УДК 004.021

ВЫЧИСЛЕНИЕ И ОБУЧЕНИЕ ИСКУСТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ ПРЯМОГО РАСПРОСТАНЕНИЯ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

С. А. Запрягаев, А. А. Карпушин

Воронежский Государственный Университет

Поступила в редакцию 29.03.2011 г.

Аннотация. В статье рассматривается возможность использования графического процессора в задаче обучения искусственных нейронных сетей на основе многослойного перцептрона. Предлагается алгоритм, значительно ускоряющий обучение нейронной сети на графическом процессоре в сравнении с обучением сети на центральном процессоре.
Ключевые слова: параллельные вычисления, искусственные нейронные сети, многослойный перцептрон, OpenCL.
Abstract. The article describes implementation of artificial neural networks on GPU. An effective neural network training algorithm, which takes into account GPU architecture, is provided. The comparison to popular artificial neural networks libraries is made.
Keywords: artificial neural networks, GPGPU, OpenCL.

ВВЕДЕНИЕ

Применение искусственных нейронных сетей получило широкое распространение со времени разработки одной из первых моделей нейрона МакКалока-Питса. Суммарное количество нейронов в современных искусственных нейронных сетях достигает нескольких тысяч, а количество межнейронных связей в них сотен тысяч.

Обучение и вычисление таких нейронных сетей требует огромных вычислительных ресурсов. Для решения этой проблемы используются как специальные типы архитектур нейронных сетей [1], аккумулирующие промежуточные данные между эпохами обучения сети, так и различные методы распараллеливания и распределенного вычисления нейронной сети [2, 3].

Другим способом уменьшения времени обучения нейронной сети является использование графического процессора во время ее обучения. Современный графический процессор, являющийся неотъемлемой частью любого персонального компьютера, представляет собой высокопроизводительное устройство, архитектура которого позволяет выполнять несколько сотен простых операций параллельно.

Возможность программировать GPU привела к использованию его для решения широкого класса задач, несвязанных с графикой. Такое использование графического процессора получило название GPGPU (General Purpose computations on Graphics Processing Unit). Типичными областями применения GPGPU стали: видеообработка [4], вычислительная химия [5], визуализация в медицине [6], обработка сигналов [7] и др.

Результатом развития GPU и его использования в задачах, несвязанных с компьютерной графикой, стала разработка единого стандарта описания гетерогенных вычислений на высокопараллельных системах – OpenCL (Open Computational Language).

OpenCL позволяет описывать вычисления, абстрагируясь от конкретного устройства, на котором эти вычисления могут быть исполнены. Так, алгоритмы, написанные с использованием OpenCL, могут исполняться на нескольких ядрах центрального процессора, или на графическом процессоре.

Цель настоящей работы заключается в разработке алгоритма обучения нейронной сети с применением графического процессора. Исследуется область применимости разрабатываемого алгоритма и анализируется его производительность по сравнению с имеющимися решениями по работе с искусственными нейронными сетями.

1. МЕТОДЫ ОБУЧЕНИЯ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ ПРЯМОГО РАСПРОСТРАНЕНИЯ.

Одним из распространенных типов искусственных нейронных сетей является сеть прямого распространения. Такая сеть состоит из нескольких слоев, каждый из которых имеет только односторонние связи со следующим слоем. В такой сети отсутствуют циклы (рис. 1). Для обучения нейронной сети прямого распространения может использоваться один из градиентных методов обучения [3]. Градиент при этом вычисляется на основе сопряженного графа сети с использованием метода обратного распространения ошибки.

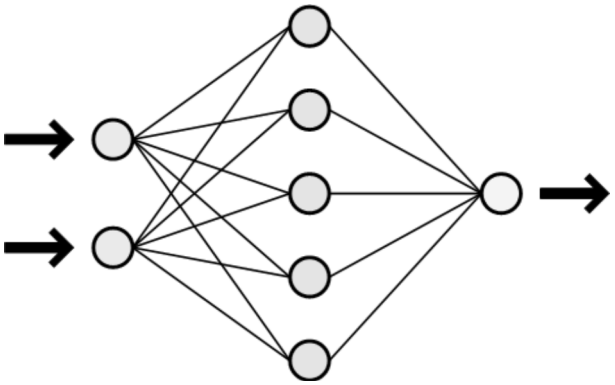

Рис. 1. Схема нейронной сети прямого распространения.

