УДК 519.85

ТЕХНИЧЕСКИЕ АСПЕКТЫ РАЗРАБОТКИ НЕЙРОСЕТЕВОГО СИМУЛЯТОРА

© О.В. Крючин, А.А. Арзамасцев

Ключевые слова: симуляторы; искусственные нейронные сети; параллельные алгоритмы.
Описана разработка нейросетевого симулятора, который использует параллельные алгоритмы для построения моделей искусственных нейронных сетей. Приведено описание библиотек, используемых для разработки.

ВВЕДЕНИЕ

Имеющиеся на данный момент реализации искусственных нейронных сетей (ИНС) можно разделить на две группы – версии для персональных компьютеров и программно-аппаратные решения. К первой группе относятся такие продукты, как *JavaNNS* (*Tuebingen universitet*), нейросетевые инструменты *Matlab* (*The MathWorks*) и др. Пример реализации второй группы – разработка *Nimfa* (ВНИИТФ). Имеются также различные универсальные инструменты распараллеливания, которые не учитывают специфику построения ИНС-моделей. Примером такого инструмента является Т-Система (Т-Платформа).

Исходя из вышесказанного, на данный момент не существует нейросетевых симуляторов для кластерных систем. По этой причине следует считать актуальной разработку такого симулятора, реализующего параллельные алгоритмы построения ИНС-моделей, который может работать на кластерных вычислительных системах. Таким образом, целью данной работы является разработка нейросетевого симулятора для вычислительных кластеров.

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ

Обучение ИНС сводится к минимизации значения невязки

$$\varepsilon = \frac{1}{N}\sum_{i=0}^{N-1}\left(\vec{d}_i - \vec{y}_i\right)^2 = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{P-1}\left(d_{i,j} - y_{i,j}\right)^2, \quad (1)$$

где $\vec{d}_i$, $\vec{y}_i$ – i-е выходные значения моделируемого объекта и ИНС; N – число строк в обучающей выборке; P – число выходов объекта (размерность векторов $\vec{d}_i$ и $\vec{y}_i$).

Выходные значения ИНС рассчитываются по формуле:

$$\vec{y}_i = F(\vec{x}_i, \vec{w}, \vec{\mu}). \quad (2)$$

Здесь $\vec{x}_i$ – входные данные, а $\vec{w}$ и μ – управляющие параметры – значения весовых коэффициентов и активационных функций нейронов.

Объединив формулы (1) и (2), можно получить

$$\varepsilon = \frac{1}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{P-1}\left(d_{i,j} - F(\vec{x}_i, \vec{w}, \vec{\mu})\right)^2. \quad (3)$$

Как можно увидеть из рис. 1, на котором изображена блок-схема алгоритма обучения сети, оно состоит из нескольких уровней:

– подбор структуры (определяются размерности векторов μ и $\vec{w}$ (зависит от размерности вектора μ)) – как правило, вначале устанавливается минимально-возможное количество нейронов, и затем в процессе обучения добавляются новые;

– подбор состояний (активационных функций) нейронов (элементов вектора μ) – как правило, полный перебор всех возможных вариантов);

– подбор значений весовых коэффициентов (элементов вектора $\vec{w}$).

Для применения распараллеливания были выбраны все перечисленные выше уровни, а также добавлен уровень «вычисления значения целевой функции» [1, 2].

ИСПОЛЬЗУЕМЫЕ СРЕДСТВА

Для реализации нейросетевого симулятора необходим ряд инструментов, таких, как анализатор *XML*, библиотека графического интерфейса, средства межпроцессороной передачи данных.

Анализаторы XML. Разработка *XML* началась в 1996 г. и была стандартизирована организацией *W3C* (*World Wide Web Consortium*). Существует несколько моделей анализа *XML*-документов, самые известные из которых *DOM* и *SAX* [3, 4].

При использовании *DOM* (*Document Object Model*) строится дерево документа, предоставляющее приложению возможность навигации. Средства *DOM* предназначены для работы с различными языками, в частности, *Python*, *Perl*, *C++* и *Java*. Кроме того, реализация *DOM* входит в состав системы *Microsoft Windows* [5]. Недостатком этой модели является большой объем используемой памяти.

