УДК 004.93’11

А.А. Южаков, В.Е. Щавлев

Пермский национальный исследовательский
политехнический университет

МОДЕЛИРОВАНИЕ НЕЙРОСЕТИ С ИСПОЛЬЗОВАНИЕМ ВОЗМОЖНОСТЕЙ ОБЪЕКТНО-ОРИЕНТИРОВАННЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Описывается подход к исследованию работы нейронных сетей с помощью средств объектно-ориентированных языков программирования (в частности, C++). Приведены кратная инструкция по установке и подключению нейронной библиотеки в среде GNU/Linux, описание программной реализации задачи распознавания рукописного ввода с помощью нейросети.

Архитектура вычислительных устройств, работающих по принципам нейронной логики, в корне отличается от архитектуры наиболее распространённых в настоящее время персональных компьютеров. Поэтому многие исследователи используют программные решения, позволяющие смоделировать нейросеть на обычном ПК фоннеймановской архитектуры [1]. Одним из подобных инструментов являются объектно-ориентированные языки программирования. Гибкость реализованной в них парадигмы, наличие универсальных компиляторов, доступность прикладных средств обеспечивают неограниченный простор для деятельности. При современном уровне развития нейронауки необходимо пользоваться данным инструментарием, поскольку он позволяет существенно снизить затраты на исследования как временные, так и финансовые.

Здесь также следует выделить такую полезную особенность объектно-ориентированного программирования, как агрегирование созданных структур и функций в обособленные продукты, которые в дальнейшем можно использовать при разработке прикладных программ. Речь идёт о библиотеках. Сегодня разработчикам доступно довольно обширное множество нейросетевых библиотек [2], рассмотрим подробнее одну из наиболее удачных, по мнению автора, разработок данного назначения под названием *Fast Artificial Neural Network (FANN)* [3].

Описание исследуемого продукта.

Данная библиотека является свободно-распространяемым программным продуктом (лицензия *GPL*). Автор проекта – Штеффен Ниссен (*Steffen Nissen*), магистрант кафедры информатики Университета Копенгагена. Первая версия библиотеки *FANN* – это продукт магистерской диссертации Штеффена, которую он защитил в 2003 г. Эта диссертация [4] главным образом и является самой подробной документацией к продукту.

Проект создавался с использованием окружения операционной системы *GNU/Linux*, языка *C* и компилятора *gcc*, поэтому предпочтительнее при работе с ним применять именно эти продукты. Но стоит отметить, что разработчики, поддержавшие проект Штеффена Ниссена, не ограничились данной операционной средой и языком программирования – в дальнейшем библиотека была портирована для работы под управлением *Windows* (с использованием *Cygwin*), *MacOS*, а также существуют реализации на нескольких языках программирования: это *C++, Java, Python, PHP, Matlab* и другие [3], что свидетельствует об относительно высоком интересе к библиотеке со стороны разработчиков по всему миру. Помимо этого в качестве положительных особенностей проекта можно отметить наличие нескольких инструментов, реализующих графический интерфейс для конструирования и визуализации нейросетей, создаваемых с применением данной библиотеки, а также ведение разработок в направлении адаптации библиотеки для работы с использованием ресурсов видеоускорителей, современные представители которых, как известно, оснащаются мощными вычислительными процессорами с параллельной архитектурой, которые можно задействовать в качестве аппаратной основы для реализации вычислительных возможностей нейросетей [5].

Установка FANN в среде *GNU/Linux*

Для того чтобы установить библиотеку, необходимо откомпилировать её исходный код, предварительно собрав и настроив проект:

– с помощью командной консоли перейти в директорию, где находятся файлы с текстом исходных кодов (**.c*, **.h*), а также файл линковщика *makefile*;

– настроить пакет с исходным кодом для установки в систему, запустив на выполнение команду *./configure*;

– выполнить последовательность, описанную в *makefile*, командой *make* – в результате будут произведены сборка и компиляция

исходного кода и получены файлы динамических и статических библиотек, которые необходимо подключать к приложениям, созданным с использованием библиотеки *FANN*;

– установить полученные файлы в систему командой *make install* (для выполнения этого действия потребуются права суперпользователя).