Несмотря на то, что обучение искусственной нейронной сети является ресурсоемким процессом, вычисления, выполняемые для одного нейрона сети, описываются достаточно просто и не требуют больших ресурсов. Вместе с тем, в нейронных сетях прямого распространения, нейроны одного слоя не имеют межнейронных связей между собой. Эти два фактора делают возможным разработку эффективного алгоритма для распараллеливания вычислений отдельного слоя нейронной сети, в том числе и на графическом процессоре.

Слой нейронной сети характеризуется набором исходных сигналов, подающихся на вход сети, набором весов, функцией активации каждого нейрона и набором выходных сигналов слоя.

Выходной сигнал i-го нейрона в слое описывается формулой :

$$y_i = f\left(u_i\right), \qquad (1)$$

где f – функция активации нейрона, $u_i = \sum_{j=0}^{N} w_{ij} x_j$ – суммарный сигнал, пришедший на i-й нейрон, w_{ij} – вес j-й межнейронной связи для i-го нейрона, x_j – сигнал, поданный на j-й вход нейрона, N – общее количество исходных сигналов (с учетом сигнала поляризации $x_0 = 1$), y_i – итоговый выходной сигнал нейрона.

Обозначения, введенные в формуле , схематично представлены на рис. 2.

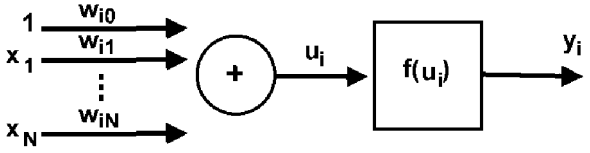

Рис. 2. Модель нейрона

Веса нейронов отдельного слоя сети удобно представить в виде матрицы W , где количество строк соответствует количеству нейронов в слое, а количество столбцов соответствует количеству входных сигналов для каждого нейрона. Входные сигналы для слоя, в свою очередь, можно представить как вектор-столбец X :

$$W = \begin{pmatrix} w_{10} & w_{11} & w_{12} & \cdots & w_{1N} \\ w_{20} & w_{21} & w_{22} & \cdots & w_{2N} \\ w_{30} & w_{31} & w_{32} & \cdots & w_{3N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{M0} & w_{M1} & w_{M2} & \cdots & w_{MN} \end{pmatrix}, \qquad (2)$$

$$X = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}.$$

Тогда, для вычисления выходных сигналов слоя нейронной сети необходимо вычислить вектор $U = W \times X$, а затем для каждой компоненты полученного вектора посчитать значение функции активации f . Параллельное вычисление произведения матрицы на вектор-столбец является часто встречаемой задачей. Эффективная реализация алгоритма матричного умножения на графическом процессоре подробно описана в [8]. Суть реализации алгоритма умножения матрицы на вектор-столбец на графи-

ческом процессоре заключается в том, что каждый вычислительный элемент рассчитывает значения нескольких компонент результирующего вектора-строки.

Для вычисления всей нейронной сети выходные данные одного слоя используются в качестве входных данных следующего за ним слоя сети. При этом эффективно распараллелить процесс вычисления различных слоев сети не представляется возможным в силу зависимости данных, используемых в разных слоях, между собой.

Обучение нейронной сети приводит к минимизации конкретной функции погрешности (целевой функции). Часто целевая функция определяется как:

$$E = \frac{1}{2}\sum_{k=1}^{p}\sum_{i=1}^{n}\left(y_i^{(k)} - d_i^{(k)}\right)^2, \qquad (3)$$

где p – количество предъявляемых обучающих выборок, n – количество нейронов выходного слоя сети, $y_i^{(k)}$ – фактическое выходное значение для i-го нейрона k-ой обучающей выборки, $d_i^{(k)}$ – целевое выходное значение для i-го нейрона k-ой обучающей выборки.

Целевая функция, определяемая формулой , является функцией от совокупности весов всех нейронных связей сети. С учетом обозначений веса нейронной сети можно объединить в вектор-столбец $\overline{w}$, компоненты которого определяются следующим образом:

$$w_1 = w_{10}^{(1)},\ w_2 = w_{11}^{(1)},\ w_n = w_{MN}^{(K)}, \qquad (4)$$

где $w_{ij}^{(k)}$ – вес j-й межнейронной связи для i-го нейрона в слое k.

