

Шаг 1. Параллельный пересчет состояний нейронов на GPU. Для каждого нейрона второго (приемного) слоя параллельно вычисляется суммарный входной потенциал, поступающий от возбужденных нейронов предыдущего (передающего) слоя с учетом весов синапсов и пространственных сдвигов единичных образов. Если суммарный потенциал превышает порог, нейрон переводится в состояние возбуждения и формирует единичный образ на один такт. В случае рефрактерности состояние нейрона обнуляется.

Шаг 2. Параллельный пересчет весов синапсов на GPU. Для каждого нейрона приемного слоя и каждого возбужденного нейрона передающего слоя параллельно пересчитываются веса соответствующих синапсов. Если принимающий нейрон находится в рефрактерности, вес уменьшается на заданное значение. Если принимающий нейрон возбужден, вес увеличивается на заданное значение. Все изменения весов синапсов происходят одновременно для всех пар нейронов.

Шаг 3. Обработка состояния возбуждения и рефрактерности для каждого нейрона передающего слоя. Для каждого нейрона передающего слоя, если нейрон возбужден, состояние возбуждения снимается, и нейрон переходит в состояние невосприимчивости (рефрактерности), которое длится заданное количество тактов. После окончания рефрактерного периода нейрон возвращается в обычное (восприимчивое) состояние.

Рассмотрим далее более детально особенности реализации шагов 1, 2.

Алгоритм пересчета состояний нейронов при передаче сигналов. Этот алгоритм реализует параллельную обработку нейронов слоя, вычисляя суммарный входной потенциал для каждого нейрона с учетом возбуждения входящих нейронов, весов синапсов и пространственных сдвигов единичных образов. Если суммарный потенциал превышает порог, нейрон возбуждается. Формируется единичный образ. Затем начинается период временной невосприимчивости.

Далее представлено пошаговое описание алгоритма, детализирующее последовательность операций параллельной обработки нейронов слоя:

Шаг 1. $i = 0$. В рамках параллельной обработки каждый вычислительный поток, соответствующий одному нейрону, обрабатывает состояние i-го нейрона k-го слоя. Начинается анализ воздействия единичных образов на i-й нейрон k-го слоя.

Шаг 2. Если номер нейрона i превышает размер слоя N, алгоритм заканчивает работу.

Шаг 3. $j = 0$. Если i-й нейрон находится в состоянии невосприимчивости (рефрактерности), обработка переходит к следующему нейрону.

Шаг 4. $j = j + 1$. Начинается анализ воздействия на i-й нейрон k-го слоя со стороны j-х нейронов другого слоя через ji-й синапс.

Шаг 5. Если номер нейрона-источника j превышает размер слоя N, обработка переходит к шагу 8.

Шаг 6. Если j-й нейрон не возбужден, обработка переходит к следующему нейрону-источнику.

Шаг 7. Определяется вес *ji*-го синапса и соответствующий потенциал, поступающий на вход *i*-го нейрона *k*-го слоя с учетом пространственных сдвигов СЕО, обеспечивающих формирование требуемой спиральной структуры слоев сети. Прибавление этого потенциала к уже имеющемуся потенциалу на входе нейрона. Переход к следующему нейрону-источнику.

Шаг 8. Если суммарный потенциал на входе *i*-го нейрона *k*-го слоя превышает заданный порог, этот нейрон переводится в состояние возбуждения и на его выходе формируется единичный образ длительностью в один такт. Далее осуществляется переход к следующему нейрону.

В данном алгоритме веса прямых синапсов между нейронами определяются следующим образом:

$$\omega_{ij}(t) = k_{ij}(t)\ \beta\left(r_{ij}(t)\right),$$

где $\beta(r_{ij})$ – функция ослабления, зависящая от расстояния r_{ij} между соединенными через синапс нейронами на плоскости (X, Y). При малых расстояниях между слоями функция $\beta(r_{ij})$ может быть задана следующим образом [6]:

$$\beta\left(r_{ij}\right) = \frac{1}{1+\alpha\left(r_{ij}\right)^{1/h}};\ i = 1,\ 2, \ldots, N;\ j = 1,\ 2, \ldots, N,$$

где α и h – положительные постоянные коэффициенты; N – число нейронов в каждом слое РНС.