При использовании дистрибутивов *Debian* или *Ubuntu*, подключенных к своим репозитариям, для установки библиотеки достаточно выполнить команду

apt-get install libfann2 libfann2-dev

Использование библиотеки при разработке ПО

После успешного выполнения перечисленных выше действий библиотека готова к работе. Для использования возможностей, предоставляемых ей, достаточно выполнить следующее:

– подключить необходимые заголовочные файлы библиотеки в исходном коде программы, с помощью которой выполняется создание модели нейросети;

– подключить библиотеку при запуске компилятора.

При использовании компилятора *gcc* и языка программирования *C* первое правило выполняется добавлением в файл с исходным кодом директивы

#include "fann.h"

Для выполнения второго условия компилятор нужно вызывать следующим образом:

gcc main.c -omainbin -lfann -lm

где *main.c* – файл с исходным кодом вашей программы;

– *omainbin* – имя исполняемого файла;

– *lfann* – параметр, сообщающий компилятору о необходимости подключения внешней библиотеки.

Пример использования *FANN* в разработке

В процессе исследования возможностей *FANN* была разработана программа распознавания рукописного ввода символов. Приложение реализовано с использованием библиотеки *Qt* и состоит из следующих базовых элементов:

1. Окно ввода символа

Представляет собой поле ввода размером 200×200 точек, на котором с помощью курсора мыши выполняется рисование символа.

2. Окно управления работой программы.

Выполняет следующие функции:

– переключение режимов работы программы: формирование массива обучения, непосредственное распознавание вводимых символов;

– запуск процесса обучения сети;

– сохранение массива обучения, параметров нейросети в файлы;

– загрузка из файлов массива обучения, параметров нейросети.

Функции или типы данных нейронной библиотеки в фрагментах исходного кода программы можно отличить от остальных по символам «*fann_*» в начале названия функции, описание их работы можно найти в документации библиотеки и в файлах с исходным кодом (рис. 1).

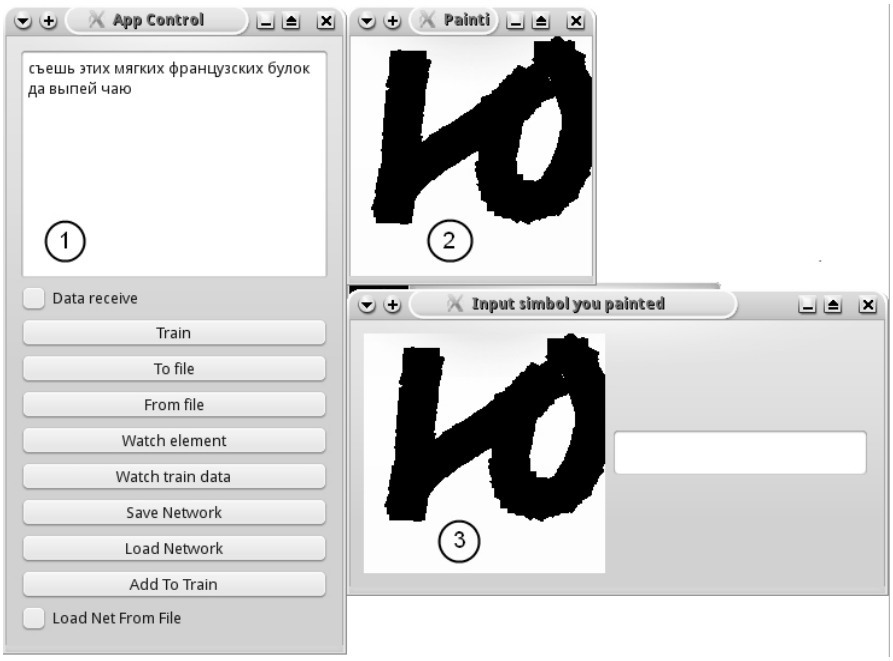

Рис. 1. Интерфейс программы

Пояснения к рис. 1: *1* – текстовое поле вывода распознанных символов; *2* – поле рисования; *3* – окно ввода символа, соответствующего рисунку.

Назначение кнопок и чекбоксов окна управления работой программы:

Data receive – выбор режима обработки содержимого поля рисования: создание обучающей выборки или распознавание нарисованного символа;

Train – запуск процесса обучения;

To file – запись обучающей выборки в файл;

From file – чтение обучающей выборки из файла;

Watch element – просмотр матрицы элемента обучающей выборки;

Watch train data – просмотр всей матрицы обучающей выборки;

Save network – сохранение конфигурации сети в файл;

Load network – создание новой сети;

Load Net From File – выбор режима создания новой сети: загрузка конфигурации из файла или создание «чистой» сети.