В теории оптимизации наиболее эффективными алгоритмами минимизации значения целевой функции являются градиентные методы. В этом случае целевая функция $E(\overline{w})$ разлагается в ряд Тейлора в окрестности точки имеющегося решения $\overline{w}$ (при старте алгоритма это некоторая исходная точка $\overline{w}_0$, выбор которой может быть связан с конкретной задачей). Такое разложение вдоль направления $\overline{p}$ в общем виде описывается следующим образом:

$$E\left(\overline{w}+\overline{p}\right) = E\left(\overline{w}\right) + \\ +\left[g\left(\overline{w}\right)\right]^T\overline{p} + \frac{1}{2}\overline{p}^T H\left(\overline{w}\right)\overline{p} + \ldots, \qquad (5)$$

где $g\left(\overline{w}\right) = \nabla E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \cdots, \frac{\partial E}{\partial w_n}\right]^T$ – вектор-столбец, градиент функции E, $H\left(\overline{w}\right)$ – ее гессиан, определяемый как:

$$H\left(\overline{w}\right) = \begin{pmatrix} \frac{\partial^2 E}{\partial w_1^2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial^2 E}{\partial w_n \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_n^2} \end{pmatrix}. \qquad (6)$$

Алгоритмы обучения нейронной сети, использующие гессиан $H\left(\overline{w}\right)$ или его приближение, такие как алгоритм переменной метрики и алгоритм Левенберга-Марквардта, достигают требуемой величины целевой функции за меньшее на порядок число итераций по сравнению с алгоритмами, использующими только градиент $g\left(\overline{w}\right)$. Но использование таких алгоритмов на нейронных сетях значительных размеров (более 1000 межнейронных связей) не так эффективно в силу проблем производительности и хранения матрицы гессиана [9].

Для больших нейронных сетей оптимальными алгоритмами обучения являются: алгоритм метода наискорейшего спуска, алгоритм метода сопряженных градиентов, а также алгоритмы на основе эвристических методов обучения Quickprop [10] и RPROP [11].

Метод наискорейшего спуска в качестве направления $\overline{p}$ использует направление, противоположное градиенту $g\left(\overline{w}\right)$:

$$\overline{p} = -g\left(\overline{w}\right) \qquad (7)$$

В рамках метода наискорейшего спуска уточнение весов для $k+1$ шага обучения ($\overline{w}_{k+1} = \overline{w}_k + \Delta\overline{w}_k$) осуществляется по следующей формуле:

$$\Delta\overline{w} = \eta_k\overline{p}_k + \alpha\left(\overline{w}_k - \overline{w}_{k-1}\right), \qquad (8)$$

где α – коэффициент момента, принимающий значения в интервале $[0, 1]$, а η_k – коэффициент обучения, используемый на k-м шаге обучения. Коэффициент обучения на первом шаге обучения определяется пользователем. Первое слагаемое в формуле соответствует обычному обучению по методу наискорейшего спуска, второе слагаемое учитывает последнее изменение весов и не зависит от фактического значения градиента. Таким образом, влияние второго слагаемого значительно возрастает на плоских участках целевой функции, а также вблизи локального минимума, где значение градиента $g\left(\overline{w}\right)$ близко к нулю, что предотвращает замедление процесса обучения нейронной сети.

Метод сопряженных градиентов выбирает очередное направление $\overline{p}_k$ так, чтобы оно было

ортогональным ко всем предыдущим $\overline{p}_0$, $\overline{p}_1$, ..., $\overline{p}_{k-1}$. Множество векторов $\overline{p}_i, i = 0,1,...,k$ будет взаимно сопряженным относительно произвольной матрицы G, если

$$\overline{p}_i^T G \overline{p}_j = 0, i \neq j. \quad (9)$$

Вектор $\overline{p}_k$, удовлетворяющий условию , имеет вид [12, 13]:

$$\overline{p}_k = -\overline{g}_k + \beta_{k-1}\overline{p}_{k-1}, \quad (10)$$

где $\overline{g}_k = g(\overline{w}_k)$ – значение вектора градиента на шаге k, β_{k-1} – коэффициент сопряжения. Существуют различные правила подсчета коэффициента сопряжения. Наиболее известные из них приведены в работе [13] и имеют вид:

$$\beta_{k-1} = \frac{\overline{g}_k^T (\overline{g}_k - \overline{g}_{k-1})}{\overline{g}_{k-1}^T \overline{g}_{k-1}}, \quad (11)$$