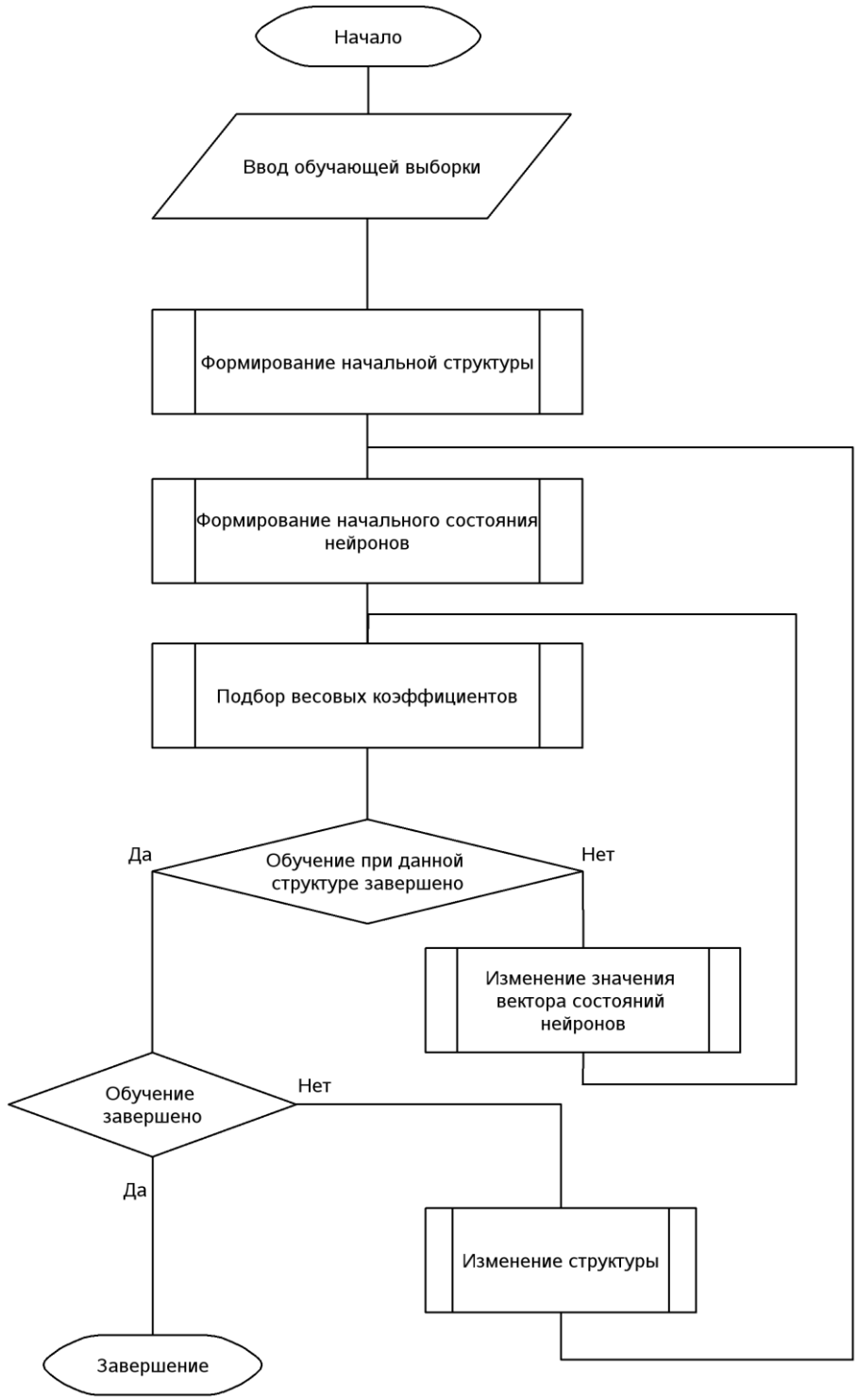

Рис. 1. Блок-схема алгоритма обучения ИНС

SAX (*Simple API for XML*) является стандартом *JavaAPI* для считывания *XML*-документов. *SAX* применяется для последовательного считывания *XML*-данных, что позволяет без проблем работать с очень большими файлами [4]. Реализации *SAX* также имеются для различных языков. Недостатком модели является невысокая, по сравнению с *DOM*, скорость анализа.

