УДК 681.327.12.001.362

ДОРОГОЙ Я.Ю.,
ЯШИН В.Е.,
ЯЦУК С.В.

МНОГОПОТОЧНАЯ ЭМУЛЯЦИЯ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ ПРОИЗВОЛЬНОЙ ТОПОЛОГИИ

В статье рассмотрена структура программного комплекса PANN, позволяющего использовать преимущества многопоточной обработки данных для эмуляции работы искусственных нейронных сетей произвольных топологий.

The structure of the software complex PANN was described in this article. PANN allows the use of the advantages of multithreading for emulation of artificial neural networks of arbitrary topologies.

Введение в проблему

На данный момент, изучение и использование нейронных сетей, снова набирает обороты [1]. Это обусловлено развитием компьютерной техники и постоянного роста вычислительных мощностей, доступных исследователю. Вместе с тем, инструменты для эмуляции работы и предварительного расчета нейронных сетей, развиваются слабо. Одним из преимуществ нейронных сетей является их скорость обработки данных. Хорошо спроектированная и обученная сеть работает на порядки быстрее аналогичных решений с использованием классических алгоритмов. Но, к сожалению, процесс проектирования и обучения нейронной сети занимает несравнимо больше времени, чем прямое ее использование. Зачастую для достижения необходимых результатов нужно перебрать десяток топологий. А сам процесс обучения занимает много времени ввиду необходимости обработки больших объемов данных. Таким образом, необходимо иметь инструмент для проектирования нейронных сетей, который позволил бы задействовать максимум ресурсов компьютера. Сегодня даже на домашних компьютерах стоят многопроцессорные системы, которые при оптимизированном программировании позволяют ускорить работу приложений в несколько раз. К сожалению, ни одна из известных на данный момент систем для проектирования нейронных сетей не использует возможности компьютера на полную мощность. Это объясняется тем, что большинство из таких пакетов написаны давно, когда многопроцессорные системы были мало распространены и не имело смысла оптимизировать код под такие системы. Сейчас же, имея материнскую плату с 2-мя процессорами по 4 ядра в каждом, можно ускорить обучение нейронной сети в 8 раз, что позволило бы существенно сократить время проектирования и тестирования.

Анализ существующих решений

Сейчас существуют такие пакеты для проектирования нейронных сетей:
SNNS – мощная библиотека, разработанная в Штуттгардском университете. Большая часть кода написана еще в начале 90-х на чистом C, без использования объектно-ориентированного подхода. Это сильно усложняет дальнейшее развитие.

Joone – более современный пакет, написан на Java, что несколько отражается на скорости работы. Преимуществом является полностью объектная модель. Но сама по себе библиотека малоразвита, основной упор поставлен на визуализацию, видимо, поэтому разработчики мало времени уделили развитию ядра.

Matlab Neural Network Toolbox - это пакет расширения MATLAB, содержащий средства для проектирования, моделирования, разработки и визуализации нейронных сетей. Основной недостаток - очень низкая скорость работы.

Общим недостатком для всех существующих пакетов, является однопоточность обработки, а также представление нейронной сети исключительно слоями. Слоистое представление упрощает разработку приложения, но отсекает возможность проектирования нейронных сетей с произвольной топологией.

Проблемы многопоточности

Современные операционные системы многозадачны. Это позволяет пользователю запускать одновременно несколько приложений. Операционная система позволяют

запускать большое количество приложений даже на одном физическом процессоре. Такая квазимногозадачность реализовывается путем постоянного переключения активной задачи. Каждое приложение в системе на какой-то квант времени захватывает процессор и выполняет необходимые действия, затем ОС переключает активность на другое приложение и так в бесконечном цикле.

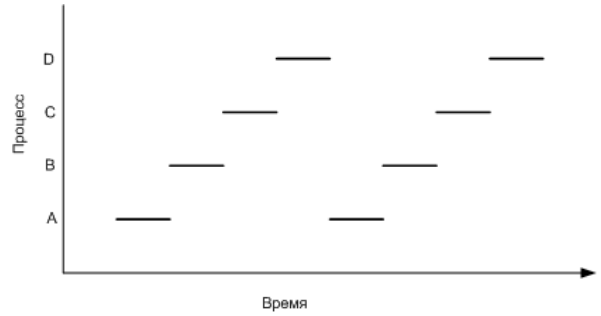

Рис. 1. Многозадачная работа процессора

Если в системе доступно множество процессоров (ядер), то ОС делит процессы на доступные ядра и тогда несколько процессов могут работать действительно одновременно. Более того, можно распараллелить вычисления одного приложения на несколько физических процессоров и тем самым увеличить скорость выполнения сложных вычислений [2].