Расстояние r_{ij} между i-м и j-м нейронами, выраженное в единицах нейронов, определяется с учетом пространственных сдвигов СЕО вдоль слоев [6]:

$$r_{ij} = \sqrt{\left(\Delta x_{ij} + n_{ij}d\right)^2 + \left(\Delta y_{ij} + m_{ij}q\right)^2};$$
$$n_{ij} = \pm 0, 1, \ldots, L-1;\ m_{ij} = \pm 0, 1, \ldots, M-1,$$

где Δx_{ij} и Δy_{ij} обозначают проекции соединения между j-м и i-м нейронами на оси X и Y без учета пространственных сдвигов. Параметры d и q характеризуют сдвиги по соответствующим координатам. Числа L и M определяют количество столбцов и строк, на которые разбивается каждый слой сети благодаря сдвигам. Площадь рабочего поля каждого слоя, равная числу нейронов, попадающих в это поле, вычисляется как произведение dq.

Весовые коэффициенты: $k_{ij} = \dfrac{1-e^{-\gamma g_{ij}(t)}}{1+e^{-\gamma g_{ij}(t)}}$ синаптических связей $w_{ij}(t)$ зависят от γ – постоянного положительного коэффициента и $g_{ij}(t)$ – количества запомненных единичных образов на этом синапсе, которые могут быть заданы как [6]

$$g_{ij}(t) = g_{ij}(t-1) + b_{ij}(t),$$

где $g_{ij}(t-1)$ – количество единичных образов, сохраненных на синапсе в предыдущий временной такт; $b_{ij}(t)$ – влияние от единичного взаимодействия нейронов в момент времени t.

Алгоритм пересчета весов синапсов. Алгоритм реализует параллельный пересчет весов синапсов для каждого нейрона слоя. Для каждого нейрона-источника, находящегося в возбужденном состоянии, вес синапса, соединяющего его с текущим нейроном, увеличивается, если текущий нейрон также возбужден, либо уменьшается, если текущий нейрон находится в рефрактерном состоянии. Обработка всех синапсов происходит одновременно для всех нейронов слоя.

Ниже представлено пошаговое описание алгоритма пересчета весов синапсов, детализирующее параллельный процесс обновления связей на основе состояний возбуждения и рефрактерности:

Шаг 1. $i = 0$. В рамках параллельной обработки каждый поток обрабатывает состояние одного нейрона основного слоя. Начинается обработка для i-го нейрона.

Шаг 2. Если номер нейрона i превышает размер слоя N, алгоритм заканчивает работу.

Шаг 3. $j = 0$. Начинается перебор всех нейронов-источников в слое.

Шаг 4. $j = j + 1$. Начинается анализ воздействия на i-й нейрон k-го слоя со стороны j-го нейрона другого слоя через ji-й синапс.

Шаг 5. Если номер нейрона-источника j превышает размер слоя N, обработка переходит к следующему нейрону основного слоя.

Шаг 6. Если j-й нейрон-источник не возбужден, обработка переходит к шагу п. 4.

Шаг 7. Если i-й нейрон находится в состоянии рефрактерности, вес ji-го синапса уменьшается, обработка переходит к шагу 4.

Шаг 8. Если i-й нейрон возбужден, вес ji-го синапса увеличивается, обработка переходит к шагу 4.

Программная реализация алгоритмов. Здесь представлена реализация части алгоритмов предложенной модели рекуррентной нейросети на GPU-архитектуре. Условные обозначения:

int – целое число;
boolean – булева переменная;
double – вещественные числа;
[] – массив;
[][] – матрица;
void – функция, не возвращающая значение.

Состояние нейронов и весов синапсов предлагается хранить в нескольких массивах для удобства передачи между CPU и GPU:

neurons_1: boolean[] – массив состояний возбужденности нейронов; true означает, что нейрон возбужден, false – что нет;
neurons_2: boolean[] – аналогично предыдущему массиву;
refract_intervals_1: int[] – массив состояний рефрактерности нейронов первого слоя; 0 означает, что нейрон на данный момент не находится в состоянии рефрактерности, *n* – что нейрон будет продолжать находиться в состоянии невосприимчивости *n* тактов;
refract_intervals_2: int[] – аналогично предыдущему массиву;
synapses_1_to_2: double[][] – матрица весов синапсов от первого слоя ко второму слою. Строки матрицы соответствуют нейронам первого слоя, столбцы – нейронам второго слоя. Чем больше значение параметра ячейки, тем сильнее корреляция между нейронами;
synapses_2_to_1: double[][] – аналогично предыдущей матрице.