На данный момент существует множество библиотек анализатора *XML*, работающих с технологиями *DOM, SAX, SAX2* и т. д., наиболее известные из них представлены в табл. 1 [6].

Для разработки нейросетевого симулятора была выбрана модель *DOM*, а в качестве анализатора – библиотека *expat*. Этот выбор обусловлен тем, что данная

Таблица 1

Сравнительные характеристики анализаторов XML

Имя библиотеки	Домашняя страница	Назначение
TinyXml	*www.grinninglizard.com/tinyxml*	*DOM*
Xerxes	*xml.apache.org/xerces-c*	*SAX2, DOM*
Xalan	*xml.apache.org/xalan-c*	*XSLT, XPath*
Pathan	*software.decisionsoft.com/pathanintro*	*XPath*
Boost.Serialisation	*www.boost.org/libs/serialisation*	*Сериализация XML*
Expat	*www.sourcefourge.org/expat*	*DOM*

библиотека является самой распространенной и входит в состав большинства дистрибутивов *GNU/Linux* и *BSD Unix*.

Потоки. Разработка технологии многопотокового программирования началась в середине 1980-х гг. и изначально не была согласована. Поставщики различных вариантов ОС *UNIX* предлагали различные версии интерфейсов многопоточного программирования. Позднее комитет *POSIX* разработал комплект многопотоковых *API*, который вошел в стандарт *POSIX.1c* [5]. На данный момент *POSIX*-потоки наряду с потоками *Sun* являются наиболее распространенными в мире *UNIX*.

Первая подсистема потоков в *Linux* появилась в 1996 г. и называлась *LinuxThreads*. Подсистема потоков *Linux* стремится соответствовать требованиям стандартов *POSIX*, так что новые многопоточные приложения *Linux* должны без проблем компилироваться в *POSIX*-совместимых системах [6–9].

Типичный процесс можно представить как имеющий единственный поток управления – каждый процесс в один момент времени решает только одну задачу. При использовании нескольких потоков управления мы можем спроектировать приложение таким образом, что оно будет решать одновременно несколько задач в рамках единственного процесса, где каждый поток решает отдельную задачу [10, 11].

В *Linux* каждый поток является процессом, и, для того чтобы создать новый поток, нужно создать новый процесс. Эти процессы представляют собой обычные дочерние процессы главного процесса, но они разделяют с главным процессом адресное пространство, файловые дескрипторы и обработчики сигналов. Для обозначения процессов этого типа применяется специальный термин – легкие процессы (*lightweight processes*). Прилагательное легкий в названии процессов-потоков вполне оправдано. Поскольку этим процессам не нужно создавать собственную копию адресного пространства (и других ресурсов) своего процесса-родителя, создание нового легкого процесса требует значительно меньших затрат, чем создание полновесного дочернего процесса [12, 13].

Библиотеки графического интерфейса. На сегодняшний день существует множество библиотек графического интерфейса, сравнительные характеристики которых приведены в табл. 2.

Таблица 2

Сравнительные характеристики библиотек графического интерфейса

Библиотека	Производитель	Языки	Платформы	Поддерживается
VCL	*Embarcadero Technologies*	*Delphi, C++*	*MS Windows*	Да
CLX	*Borland*	*Delphi, C++*	*MS Windows, GNU/Linux*	Нет
MFC	*Microsoft*	*C++*	*MS Windows*	Да
GTK+	*GNOME Foundation*	*C*	*MS Windows, GNU/Linux, BSD, MacOS*	Да
QT	*Nokia (Trolltech)*	*C++, Python*	*MS Windows, GNU/Linux, BSD, MacOS*	Да
KDE	*Ассоциация KDE*	*C++*	*MS Windows, GNU/Linux, BSD, MacOS*	да
xwWidgets	*Джулиан Смарт*	*C++*	*MS Windows, GNU/Linux, BSD, MacOS*	Да
AWT	*Oracle (Sun Microsystem)*	*Java*	*MS Windows, GNU/Linux, BSD, MacOS*	Да
Swing	*Oracle (Sun Microsystem)*	*Java*	*MS Windows, GNU/Linux, BSD, MacOS*	Да