Основной проблемой при работе многопоточных приложений являются ресурсы компьютера – устройства, потоки ввода/вывода, файлы, данные. Сложность состоит в том, что если несколько потоков используют один и тот же ресурс, то могут возникнуть коллизии. Представим ситуацию: один поток открывает файл для записи данных, и в это же время второй поток удаляет этот файл. В результате первый поток получает невалидный указатель на несуществующий файл. Другой пример, когда два потока пытаются одновременно получить доступ в какую-то область данных. Допустим, существует общая переменная, в которую два потока пишут данные, такое приложение будет нестабильным и работать будет неправильно. Для обеспечения уникальности и атомарности доступа к конкурентным ресурсам, операционная система дает несколько инструментов:

- Критическая область — часть исполняемого кода, который гарантировано выполнится без переключения текущего процесса.
- Семафор — объект, позволяющий войти в заданный участок кода не более чем n потокам.
- Мьютексы — это простейшие двоичные семафоры, которые могут находиться в одном из двух состояний — отмеченном или неотмеченном (открыт и закрыт соответственно). Когда какой-либо поток, принадлежащий любому процессу, становится владельцем объекта mutex, последний переводится в неотмеченное состояние. Если задача освобождает мьютекс, его состояние становится отмеченным.

В разных системах реализация таких системных вызовов отличается, поэтому мы использовали обертку boost::thread, которая унифицирует работу с потоками и дает удобные инструменты.

Цель работы

Таким образом, было принято решение о создании собственного программного пакета для проектирования нейронных сетей с использованием преимущества многопроцессорных систем. Основными требования к продукту стали:

- Высокая скорость обработки данных.
- Многопоточность
- Объектно-ориентированный дизайн
- Модульность программного пакета
- Мультиплатформенность
- Высокая стабильность.

Идея многопоточной обработки

Ввиду изложенных выше сложностей работы многопоточных приложений, было необходимо минимизировать взаимное использование одних и тех же данных разными потоками, так как неправильно спроектированное многопоточное приложение может работать медленнее однопоточного, ввиду возможности постоянной борьбы за уникальные ресурсы [3].

Ранее принятое решение об отказе от канонического послойного представления нейронной сети позволяет создавать сети произвольных топологий. Но так как обработка данных все равно идет последовательно, было введено понятие "расстояния между нейронами" - hop и "длины связи" - link latency [4]. Рассмотрим схему сложной части нейронной сети со сквозной связью. Кругами представлены нейроны N1, N2 и N3. Цифра возле круга - это hop. Цифра на связи - это

дальность связи. Такой подход позволяет проектировать топологии любой сложности (рис. 2).

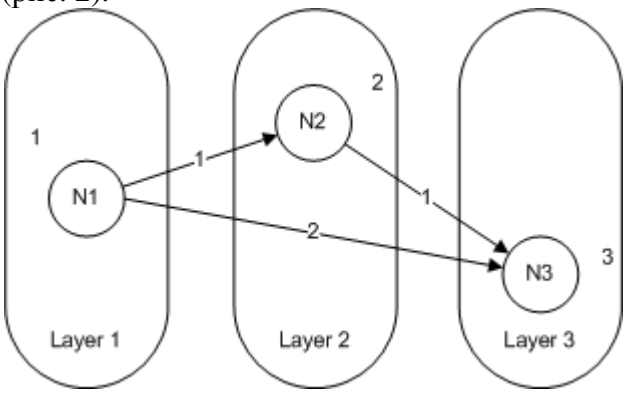

Рис. 2. Представление сложных топологий

Таким образом, мы делим сеть произвольной топологии на условные слои обработки.

Разберем многопоточную обработку нейронной сети на примере многослойного персептрона (рис.3).