Также в коде используются следующие константы:

step: doube – шаг, на который увеличивается или уменьшается вес синапса;
g_0, gamma: double – числа из формул.
get_spatial_distance – псевдофункция определения расстояния между нейронами (эвклидово расстояние с учетом пространственного сдвига).
size – псевдофункция нахождения длины массива.

Представим реализацию функции нахождения пространственного коэффициента между нейронами разных слоев:

```
double get_weight_coefficient(
    input_neuron_index: int,
    output_neuron_index: int,
) {
    double spatial_distance = get_spatial_distance
(input_neuron_index, output_neuron_index);
    float shifted_spatial_distance = spatial_distance
– g_0;
    return 1.0 – exp(-gamma_inc * shifted_
spatial_distance);
}
```

Input_neuron_index – индекс нейрона слоя, передающего сигнал; output_neuron_index – индекс нейрона, принимающего сигнал.

Функция пересчета состояний нейронов на GPU:

```
void recount_neuron_states(
    // веса синапсов из передающего слоя в
принимающий, доступно только для чтения
    input_synapses: double[][],
    // состояние нейронов передающего слоя,
доступно только для чтения
    input_neurons: boolean[],
    // состояние нейронов принимающего слоя,
доступно только для записи
    output_neurons: boolean[],
    // состояния рефрактерности нейронов
принимающего слоя, доступно только для чтения
    output_refract_intervals: int[],
    // порог, при преодолении которого
суммарным потенциалом на входе нейрон
возбуждается
    threshold: double,
) {
    // функция GPU, получающая индекс
нейрона
    int output_neuron_index =
get_neuron_index();

    //  нейрон в состоянии невосприимчивости
не может быть возбужден

    if (output_refract_intervals[neuron_index] > 0) {
        output_neurons[neuron_index] = 0;
        return;
    }

    // вычисление суммарного потенциала на
входе
    double sum = 0;
    for (int input_neuron_index = 0; input_neuron_
index < size(output_neurons); input_neuron_index
+= 1) {
    sum += get_weight_coefficient(input_neuron_
index, output_neuron_index) * input_synapses[input_
neuron_index, output_neuron_index];
    }
```

```
        // сравнение суммарного потенциала на входе с порогом
        if (sum > threshold) {
            output_neurons[output_neuron_index] = 1;
        } else {
            output_neurons[output_neuron_index] = 0;
        }
    }
```

Функция пересчета весов синапсов на GPU:

```
    void recount_synapses(
        // веса синапсов из передающего слоя в принимающий, доступно для чтения и записи
        input_synapses: double[][],
        // состояние нейронов передающего слоя, доступно только для чтения
        input_neurons: boolean[],
        // состояние нейронов принимающего слоя после пересчета, доступно только для чтения
        output_neurons: boolean[],
        // состояния рефрактерности нейронов принимающего слоя, доступно только для чтения
        output_refract_intervals: int[],
    ) {
        // функция GPU, получающая индекс нейрона
        int output_neuron_index = get_neuron_index();

        for (int input_neuron_index = 0; input_neuron_index < size(output_neurons); input_neuron_index += 1) {
            if (input_neurons[input_neuron_index] > 0) {
                if (output_refract_intervals[output_neuron_index] > 0) {
                    input_synapses[input_neuron_index, output_neuron_index] -= step;
                } else if (output_neurons[output_neuron_index] > 0) {
                    input_synapses[input_neuron_index, output_neuron_index] += step;
                }
            }
        }
    }
```

Рекомендации по повышению эффективности. Необходимо также учитывать, что многие существующие модели не поддерживают тип данных boolean, что приводит к необходимости использовать минимум 8 бит для хранения бинарного состояния. Стоит рассмотреть побитовые операции для эффективности организации памяти.

Также при программной реализации необходимо ограничить диапазон весов синапсов минимальным и максимальным значениями, что было опущено в псевдокоде для удобства чтения.

Весовые коэффициенты, вычисляемые функцией get_weight_coefficient, не меняются во время работы программы, соответственно, их можно посчитать один раз при инициализации программы. Это приведет к дополнительному расходу оперативной памяти, но позволит избежать таких медленных математических операций, как деление или вычисление экспонент.