Для разработки нейросетевого симулятора была выбрана библиотека *QT*, т. к. она обеспечивает кроссплатформенность, имеет удобные средства разработки, а также поддерживается используемым компилятором (*gcc*).

Изначально *QT* была задумана в качестве моста между *API* функциями *Motif* (*UNIX*) и *WinApi* (*Microsoft Windows*). На сегодняшний день *Qt* – это продукт, широко используемый разработчиками всего мира. Компаний, использующих эту библиотеку, более 4000. Из числа некоторых активных пользователей *Qt* можно назвать такие известные компании, как: *Adobe*, *AT&T*, *Cannon*, *HP*, *Bosch*, *Boeing*, *IBM*, *Motorola*, *NASA*, *NEC*, *Pioneer*, *Sharp*, *Siemens*, *Sony*, *Xerox* и др. [4, 14–15]. Наиболее известные программы, разработанные на основе *QT*, это:

– рабочий стол *KDE* (*K Desktop Environment*), используемый в *GNU/Linux* и *FreeBSD* (*www.kde.org*);

– веб-браузер *Opera* (*www.opera.com*);

– средство голосовой связи *VoIP Skype*, предназначенное для звонков на обычные телефоны и проведения видеоконференций через Интернет (*www.skype.com*);

– программа обработки растровых изображений *Adobe Photoshop* (*www.adobe.com*);

– сетевая карта мира *Google Earth* (*earth.google.com*).

Библиотека *QT* не ограничивается визуальными компонентами. Кроме того, она предоставляет средства работы с сетью, *XML*, потоками и многими другими технологиями [16].

Средство межпроцессорной передачи данных. Наиболее высокопроизводительными машинами на данный момент являются суперкомпьютеры (многопроцессорные машины, использующие общую память) и кластерные системы (вычислительные узлы – однопроцессорные или многопроцессорные компьютеры, связанные в вычислительную сеть, в которой в качестве интерконекта обычно применяется *Fast Ethernet*). Поскольку кластерные системы используют распределенную память, то для взаимодействия частей программы, работающих на разных узлах, необходим какой-либо механизм. До 1994 г. различные производители кластеров (*IBM*, *HP*, *Sun* и др.) создавали свои инструменты передачи сообщений, предназначенные для конкретной аппаратной архитектуры. Стремясь достигнуть максимальной производительности, производители кластеров адаптировали свои инструменты для конкретных моделей, что приводило к невозможности переносить программы между кластерами.

В 1994 г. был принят стандарт механизма передачи сообщений *MPI* (*Message Passing Interface*). Он готовился с 1992 по 1994 гг. группой *Message Passing Interface Forum*, в которую вошли представители более чем 40 организаций из Америки и Европы. Основная цель, которую ставили перед собой разработчики *MPI*, – это обеспечение полной независимости приложений, написанных с использованием *MPI*, от архитектуры многопроцессорной системы, без какой-либо существенной потери производительности. По замыслу авторов это должно было стать мощным стимулом для разработки прикладного программного обеспечения и стандартизованных библиотек подпрограмм для многопроцессорных систем с распределенной памятью. Подтверждением того, что эта цель была достигнута, служит тот факт, что в настоящее время этот стандарт поддерживается практически всеми производителями многопроцессорных систем. Реализации *MPI* успешно работают не только на классических *MPP* системах (многопроцессорных системах с массовым параллелизмом), но также на *SMP* системах (симметричных мультипроцессорных системах) и на сетях рабочих станций (в т. ч. и неоднородных) [17]. Таким образом, на данный момент большая часть кластерных систем поддерживает данный механизм.