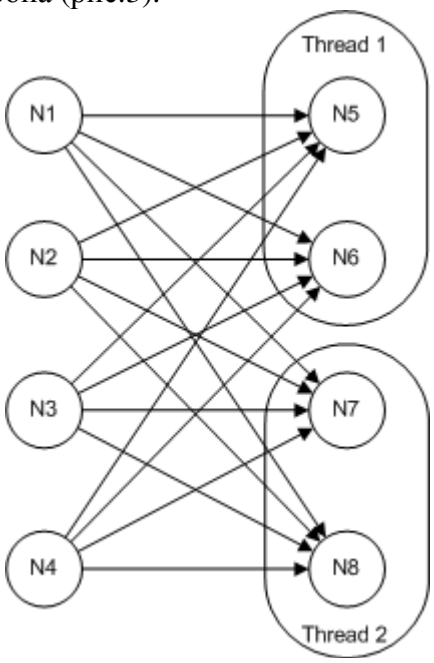

Рис. 3. Часть многослойного персептрона

Для расчета локального рецептивного поля нейрона N5 нужно прочитать значения выходов нейронов N1-N4, умножить на веса и просуммировать, затем те же операции нужно повторить со следующим нейронами N6-N8. Доступ к первому слою будет исключительно на чтение, а значит, обработку нейронов N5-N8 можно выполнять параллельно.

Точно так же можно применить данный алгоритм многопоточной обработки и для сетей более сложной топологии (например, к сверточным нейронным сетям).

Очевидно, что алгоритм обучения обратным распространением ошибки отличается от прямого прохода сети только направлением. Это означает, что данный подход позволяет легко выполнять обучение сети в несколько потоков.

В представленном пакете каждый условный слой делится на несколько частей (по количеству ядер в системе). Далее каждый поток обрабатывает свою часть данных. Очевидно, что для обработки следующего условного слоя, нам необходимо полностью посчитать результаты предыдущего слоя. Чтобы потоки двигались по сети равномерно, после каждого условного слоя ставиться специальный объект boost::barrier, который гарантирует синхронизацию потоков. Таким образом, обработка идет волнами от слоя к слою.

Выбор языка программирования и окружения

Существует множество языков программирования для реализации каких угодно задач. И важно правильно выбрать инструмент, ведь каждый язык имеет как преимущества, так и недостатки. В соответствии с предоставленными требованиями, были отвергнуты интерпретируемые языки, так как они существенно медленнее компилируемых. Для достижения стабильности пришлось отказаться от языков с динамической типизацией. В конце концов, список сузился до трех кандидатов: Java, C# и C++. Java была отвергнута ввиду наличия виртуальной машины. C# имеет проблемы с переносимостью. Поэтому мы остановили свой выбор на C++. Это достаточно старый, популярный, объектно-ориентированный язык программирования. C++ позволяет максимально использовать ресурсы компьютера, современные компиляторы генерируют настолько чистый код, что он очень близок к ассемблерной реализации. Таким образом, C++ дает нам возможность максимально использовать аппаратные ресурсы ПК. Также неоспоримым плюсом является наличие огромного количества библиотек для C++, что в какой-то мере упрощает разработку. Естественно не обошлось без препятствий – стандарт C++ не поддерживает многопоточность, но, к счастью, существуют библиотеки позволяющие писать потокобезопасные приложения на C++, в частности библиотека Boost. Эта

библиотека является очень мощным подспорьем при разработке приложений на C++, она имеет множество часто используемых компонентов, и новый стандарт разрабатывается на ее базе, что обеспечивает хорошую совместимость в будущем.

При написании вспомогательных утилит (графического интерфейса, юнит-тестов и т.д.) были задействованы следующие наборы библиотек:

✓ Для написания графического интерфейса для сопутствующих программ, использован Qt – кросс-платформенный инструментарий разработки ПО. Qt позволяет запускать написанное с его помощью ПО в большинстве современных операционных систем путём простой компиляции программы для каждой ОС без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Qt является полностью объектно-ориентированным, легко расширяемым и поддерживающим технику компонентного программирования.

✓ Для построения графиков и анализа тенденций использована библиотека построения графиков и анализа тенденций используется gnuplot – свободная программа для создания двух- и трёхмерных графиков. Gnuplot имеет собственную систему команд, может работать интерактивно (в режиме командной строки) и выполнять скрипты, читаемые из файлов.