и

$$\beta_{k-1} = -\frac{\overline{g}_k^T (\overline{g}_k - \overline{g}_{k-1})}{\overline{p}_{k-1}^T \overline{g}_{k-1}}. \quad (12)$$

При использовании метода сопряженных градиентов после нескольких итераций оптимизации в расчете коэффициентов сопряжения накапливается погрешность, приводящая к утрате свойств ортогональности между векторами $\overline{p}_i, i = 0,...,k$. По этой причине метод сопряженных градиентов нужно «перезапускать» через несколько итераций, выбирая в качестве исходного направления $\overline{p}_0$ – направление по методу наискорейшего спуска. Метод сопряженных выполняется значительно быстрее, чем метод наискорейшего спуска [9].

Алгоритм Quickprop содержит элементы, предотвращающие зацикливание в точке неглубокого локального минимума, где из-за близости к нулю производной функции активации, процесс обучения прекращается. Такая ситуация возникает, из-за больших, по абсолютному значению, величин весов. В алгоритме Quickprop, вес на k-м шаге изменяется согласно формуле:

$$\Delta w_{ij}(k) = -\eta_k \left[\frac{\partial E(w(k))}{\partial w_{ij}} + \gamma w_{ij}(k) \right] + \alpha_{ij}^{(k)} \Delta w_{ij}(k-1). \quad (13)$$

Формула является обобщением формулы , использующейся в методе наискорейшего спуска. Коэффициент γ, имеющий величину (от 1 до 10), – это фактор, приводящий к уменьшению весов. Кроме того, коэффициент момент $\alpha_{ij}^{(k)}$ теперь определяется для каждой межнейронной связи индивидуально. В оригинальной работе Фальмана этот коэффициент выбирается следующим образом:

$$\alpha_{ij}^{(k)} = \begin{cases} \alpha_{\max}, \text{ если } \beta_{ij}(k) > \alpha_{\max} \\ \text{и } S_{ij}(k)\Delta w_{ij}(k-1)\beta_{ij}(k) < 0, \\ \beta_{ij}(k), \text{ в остальных случаях} \end{cases} \quad (14)$$

где $\alpha_{\max}$ – максимальное значение коэффициента момента и

$$S_{ij}(k) = \frac{\partial E(w(k))}{\partial w_{ij}} + \gamma w_{ij}(k), \quad (15)$$

$$\beta_{ij}^{(k)} = \frac{S_{ij}(k)}{S_{ij}(k-1) - S_{ij}(k)}. \quad (16)$$

Алгоритм RPROP – эвристический алгоритм, учитывающий только знак компонент градиента, но не их значения. Модификация весов в этом алгоритме осуществляется следующим способом:

$$\Delta w_{ij}(k) = -\eta_{ij}(k) \operatorname{sgn}\left(\frac{\partial E(w(k))}{\partial w_{ij}} \right). \quad (17)$$

Кроме того, коэффициент обучения на каждом шаге определяется индивидуально для каждой связи по следующей формуле:

$$\eta_{ij}(k) = \begin{cases} \min(a\eta_{ij}(k-1), \eta_{\max}), \text{ если } B_{ij}(k) > 0 \\ \max(b\eta_{ij}(k-1), \eta_{\min}), \text{ если } B_{ij}(k) < 0 \\ \eta_{ij}(k-1), \text{ если } B_{ij}(k) = 0, \end{cases} \quad (18)$$

где $B_{ij}(k) = \tilde{S}_{ij}(k)\tilde{S}_{ij}(k-1)$, $\tilde{S}_{ij}(k) = \frac{\partial E(w(k))}{\partial w_{ij}}$, a и b – константы, определяющие увеличение и уменьшение коэффициента обучения, соответственно, $\eta_{\min}$ и $\eta_{\max}$ – минимальное и максимальное значения коэффициента обучения.

В алгоритме RPROP в соответствии со стратегией подбора весов, если на двух последовательных шагах обучения знак градиента не изменился, происходит увеличение коэффициента обучения, если же знак градиента изменяется – происходит уменьшение коэффициента обучения.

Каждый из рассмотренных алгоритмов требует вычисление компонент вектора градиента $g(\overline{w})$ для каждой межнейронной связи сети.