MPI поддерживает работу с языками *C* и *Fortran*. Полная версия интерфейса содержит описание более 120 функций и поддерживает создание параллельных программ в стиле *MIMD*, что подразумевает объединение процессов с различными исходными текстами. Однако на практике программисты чаще используют *SPMD*-модель, в рамках которой для всех параллельных процессов используется один и тот же код. В настоящее время все больше и больше реализаций *MPI* поддерживают работу с нитями [18].

ВЫСОКОУРОВНЕВАЯ БИБЛИОТЕКА МЕЖПРОЦЕССОРНОЙ ПЕРЕДАЧИ ДАННЫХ НА ОСНОВЕ MPI

MPI является мощной, но вместе с тем низкоуровневой технологией. Разрабатывая программу на каком-либо языке высокого уровня, например, *C++* (который используется для разработки нейросетевого симулятора), при использовании *MPI*, программист вынужден писать фактически на *ANSI C*. Например, для передачи массива встроенного типа, такого как *int* или *double*, необходимо выполнить ряд подготовительных действий перед отправкой на посылающем узле, произвести отправку, на принимающем узле также вызвать ряд подготовительных функций, затем отдельно принять длину массива и лишь затем сам массив. Наличие такого количества действий увеличивает время разработки и способно привести к появлению ошибок, которые в силу особенностей структуры программ обнаружить труднее, чем в однопроцессорных архитектурах. Кроме того, *MPI* позволяет передавать лишь массивы встроенных типов и при необходимости передать пользовательский тип. Его необходимо разложить на простые типы, а после принятия снова собрать. Также *MPI* требует указывать тип передаваемых данных, что делает невозможным передачу параметризованных типов, широко используемых программистами *C++*, затрудняя тем самым позднее связывание программ.

Для устранения перечисленных выше недостатков была реализована библиотека межпроцессорной передачи данных. Эта библиотека представляет собой высокоуровневую оболочку над функциями *MPI*, что делает работу с данной технологией более удобной. Данная оболочка обладает рядом преимуществ по сравнению со стандартными функциями:

– возможность избежать низкоуровневой работы (отслеживать размерности передаваемых массивов, освобождать память и т. д.);

– возможность передачи параметризованных типов;

– возможность передачи пользовательских типов.

ВЫВОДЫ

Все вышеперечисленные библиотеки, за исключением *QT*, предназначены для *POSIX*-совместимых операционных систем. Поэтому в реализации нейросетевого симулятора для платформы *Microsoft Windows* библиотеки *pthread* и *expat* заменяются средствами *QT* (в случае *POSIX*-совместимых систем библиотеки *pthread* и *expat* используются по той причине, что на вычислительных кластерах *QT* может отсутствовать). Библиотека *MPI* для платформы *Microsoft Windows* отсутствует вовсе, что обусловливает наличие для этой операционной системы только последовательной версии нейросетевого симулятора. Таким образом, параллельная версия нейросетевого симулятора может работать только на платформах *nix* (*GNU/Linux*, *BSD*, *Solaris* и т. д.).

ЛИТЕРАТУРА

1. *Крючин О.В.* Разработка алгоритмов обучения искусственных нейронных сетей, использующих параллельное вычисление целевой функции // Известия Смоленского государственного университета. Смоленск, 2010. Ежеквартальный журнал. № 2 (10). С. 126-134.
2. *Крючин О.В.* Разработка параллельных градиентных алгоритмов обучения искусственной нейронной сети // Электронный журнал «Исследовано в России». 2009. 096. С. 1208-1221. URL: http://zhurnal.ape.relarn.ru/articles/2009/096.pdf. Загл. с экрана.
3. *Наварро Э.* XHTML: учебный курс / Ann Navarro. XHTML by Example; пер. с англ. И. Синицын. СПб.: Питер, 2001. 336 с.: ил.
4. *Шлее М.* Qt4. Профессиональное программирование на C++. СПб.: БХВ-Петербург, 2007. 880 с.: ил. + CD-ROM. (В подлиннике).
5. *Ленди М., Сиддикви С., Свишер Д.* Borland JBuilder. Руководство разработчика: пер. с англ. М.: Издат. дом «Вильямс», 2004. 864 с.: ил. Парал. тит. англ.
6. *Стефенс Д.Р., Диггинс К., Турканис Д., Когсуэлл Д.* C++. Сборник рецептов / пер. с англ. О. Босис, В. Казаченко. М.: Кудиц-Пресс, 2007. 624 с.