Создание сети

Цель – создать нейросеть для распознавания рукописных символов *Unicode*, вводимых с помощью инструмента рисования, в данном случае курсора мыши. Процесс рисования представляет собой закрашивание черным цветом пикселей поля ввода с белым фоном.

Поскольку размер поля ввода равен 200×200 точек, то входной слой нейронной сети должен содержать 40 000 нейронов, каждый из которых обрабатывает значение степени насыщенности пикселя серым цветом от 0 до 255, масштабированное в пределах от 0 до 1. Таким образом необходимо создать массив, который бы содержал в себе эти значения. В коде программы данное действие описывается так:

```
*pixmap = QPixmap::grabWidget(this);
*image = pixmap->toImage();

int pixinfo_index = 0;

for (i=0; i < 200; i++)
{
  for (j=0; j < 200; j++)
  {
  pixinfo[pixinfo_index] =
(float)(qGray(image->pixel(i,j))+1)/255;
  pixinfo_index++;
  }
}
```

На данном этапе необходимо отметить, что количество нейронов во входном слое сети трудно назвать оптимальным. Забегая вперёд, следует отметить, что для построения такой сети требуется неоправданно большое для подобного приложения количество памяти

ОЗУ – порядка 500 МБ, кроме того, довольно много дискового пространства отводится для хранения массива обучения. Поэтому целесообразно произвести оптимизацию к подходу считывания изображения нейросетью: анализировать не каждый пиксель, а делить изображение на сегменты произвольного размера и оперировать усреднённым значением интенсивности в каждом сегменте. Выбор для анализа сегментов размером 10×10 пикселей позволяет уменьшить количество нейронов в 100 раз. В итоге сеть «видит» символы так, как представлено на рис. 2.

Выходной слой состоит из 45 нейронов, каждый из которых соответствует букве алфавита или цифре (точнее, её коду в таблице *Unicode*), т.е. если значение на выходе одного из этих нейронов близко к 1, то это событие соответствует успешному распознаванию сопоставленного символа.

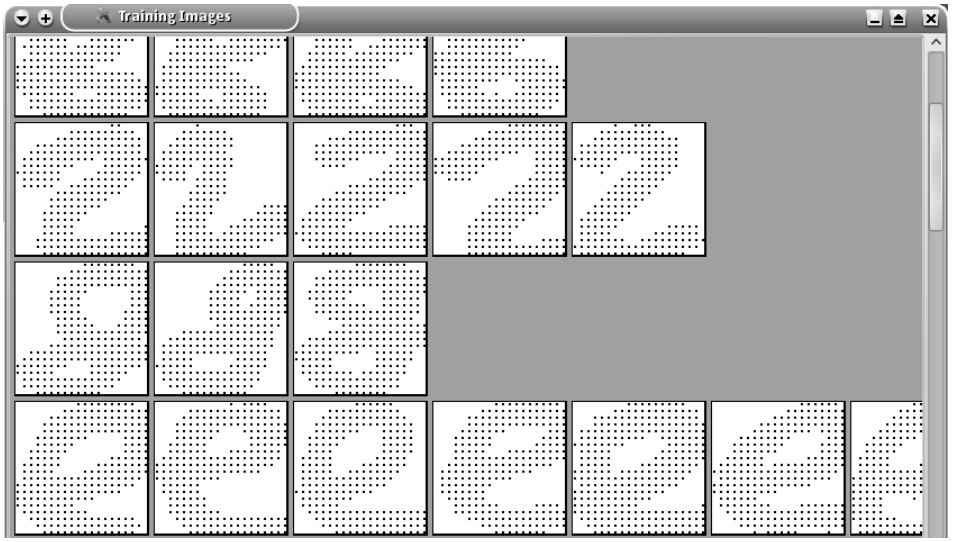

Рис. 2. Массив обучения

Далее следует процесс конструирования нейронной сети – определение количества скрытых слоёв и количества нейронов в них, но поскольку целью статьи является демонстрация работы сети как таковой, а не нахождение оптимальной топологии, то в данном случае эмпирически в сеть был добавлен один скрытый слой, содержащий 1000 нейронов.