Как показано в [9] наиболее эффективным алгоритмом вычисления вектора градиента $g(\overline{w})$ является алгоритм, основанный на применении потоковых графов.

Суть алгоритма заключается в построении к графу нейронной сети G сопряженного с ним графа $\hat{G}$ и анализе графа $\hat{G}$ при подаче сигнала на его входы (рис. 4).

Сопряженный граф $\hat{G}$ возбуждается разностями между фактическими y_i и ожидаемыми d_i значениями выходных сигналов. Дуги графа G, которым соответствует применение функции активации f, заменяются в сопряженном графе производными $\frac{df(u)}{du}$, значения которых рассчитываются раздельно для каждого слоя в точках $u_i^{(k)} = \sum_{j=0}^{N} w_{ij}^{(k)} x_j^{(k)}$. Для сигмоидальной биполярной функции активации $f(x) = \tanh(bx)$ ее производная в точке u_i вычисляется как $\left.\frac{df(x)}{dx}\right|_{x=u_i} = \beta f(u_i)\left(1 - f(u_i)\right)$. При этом используются значения $f(u_i)$, полученные на этапе вычисления нейронной сети по графу G, и не требуется никаких дополнительных вычислений.

Предложенный алгоритм позволяет рассчитать конкретные компоненты вектора градиента $g(\overline{w})$ для любого слоя нейронной сети:

– для выходного слоя:

$$\frac{\partial E(\overline{w})}{\partial w_{ij}^{(m)}} = v_j^{(m-1)} \hat{u}_i^{(m)}; \qquad (19)$$

– для k-го скрытого слоя:

$$\frac{\partial E(\overline{w})}{\partial w_{ij}^{(k)}} = v_j^{(k-1)} \hat{u}_i^{(k)}; \qquad (20)$$

– для первого скрытого слоя:

$$\frac{\partial E(\overline{w})}{\partial w_{ij}^{(1)}} = x_j \hat{u}_i^{(1)}. \qquad (21)$$

Из приведенных формул видно, что для расчета любого компонента градиента нужно знать всего два сигнала: от узла, из которого исходит взвешенная дуга в оригинальном графе G, и от узла, из которого исходит взвешенная дуга в сопряженном графе $\hat{G}$.

Иллюстрация метода применения сопряженного графа приведена на рис. 3 и рис. 4.

Таким образом, для вычисления компонент вектора градиента $g(\overline{w})$ нейронной сети необходимо сначала вычислить нейронную сеть в прямом направлении, затем подать на выходы сети сигналы ошибки $y_i - d_i$ и вычислить градиент для каждой межнейронной связи, «распространяя» сигнал ошибки в обратном направлении по сети. Распространение сигнала ошибки по сети в обратном направлении осуществляется по следующей формуле:

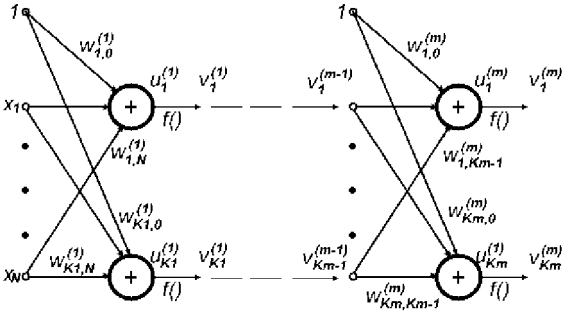

Рис. 3. Граф нейронной сети G

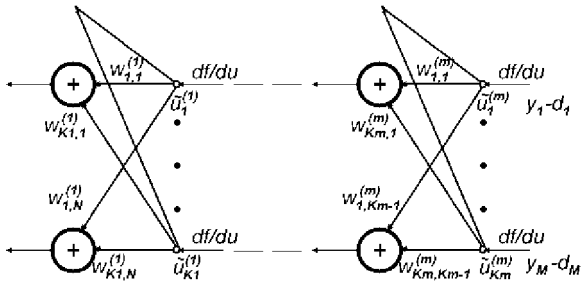

Рис. 4. Сопряженный граф нейронной сети $\hat{G}$

$$e_i^{(k-1)} = \sum_{j=1}^{K_k} w_{ji}^{(k)} \hat{u}_j^{(k)}, \qquad (22)$$

где $e_i^{(k)}$ – величина ошибки для i-го нейрона в k-слое, для нейронов выходного слоя $e_i^{(m)} = y_i - d_i$; $\hat{u}_j^{(k)} = \frac{df}{du}\left(u_j^{(k)}\right) \cdot e_j^{(k)}$.