Структура пакета

Чтобы обеспечить модульность и легкость изменения кода пакета, пришлось изначально заложить некоторые абстракции. Рассмотрим диаграмму классов пакета PANN (рис. 4):

Рис. 4. Диаграмма связей основных классов пакета

Для упрощения дальнейшей поддержки и развития проекта классы заведомо создавались со слабой связанностью. Все классы унаследованы от общего класса Object, что позволяет унифицировано работать с ними. Для динамического добавления различных временных свойств объектам введен класс AttributeName. Атрибуты позволяют быстро и удобно добавлять данные необходимые для текущих расчетов.

Каждый нейрон содержит набор ссылок на связи и веса, что позволяет работать со всеми нейронами одинаково (будь то входной или выходной слой). Веса отделены от связей для упрощения работы и экономии памяти на топологиях с общими весами (сверточные нейронные сети). Для уменьшения нагрузки процессора используются класс Cache и его наследник NetCache, который содержит информацию о текущем фронте обработки нейронной сети.

Особняком стоит набор классов Runner (рис. 5).

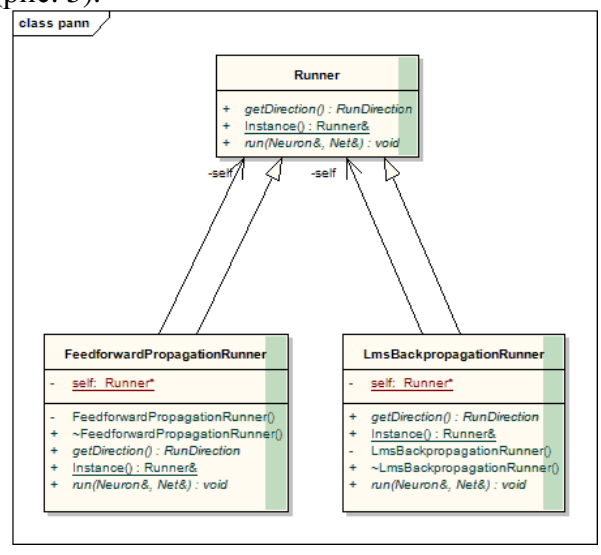

Рис. 5. Иерархия классов Runner

Класс Runner инкапсулирует в себе информацию о методе обучения сети. Как видно на первой диаграмме, класс Net содержит метод run. Именно тут формируется контекст запуска сети, создается необходимое количество потоков и их запуск.

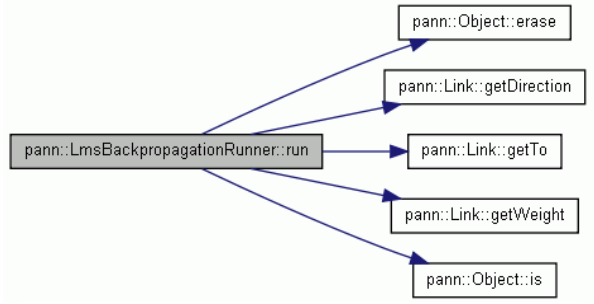

Рис. 6. Диаграмма вызова метода run у объекта LmsBackpropagationRunner

Как видно из рисунка 6, Runner сам вызовет необходимые методы обработки у каждого связи в соответствии с заданным алгоритмом.

Таким образом, мы имеем очень гибкую, слабо связанную систему базовых классов, которую можно с легкостью расширять и дополнять.

Пример работы со сверточной нейронной сетью

Рассмотрим структуру сверточной нейронной сети, топология которой в дальнейшем будет применятся для распознавания образов.

В 1981 году нейробиологи Торстен Визел и Девид Хабел исследовали зрительную кору головного мозга кошки и обнаружили, что существуют так называемые простые клетки, которые особо сильно реагируют на прямые линии под разными углами и сложные клетки, которые реагируют на движение линий в одном направлении.

Позже Ян ЛеКун предложил использовать так называемые сверточные нейронные сети, как аналог зрительной коры головного мозга для распознавания изображений.

Идея сверточных нейронных сетей заключается в чередовании сверточных слоев (C-layers), субдискретизирующих слоев (S-layers) и наличии полносвязных (F-layers) слоев на выходе (рис. 7).