Таким образом, для решения поставленной задачи необходимо создать 3-слойную сеть с 400 нейронами в первом слое, 1000 – во

втором, 45 – в третьем выходном слое. Данное действие описывается следующим образом:

```
void PaintWidget::loadNetwork()
{
  fann_destroy(ann); //удаление созданной нейросети

  ann = new struct fann;  //создание модели нейросети
//в куче
  if (loadNetFromFile->isChecked()) //если чекбокс
//"загрузить сеть из файла" активен
    ann = fann_create_from_file("ann.net");
     //создать сеть функцией загрузки из файла
  else
    ann = fann_create_standard(3, 400, 1000, 45);
     //создать "чистую" сеть
  fann_set_bit_fail_limit(ann, 0.1);
     //установить порог ошибки выхода равным 0.3
}
```

В результате в оперативной памяти формируется модель нейросети с алгоритмом обратного распространения ошибок, в которой каждый нейрон связан со всеми нейронами предыдущего слоя. Стоит отметить, что в приведённом коде предусмотрено не только создание «чистой» сети, но и при необходимости загрузка сконфигурированной обученной сети из файла, предварительно созданного с помощью функции:

```
fann_save(ann,"ann.net");
```

Обучение организуется так:

```
void PaintWidget::learn_net()
{
    for(int epoch = 1; epoch <= 1000; epoch++)
    {
      fann_reset_MSE(ann);     //сбросить количество
//битов с ошибкой
      for (int i=0; i < data.size(); i++)
      {
          fann_train(ann, data[i].net_input,
data[i].net_output);     //один цикл обучения
      }
      std::cout << epoch << " - " <<
fann_get_bit_fail(ann) << "\n" << std::flush;
```

```
     //отобразить текущее количество ошибочных
//выходов в стандартном выводе
      if (fann_get_bit_fail(ann) == 0) break;
     //если битов с ошибкой нет, остановить обучение
    }
}
```

Условием окончания обучения является отсутствие ошибок на выходе нейронной сети после обработки всех обучающих выборок, которые представляют собой массив из 400 элементов, каждый из которых соответствует усреднённому значению степени интенсивности чёрного цвета для определённого сегмента поля ввода.

После обучения сеть готова к работе, программа переводится в режим распознавания и далее работает в соответствии с приведённым ниже кодом:

```
pOutput = fann_run(ann, pixinfo);   //выполнение
//сетью ann операции над массивом pixinfo с записью
//результата (состояния выходного слоя) в массив по
//адресу, хранимому в указателе pOutput.
for (i=0; i < num_of_simb; i++)
{
std::cout << pOutput[i] << " " << std::flush;

if (pOutput[i] > 0.7)
{
int shift;
if (lat)  //буквы латиницы
shift = 97;

if (cyr)  //буквы кириллицы
shift = 1072;

if (i == 35)   //пробел
shift = -3;

QChar str(i+shift); //фомирование символа из
//таблицы Unicode
resultOutput->insertPlainText(str);
}
}
```

Работа программы

Влияние размера нейросети на ее функциональные параметры

Критерий сравнения	Количество нейронов во входном слое	
	40000	400
Объём занимаемой памяти ОЗУ	~ 500 MB	~10 MB
Длительность обучения	~ 10 мин	~ 15 с
Среднее количество шагов обучения	300	30

В итоге был получен не только прототип программного продукта, решающего проблему распознавания машиной рукописного текста человека, но и некий стенд для изучения возможностей нейросетей при распознавании изображений в зависимости, например, от топологии, вида алгоритма обучения, пороговой функции и т.д.

В качестве заключения стоит выделить основные преимущества использования объектно-ориентированных языков программирования при моделировании нейронных сетей. Это гибкость и возможность адаптации к любой аппаратной платформе, интеграция с другими библиотеками (в приведённом примере наглядно демонстрируется совместное использование библиотеки *FANN* и библиотеки графических интерфейсов *Qt*), а также унифицированность и вседоступность полученных моделей.

Библиографический список

1. Gael de La Croix Vaubois, Catherine Moulinoux, Benolt Derot, The N Programming Language // Neurocomputing, NATO ASI series. – Vol.F68. – P. 89–92.

2. URL: http://directory.fsf.org/wiki/Category/Science/artificial-intelligence

3. URL: http://leenissen.dk/fann/wp/

4. Steffen Nissen. Implementation of a Fast Artifitial Neural Network Library. – DIKU, 2003.

5. Аляутдинов М.А. Использование графических процессоров с параллельной архитектурой для построения масштабируемых нейрокомпьютерных конфигураций // Нейрокомпьютеры: разработка, применение. – 2006. – № 8–9. – С. 18–27.

Получено 06.09.2011