Программные нейронные сети. Нейронная сеть Хопфилда

Матвей Андриенко, 10 „B“

Данная работа дает краткий обзор существующих разновидностей искусственных программных нейронных сетей. Рассматриваются особенности их архитектуры, возможности и способы применения. Подробно затронут вопрос написания простейшей нейронной сети — сети Хопфилда — и представлена ее программная реализация на языке программирования C++.

В последнее время все чаще классические методы обработки информации рассматриваются как простейшие реализации универсальных способов функционирования биологических объектов. Это явление получило название *биологизации* кибернетики. С другой стороны, из существующего соглашения о том, что любая искусственная модель реального объекта всегда будет примитивнее и проще оригинала, следует, что для получения наиболее полной картины следует изучать объект с использованием различных методов и последующей интеграцией результатов. Такой подход носит название *гибридизации.* Удачной иллюстрацией названных тенденций является одна из новых прикладных областей математики, специализирующаяся на нейронных сетях.

Этот реферат дает короткий обзор архитектуры, возможностей и существующих разновидностей программных нейронных сетей. При этом особый упор делается на рассмотрение нейронных сетей Хопфилда, как наиболее простых для реализации.

Область применения нейронных сетей

Выполняемые сетью функции можно разделить на несколько основных групп:

1. аппроксимации и интерполяции,
2. распознавания и классификации образов,
3. сжатия данных,
4. прогнозирования,
5. идентификации,
6. управления,
7. ассоциации.

Традиционно нейронные сети применяются для распознавания образов (в широком смысле этого слова, т. е., например, как для обнаружения лиц на фотографиях, так и для распознавания текстов) и для сжатия информации. В последнее время, однако, нейронные сети все чаще применяются в задачах прогнозирования и управления. На фондовых рынках успешно действуют боты, написанные с применением нейронных сетей, принимающие решения о купле и продаже на основании значений индексов.

Общие черты нейронных сетей

Широкий круг задач, решаемых нейронными сетями, не позволяет создавать мощные универсальные сети из-за ограниченности вычислительных ресурсов. И хотя разработчики вынуждены соз-

давать специализированные нейронные сети, функционирующие по различным алгоритмам, эти отдельные типы сетей обладают несколькими общими чертами.

Во-первых, основу каждой нейронной сети составляют относительно простые однотипные ячейки, имитирующие работу нервных клеток (нейронов) человеческого организма, которые в некоторых случаях объединяют в слои:

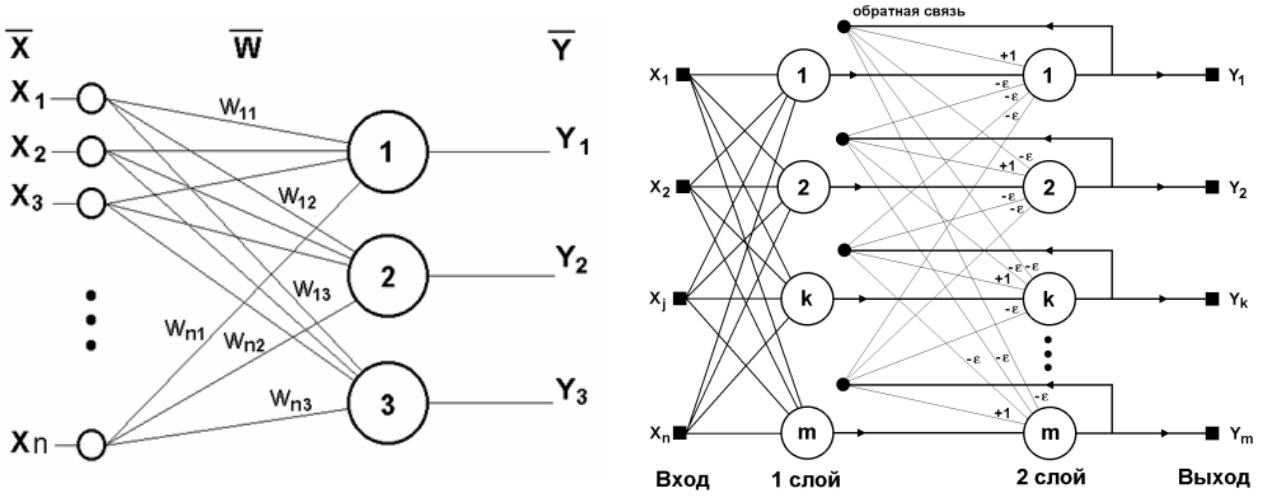

Рисунок 1. Однослойная и двухслойная нейронные сети (из [1] и [2]).