Результаты моделирования. Параметры устройства и ОС, на котором проводились измерения:

CPU(s): 8
Model name: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
Thread(s) per core: 2
Core(s) per socket: 4
Socket(s): 1
NUMA node0 CPU(s): 0-7
SSD: 476.9G SAMSUNG MZVLB512HAJQ-000H1
Максимальная частота GPU: до 1100 МГц
Производительность GPU: до 0,25 TFLOPS (FP32)
ОС: Ubuntu 24.04.3 LTS

Сравнение производительности было проведено на различных нейросетях с одинаковой архитектурой, но отличающихся количеством логических полей в слоях. В результате в каждом слое сети использовалось разное количество нейронов, что позволило оценить влияние масштабируемости структуры на эффективность работы модели. Одним циклом считается обучение модели на выборке из 1000 элементов и вычисление прогноза с горизонтом в 72 такта. Сравнение производительности представлено в виде таблицы.

Сравнение производительности реализаций на CPU и GPU
Performance comparison of CPU and GPU implementations

Количество логических полей в каждом слое	Количество нейронов в одном слое	Среднее время выполнения одного цикла, с	
		на GPU	на CPU
2	648	3.385	33.679
3	972	4.861	76.043
4	1296	4.987	134.823
5	1620	5.525	214.781
6	1944	7.041	315.442
7	2268	9.018	404.078
8	2592	9.827	530.350
9	2916	10.214	675.929
10	3240	11.257	822.288
11	3564	12.277	1010.922
12	3888	14.190	1183.601
13	4212	14.190	1402.023
14	4536	16.407	1627.382
15	4860	20.553	1862.220

Как видно из данных таблицы, время вычислений на GPU существенно ниже, чем на CPU, и разница между ними увеличивается по мере роста числа нейронов. Например, при 648 нейронах среднее время работы на GPU составляет 3.385 с, а на CPU – 33.679 с, что в 10 раз медленнее. При увеличении количества нейронов до 4860 эта разница становится еще более заметной: на GPU вычисления занимают 20.553 с, а на CPU – 1862.220 с, т. е. GPU быстрее примерно в 90 раз.

Это подтверждает высокую эффективность реализации предложенной архитектуры нейросети на GPU, особенно при больших размерах слоев. Использование GPU позволяет существенно ускорить обработку данных и сделать архитектуру пригодной для масштабирования и применения в задачах с большими объемами информации.

Заключение. В статье предложен способ реализации архитектуры нейронной сети на GPU, включая эффективные методы хранения данных, алгоритмы обработки на CPU и GPU, а также механизмы их взаимодействия. Показано, что правильное распределение вычислительных задач между центральным и графическим процессорами позволяет реализовать высокопроизводительную и масштабируемую архитектуру, способную оперативно обрабатывать большие потоки данных. Проведенные эксперименты подтвердили возможность предложенной модели эффективно прогнозировать будущие состояния системы, что подтверждает ее практическую применимость для задач анализа и прогнозирования сложных динамических процессов в реальном времени.

Список литературы

1. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. 2015. Vol. 521. Art. 7553. P. 436–444. doi: 10.1038/nature14539.

2. Bengio Y., Courville A., Vincent P. Representation learning: A review and new perspectives // IEEE Trans. on Pattern Analysis and Machine Intelligence. IEEE, 2013. Vol. 35, no. 8. P. 1798–1828. doi: 10.1109/TPAMI.2013.50.

3. Pan S. J., Yang Q. A survey on transfer learning // IEEE Transactions on Knowledge and Data Engin. IEEE, 2010. Vol. 22, no. 10. P. 1345–1359. doi: 10.1109/TKDE.2009.191.

4. Overcoming catastrophic forgetting in neural networks / J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, R. Hadsell // Proc. of the National Acad. of Sci. 2017. Vol. 114, no. 13. P. 3521–3526. doi: 10.1073/pnas.1611835114.

5. Oh K. S., Jung K. GPU implementation of neural networks // Pattern Recognition. 2004. Vol. 37, no. 6. P. 1311–1314. doi: 10.1016/j.patcog.2004.01.013.

6. Осипов В. Ю. Интеллектуальная нейросетевая машина с функциями мышления // Информатика и автоматизация. 2024. Т. 23, № 4. С. 1077–1109. doi: 10.15622/ia.23.4.6.

7. Осипов В. Ю. Устойчивость интеллектуальной обработки сигналов потоковыми рекуррентными нейронными сетями с непрерывным обучением // Информатика и автоматизация. 2025. Т. 24, № 6. С. 1649–1682. doi: 10.15622/ia.24.6.5.