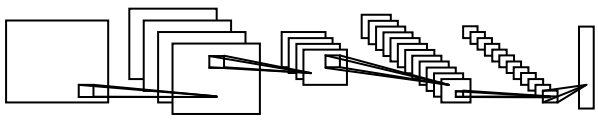

Рис. 7. Архитектура сверточных нейронных сетей

Такая архитектура заключает в себе 3 основных парадигмы:

1. Локальное восприятие
2. Разделяемые веса
3. Субдискретизация

Локальное восприятие подразумевает, что на вход одного нейрона подается не все изображение (или выходы предыдущего слоя), а лишь некоторая его область. Такой подход позволяет сохранять топологию изображения от слоя к слою. Концепция разделяемых весов предполагает, что для большого количества связей используется очень небольшой набор весов. Т.е. если у нас имеется на входе изображение размерами 32x32 пикселя, то каждый из нейронов следующего слоя примет на вход только небольшой участок этого изображения размером, к примеру, 5x5, причем каждый из фрагментов будет обработан одним и тем же набором. Важно понимать, что самих наборов весов может быть много, но каждый из них будет применен ко всему изображению. Такие наборы часто называют ядрами (kernels).

Большинство систем распознавания изображений строятся на основе двумерных фильтров. Фильтр представляет собой матрицу коэффициентов, обычно заданную вручную. Эта матрица применяется к изображению с помощью математической операции, называемой сверткой. Суть этой операции в том, что каждый фрагмент изображения умножается на матрицу (ядро) свертки поэлементно и результат суммируется и записывается в аналогичную позицию выходного изображения. Основное свойство таких фильтров заключается в том, что значение их выхода тем больше, чем больше фрагмент изображения похож на сам фильтр. Таким образом, изображение, свернутое с неким ядром, даст нам другое изображение, каждый пиксель которого будет означать степень похожести фрагмента изображения на фильтр. Иными словами это будет карта признаков.

Разберем более подробно процесс распространения сигнала в С-слое. Каждый фрагмент изображения поэлементно умножается на небольшую матрицу весов (ядро), результат суммируется. Эта сумма является пикселем выходного изображения, которое называется картой признаков. Следует сказать, что в идеале не разные фрагме-

нты проходят последовательно через ядро, а параллельно все изображение проходит через идентичные ядра. Кроме того, количество ядер (наборов весов) определяется разработчиком и зависит от того, какое количество признаков необходимо выделить. Еще одна особенность сверточного слоя в том, что он немного уменьшает изображение за счет краевых эффектов.

Суть субдискретизации и S-слоев заключается в уменьшении пространственной размерности изображения. Т.е. входное изображение грубо (усреднением) уменьшается в заданное количество раз. Чаще всего в 2 раза, хотя может быть и не равномерное изменение, например, 2 по вертикали и 3 по горизонтали. Субдискретизация нужна для обеспечения инвариантности к масштабу.

Чередование слоев позволяет составлять карты признаков из карт признаков, что на практике означает способность распознавания сложных иерархий признаков. Обычно после прохождения нескольких слоев карта признаков вырождается в вектор или даже скаляр, но таких карт признаков становится сотни. В таком виде они подаются на один-два слоя полносвязной сети. Выходной слой такой сети может иметь различные функции активации. В простейшем случае это может быть тангенциальная функция, также успешно используются радиальные базисные функции.

Эксперименты и результаты

Для проведения ряда экспериментов с пакетом была выбрана топология сверточной нейронной сети [5]. Сеть состоит из семи слоев, в которых в общей сложности 5161 нейрон и 132418 связей. В качестве функции активации был выбран тангенс гиперболический.

Эксперимент проводился на персональном компьютере с 8-мю процессорами по 2 гигагерца каждый.

После проведения ряда запусков, были получены усредненные данные (табл. 1):

Табл. 1. Результаты эксперимента

Кол-во потоков	1	2	3	4	5	6	7	8	9	10	11	12
Время (с)	89,8	67,9	55,7	50,8	46,7	44,4	42,1	40,8	43,7	42,4	42,6	42,4

Для упрощения анализа построим график на рисунке 8:

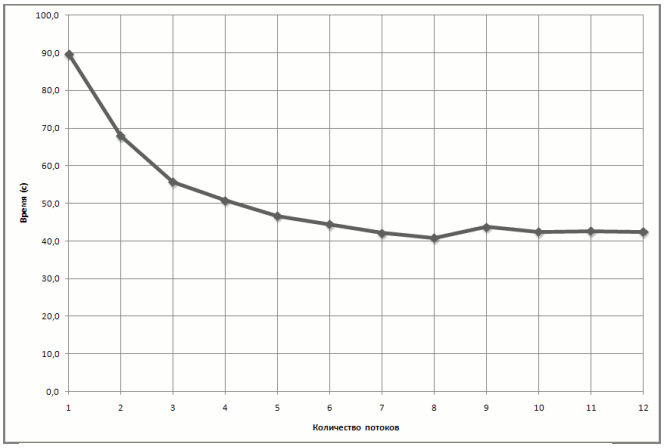

Рис. 8. График зависимости времени обучения сети от количества процессоров

Как видно, с ростом количества задействованных процессоров, время обработки падает. Так продолжается, пока количество потоков не начинает превышать количество доступных процессоров. В таком случае время обработки снова увеличивается, это обусловлено необходимостью переключения потоков между доступными процессорами. Также следует заметить, что скорость обработки от доступных процессоров не линейна - это результат синхронизации потоков с помощью boost::barrier. Больший выигрыш по времени можно получить при еще больших слоях, ведь, как было описано выше, параллельно обрабатывается каждый слой, а поле его обработки необходимо синхронизировать потоки обработки.

Теперь усложним обработку. Возьмем ту же топологию сверточной нейронной сети и проведем обучение алгоритмом обратного распространения ошибки. Получены такие усредненные результаты (табл. 2):

Табл. 2. Результаты эксперимента

Кол-во потоков	1	2	3	4	5	6	7	8	9	10	11	12
Время (с)	241,8	145,1	91,4	68,6	51,4	43,7	35,0	31,5	33,6	32,4	32,7	32,1

График затрат времени обработки при использовании разного количества ядер представлен на рисунке 9:

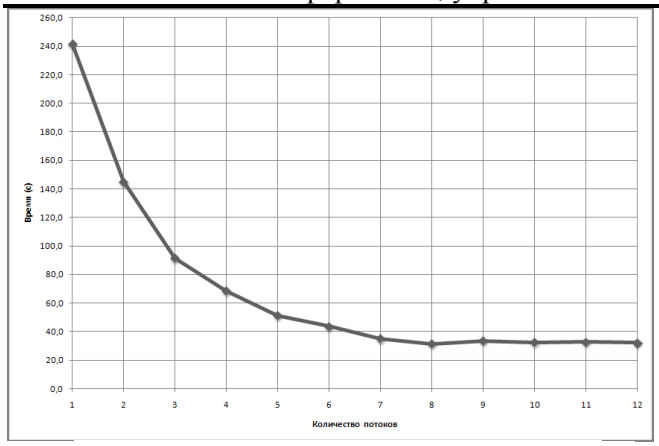

Рис. 9. График зависимости времени обучения сети от количества процессоров

Построенный график схож с графиком первого эксперимента. Но ввиду сложности вычислений для каждого нейрона получен значительно больший выигрыш от многопоточной обработки. При использова-нии 8-ми ядер скорость обработки возросла практически в 8 раз.

Выводы

В ходе разработки пакета PANN были созданы уникальные алгоритмы параллельной обработки нейронных сетей с произвольной топологией. В процессе создания данного пакета разработчики преодолели множество сложностей проектирования и разработки концепции. Пришлось уйти от канонического послойного представления топологии, что позволило проектировать самые сложные и немыслимые топологии. В результате имеем мощный, гибкий и быстрый пакет для проектирования, тестирования и использования нейронных сетей с произвольной топологией.

Список литературы

1. Neural Networks: A Comprehensive Foundation (2nd Edition), Simon Haykin, 842 pages,Prentice Hall; 2 edition (July 16, 1998),ISBN-10: 0132733501

2. Операционные системы: разработка и реализация, Таненбаум Э. С., Вудхалл А. С., 576 стр., 2005 г., изд. Питер, ISBN 5469001482.

3. Алгоритмы: построение и анализ, Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн, 296 стр., с ил.; ISBN 978-5-8459-0857-5, 0-07-013151-1; формат 70x100/16; твердый переплет2008, 4 кв.; Вильямс.

4. Дорогой Я.Ю., Яшин В.Е. Программный комплекс для симуляции многопоточных нейронных сетей.//Вісник НТУУ «КПІ», «Інформатика, управління та обчислювальна техніка», №49. – 2008. – С. 123-127.

5. Gradient-Based Learning Applied to Document Recognition (Proc. IEEE 1998), Yann Lecun, Léon Bottou, Yoshua Bengio, Patrick Haffne.

Поступила в редакцию