Вычисление фактических величин ошибок для отдельного слоя нейронной сети удобно представить в векторно-матричном виде:

$$E_{k-1} = W_k^T \hat{U}_k, \qquad (23)$$

где $E_{k-1} = \left[e_1^{(k-1)} \quad e_2^{(k-1)} \quad \cdots \quad e_{K_k}^{(k-1)}\right]^T$ – вектор-столбец фактических величин ошибок для $(k-1)$-го слоя нейронной сети; W_k – матрица весов k-го слоя сети; $\hat{U}_k = \left[\hat{u}_1^{(k)} \quad \hat{u}_2^{(k)} \quad \cdots \quad \hat{u}_{K_k}^{(k)}\right]$ – вектор сигналов сопряженного графа $\hat{G}$ для k-го слоя сети.

Обучение нейронной сети может выполняться как в режиме *онлайн* – уточнение весов происходит после предъявления каждой обучающей выборки, так и в режиме *оффлайн* – пос-

ле предъявления всех обучающих данных. Для режима обучения «оффлайн» итоговое значение вектора градиента, используемое в различных методах обучения нейронной сети, получается путем суммирования векторов градиента $g(\overline{w})$, полученных после предъявления каждой обучающей выборки.

2. АЛГОРИТМ ОБУЧЕНИЯ НЕЙРОННОЙ СЕТИ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

С учетом всего вышеизложенного общий алгоритм обучения нейронной сети прямого распространения с несколькими скрытыми слоями может быть описан следующей блок-схемой (рис. 5).

В целом, алгоритм процесса обучения с использованием графического процессора состоит из следующих основных шагов:

1. Подготовка начальных данных. На этом шаге пользователем определяется конфигурация нейронной сети, формируются обучающие выборки, выбирается алгоритм и параметры обучения (метод наискорейшего спуска, сопряженных градиентов, Quickprop, RPROP). На этом шаге также копируются все необходимые данные в видеопамять.

2. Вычисление нейронной сети в прямом направлении. На этом шаге хост-приложение добавляет в очередь выполнения GPU необходимые вызовы микропрограмм и их параметры. Добавление осуществляется слой за слоем, начиная с самого первого слоя.

3. Вычисление нейронной сети в обратном направлении. Хост-приложение устанавливает величину погрешности для каждого нейрона выходного слоя, затем добавляет в очередь выполнения GPU вызовы микропрограмм и их параметры для подсчета градиента связей каждого слоя и распространения ошибки на предыдущие слои.

4. Модификация весов межнейронных связей в соответствии с градиентом и параметрами обучения.

5. Расчет среднеквадратичной погрешности и модификация параметров обучения в соответствии с алгоритмом обучения.

Следует отметить, что шаги 2–5 выполняются полностью на графическом процессоре. Центральный процессор осуществляет только добавление команд в очередь выполнения, но никак не контролирует момент исполнения этих команд.