7. *Чан Т.* Системное программирование на C++ для UNIX // UNIX System Programming Using C++. Terrence Chan / пер. с англ. С. Тимпчева; под ред. М. Коломыцева. BHV. К., 1997.
8. *Боровский А.* Программирование для Linux. СПб., 2007.
9. *Рочкинд М.Д.* Программирование для UNIX / пер. с англ. под общ. ред. В.Г. Вшивцева. (Rochkind M.J. Advanced Unix Programming). СПб.: БХВ-Петербург, 2005.
10. *Стивенс У.* UNIX: взаимодействие процессов / пер. с англ. Д. Солнышков. СПб.: Питер, 2003. 576 с.: ил.
11. *Стивенс Р., Рago С.* UNIX. Профессиональное программирование / пер. А. Киселева. 2-е изд. СПб.: Символ-Плюс, 2007. 1040 с.: ил.
12. *Bovet D.P., Cesati M.* Understanding the Linux Kernel. 3-rd ed. Sebastopol, CA, 2005.
13. *Stevens W.R., Rago S.A.* Advanced Programming in the UNIX R Environment: 2-nd ed. Westford, Mass., 2005.
14. *Бланшет Ж., Саммерфильд М.* QT4: программирование GUI на C++. М.: Кудиц-пресс, 2007.
15. *Секунов Н.* Программирование на C++ в Linux. СПб.: БХВ-Петербург, 2004. 368 с.: ил.
16. *Шлее М.* Qt4. Профессиональное программирование на C++. СПб.: БХВ-Петербург, 2005. 544 с.: ил.
17. *Букатов А.А., Дацюк В.Н., Жегуло А.И.* Программирование многопроцессорных вычислительных систем. Ростов н/Д: Изд-во ООО ЦВВР, 2003. 208 с.
18. *Антонов А.С.* Введение в параллельные вычисления: метод. пособие / Московский государственный университет им. М.В. Ломоносова. Научно-исследовательский центр. М., 2002.

Поступила в редакцию 20 ноября 2013 г.

Kryuchin O.V., Arzamastsev A.A. TECHNICAL ASPECTS OF DEVELOPMENT OF NEURAL NETWORK SIMULATOR

The development of neural network simulator that uses parallel algorithms for constructing models of artificial neural networks is described. The description of libraries used for development is given.

Key words: simulators; artificial neural networks; parallel algorithms.

Крючин Олег Владимирович, Тамбовский государственный университет им. Г.Р. Державина, г. Тамбов, Российская Федерация, магистрант по направлению подготовки «Прикладная математика и информатика» института математики, физики и информатики, e-mail: kryuchov@gmail.com

Kryuchin Oleg Vladimirovich, Tambov State University named after G.R. Derzhavin, Tambov, Russian Federation, Tambov State University named after G.R. Derzhavin, Tambov, Russian Federation, Candidate for Master's Degree of Direction of Preparation of "Applied Mathematics and Informatics" of Mathematics, Physics and Informatics Institute, e-mail: kryuchov@gmail.com

Арзамасцев Александр Анатольевич, Тамбовский государственный университет им. Г.Р. Державина, г. Тамбов, Российская Федерация, доктор технических наук, профессор, зав. кафедрой компьютерного и математического моделирования, e-mail: arz_sci@mail.ru

Arzamastsev Alexander Anatolyevich, Tambov State University named after G.R. Derzhavin, Tambov, Russian Federation, Doctor of Technics, Professor, Head of Computer and Mathematical Simulation Department, e-mail: arz_sci@mail.ru