8. Urban traffic flows forecasting by recurrent neural networks with spiral structures of layers / Osipov V., Nikiforov V., Zhukova N. D. Miloserdov // Neural Comp. and Appl. 2020. Vol. 32, no. 18. P. 14885–14897. doi: 10.1007/s00521-020-04843-5.

9. Богомаз М. Графический процессор (GPU): что это и как применяется. 2023. URL: https://timeweb.cloud/blog/graficheskij-processor-gpu-chto-ehto-i-kak-primenyaetsya (дата обращения: 26.03.2026).

10. Steffen M., Zambreno J. Improving SIMT Efficiency of global rendering algorithms with architectural support for dynamic micro-Kernels // 43rd Ann. IEEE/ACM Intern. Symp. on Microarchitecture. Atlanta, GA, USA: IEEE, 2010. P. 237–248. doi: 10.1109/MICRO.2010.45.

Информация об авторе

Тайц Вадим Максимович – аспирант СПИИРАН СПб ФИЦ РАН, 14-я линия В. О., д. 39, Санкт-Петербург, 199178, Россия.
E-mail: taizvadim@gmail.com

References

1. LeCun Y., Bengio Y., Hinton G. Deep learning // Nature. 2015. Vol. 521. Art. 7553. P. 436–444. doi: 10.1038/nature14539.

2. Bengio Y., Courville A., Vincent P. Representation learning: A review and new perspectives // IEEE Transactions on Pattern Analysis and Machine Intelligence. IEEE, 2013. Vol. 35, no. 8. P. 1798–1828. doi: 10.1109/TPAMI.2013.50.

3. Pan S. J., Yang Q. A survey on transfer learning // IEEE Trans. on Knowledge and Data Engin. IEEE, 2010. Vol. 22, no. 10. P. 1345–1359. doi: 10.1109/TKDE.2009.191.

4. Overcoming catastrophic forgetting in neural networks / J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, R. Hadsell // Proc. of the National Academy of Sci. 2017. Vol. 114, no. 13. P. 3521–3526. doi: 10.1073/pnas.1611835114.

5. Oh K. S., Jung K. GPU implementation of neural networks // Pattern Recognition. 2004. Vol. 37, no. 6. P. 1311–1314. doi: 10.1016/j.patcog.2004.01.013.

6. Osipov V. Yu. Intellektualnaja nejrosetevaja mashina s funkcijami myshlenija // Informatika i avtomatizacija. 2024. T. 23, № 4. S. 1077–1109. doi: 10.15622/ia.23.4.6. (In Russ.).

7. Osipov V. Yu. Ustojchivost intellektualnoj obrabotki signalov potokovymi rekurrentnymi nejronnymi setjami s nepreryvnym obucheniem // Informatika i avtomatizacija. 2025. T. 24, № 6. S. 1649–1682. doi: 10.15622/ia.24.6.5. (In Russ.).

8. Urban traffic flows forecasting by recurrent neural networks with spiral structures of layers / Osipov V., Nikiforov V., Zhukova N. D. Miloserdov // Neural Comp. and Appl. 2020. Vol. 32, no. 18. P. 14885–14897. doi: 10.1007/s00521-020-04843-5.

9. Bogomaz M. Graficheskij processor (GPU): chto eto i kak primenjaetsja. 2023. URL: https://timeweb.cloud/blog/graficheskij-processor-gpu-chto-ehto-i-kak-primenyaetsya (data obrashhenija: 26.03.2026) (In Russ.).

10. Steffen M., Zambreno J. Improving SIMT efficiency of global rendering algorithms with architectural support for dynamic micro-Kernels // 43rd Ann. IEEE/ACM Intern. Symp. on Microarchitecture. Atlanta, GA, USA: IEEE, 2010. P. 237–248. doi: 10.1109/MICRO.2010.45.

Information about the author

Vadim M. Taits, postgraduate student, SPIIRAS of St. Petersburg Federal Research Center of the Russian Academy of Sciences, 14st line, 39, Vasilievsky Island, Saint Petersburg, 199178, Russia.
E-mail: taizvadim@gmail.com

Статья поступила в редакцию 17.12.2025; принята к публикации после рецензирования 19.03.2026; опубликована онлайн 25.05.2026.

Submitted 17.12.2025; accepted 19.03.2026; published online 25.05.2026.