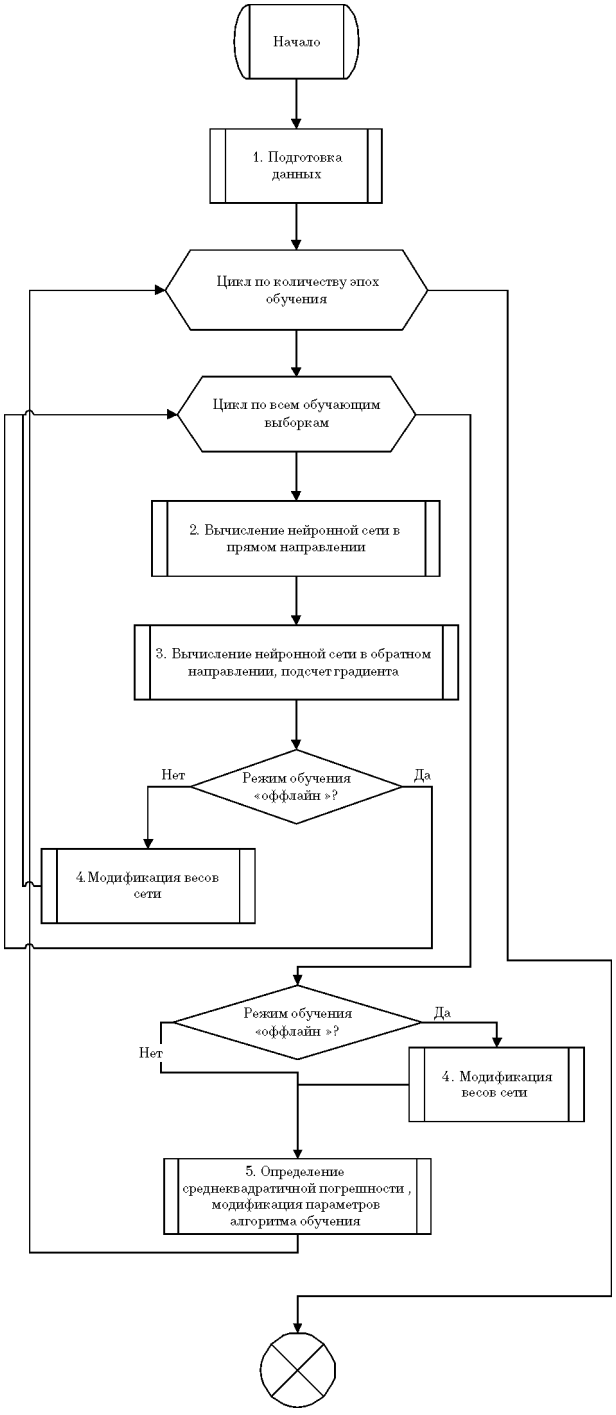

Рис. 5. Блок-схема алгоритма обучения нейронной сети

В целях оптимизации работы, передача данных между оперативной памятью и видеопамятью в библиотеке сведена к минимуму. Хост-приложение, по своей сути, в процессе обучения нейронной сети лишь осуществляет добавление команд в очередь выполнения графического процессора и сравнивает значение среднеквад-

ратичной погрешности обучения с целевым значением для прекращения процесса обучения. Сравнение среднеквадратичной погрешности с целевым значением происходит не на каждой итерации, так как это требует ожидания данных от графического процессора, что приводит к остановке очереди выполнения.

При добавлении вызовов микропрограмм учитываются особенности работы очередь выполнения GPU. Графический процессор может выполнять команды из очереди выполнения как в порядке их добавления, так и в произвольном порядке, в зависимости от заданных параметров.

Выполнение команд в произвольном порядке является более гибким решением, так как для каждой команды в очереди выполнения указывается исчерпывающий список команд, от которых она зависит. Если в определенный момент времени в очереди команд оказывается несколько команд, каждая из который доступна для выполнения, то эти команды выполняются параллельно, при наличии нескольких видеокарт в системе. Это особенно актуально при запуске программного комплекса на высокопроизводительных кластерах или суперкомпьютерах, где установлено несколько видеокарт.

3. АНАЛИЗ РЕЗУЛЬТАТОВ

Для анализа производительности разработанного алгоритма был проведен ряд тестов. В качестве основной метрики производительности была выбрана метрика среднего времени τ, затрачиваемого на модификацию веса одной межнейронной связи, вычисляемая по формуле:

$$\tau = \frac{T}{N \cdot K}, \qquad (24)$$

где T – общее время обучения, N – количество итераций обучения, K – общее количество весов нейронной сети.

Был проведен тест для определения зависимости среднего времени модификации одного веса от количества весов. Нейронная сеть содержала 1 скрытый слой, в котором количество нейронов варьировалось от 1 до 100000, максимальное количество связей, таким образом, не превышало $4 \cdot 10^5$. При этом тестовый компьютер имел следующую конфигурацию: CPU – Intel Core 2 Duo, RAM – 2GB, GPU – GeForce 8800GT.

В данной работе был также проведен сравнительный анализ производительности с другими библиотеками для работы с нейронными сетями.

В качестве кандидата для сравнения была выбрана одна из самых быстрых и оптимизированных современных библиотек – FANN (Fast Artificial Neural Network Library) [14]. Эта библиотека использует вычисления только на центральном процессоре. FANN также позволяет вывести метрику среднего времени, затрачиваемого на модификацию веса одной межнейронной связи. Полученные данные свидетельствуют о том, что пиковая производительность библиотеки FANN в 28–30 наносекунд достигается при общем количестве связей нейронной сети порядка $5 \cdot 10^4$. При дальнейшем увеличении размеров нейронной сети, среднее время на модификацию одного веса сети начинает возрастать. Это, прежде всего, связано с тем, что центральный процессор не может эффективно обрабатывать большие объемы данных, так как они не помещаются полностью в кэш центрального процессора.

Полученные результаты приведены на рис. 6.

Рис. 6. Зависимость среднего времени модификации одного веса нейронной сети от общего количества весов

ЗАКЛЮЧЕНИЕ

Из полученных результатов можно сделать вывод о том, что использование графического процессора при вычислении и обучении нейронной сети оказывается более эффективным при больших размерах сети, в отличие от реализации на центральном процессоре.

СПИСОК ЛИТЕРАТУРЫ

1. *Noel Lopes and Bernardete Ribeiro*. 2003. An efficient gradient-based learning algorithm applied to neural networks with selective actuation neurons. Neural, Parallel Sci. Comput. 11, 3 (September 2003), 253–272.

2. *Герасименко М. С.* Вычисление искусственных нейронных сетей на вычислительных кластерах или ЛВС / М. С. Герасименко // Вестник ВГУ. Серия «Системный анализ и информационные технологии», №1. 2010.

3. *Noel Lopes and Bernardete Ribeiro*. 2009. GPU implementation of the multiple back-propagation algorithm. In Proceedings of the 10th international conference on Intelligent data engineering and automated learning (IDEAL'09), Emilio Corchado and Hujun Yin (Eds.). Springer-Verlag, Berlin, Heidelberg, 449–456.

4. *Bart Pieters* et al. Motion estimation for H.264/AVC on multiple GPUs using NVIDIA CUDA. Proc. SPIE, Vol. 7443, 74430X (2009).

5. *John E.* Stone et al. High Performance Computation and Interactive Display of Molecular Orbitals on GPUs and Multi-core CPUs. ACM International Conference Proceeding Series; Vol. 383 (2009).

6. *Hongsheng Li*, et al, "Actin Filament Tracking Based on Particle Filters and Stretching Open Active Contour Models", Int'l Conf. Medical Image Computing and Computer Assisted Intervention (MICCAI), 2009.

7. *Carmine Clemente*, et al. PROCESSING OF SYNTHETIC APERTURE RADAR DATA WITH GPGPU.

8. http://developer.download.nvidia.com/compute/opencl/sdk/website/samples.html#oclMatVecMul.

9. *Осовский С.* Нейронные сети для обработки информации / С. Осовский // Финансы и статистика, 2004. – 344 с.

10. *Fahlman S. E.* Faster-learning variations on back-propagation: an empirical study. In: D. S. Touretzky, G. E. Hinton and T. J. Sejnowski, (eds), Proc. 1988 Connectionist Models Summer School. Morgan Kaufmann, San Mateo, CA, 1988, pp. 38–51.

11. *Riedmiller, M., Braun, H.* A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP algorithm. in: Ruspini, H., (Ed.) Proc. of the ICNN 93, San Francisco, (IEEE Service Center, Piscataway, NJ, 1993) 586–591.

12. *Gill. P., Murray W., Wright M.* Practical Optimization. – N.Y.: Academic Press, 1981.

13. *Hestenes, Magnus R.; Stiefel, Eduard* (December 1952). "Methods of Conjugate Gradients for Solving Linear Systems". Journal of Research of the National Bureau of Standards 49 (6).

14. http://leenissen.dk/fann/.

Запрягаев Сергей Александрович – доктор физико-математических наук, профессор кафедры цифровых технологий Воронежского государственного университета. Тел. (4732) 208-257. Email: zsa@main.vsu.ru

Карпушин Андрей Александрович – аспирант, кафедра цифровых технологий, Воронежский государственный университет. Тел. (4732) 208-257. Email: reven86@gmail.com

Zapryagaev S. A. – Doctor of Physics-math. Sciences, Professor of the dept. of digital technologies Voronezh State University. Tel. (4732) 208-257. Email: zsa@main.vsu.ru

Karpushin A. A. – Post-graduate student of the dept. of digital technologies Voronezh State University. Tel. (4732) 208-257. Email: reven86@gmail.com