Так как принцип работы искусственного нейрона основан на механизме функционирования нейронов настоящих, рассмотрим вначале их строение.

Нервная клетка (нейрон) имеет тело, называемое сомой, со стандартным набором органелл и ядром. Можно выделить два типа отростков:

1. **дендриты,** многочисленные тонкие ветвящиеся отростки, играющие ключевую роль во взаимодействии нейронов друг с другом,
2. **аксон,** длинный и толстый отросток, ветвящийся на конце.

Входные сигналы поступают в клетку через *синапсы*, а выходной сигнал отводится аксоном через его многочисленные нервные окончания, называемые *колатералами*. Колатералы взаимодействуют с дендритами и сомами других клеток, создавая новые синапсы.

С некоторыми упрощениями можно считать, что передача нервного импульса между двумя клетками основана на выделении особых химических веществ *(нейромедиаторов),* которые, попадая на клеточную мембрану, вызывают ее поляризацию. В силу неодинаковости размеров и формы синапсов, одно и то же количество нейромедиатора возбуждает клетку в разной степени. Отсюда следует, что в математической модели нейрона каждому синапсу следует сопоставить числовой коэффициент (вес), который по физическому смыслу был бы эквивалентен его электрической проводимости.

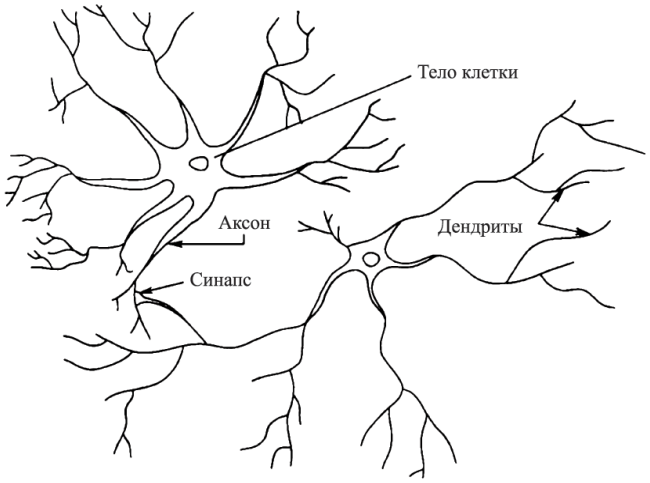

Рисунок 2. Структура и взаимодействие нервных клеток (из [3]).

Вернемся теперь к искусственному нейрону (в дальнейшем, чтобы избежать путаницы, под словом *нейрон* будет подразумеваться именно искусственный нейрон, т. е. ячейка нейронной сети).

Каждый нейрон характеризуется собственным состоянием **s**, по аналогии с нервными клетками, которые могут быть возбуждены или заторможены. Нейрон обладает группой синапсов (входных связей), соединенных с выходами других нейронов, а также имеет аксон (выходную связь). Каждый синапс характеризуется величиной *синаптической связи*, или ее *весом* $\mathbf{w_i}$.

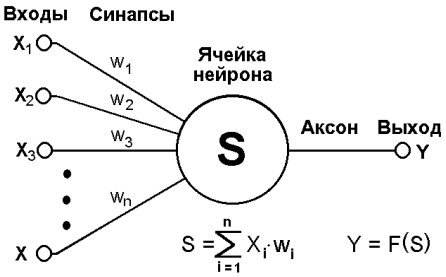

Рисунок 3. Искусственный нейрон (из [2]).

Текущее состояние i-ого нейрона будем определять как взвешенную сумму его входов:

$$s = \sum_{i=0}^{n} x_i + w_i$$

а выход — как функцию его состояния *y = f(s).*

Функция f называется активационной, и, вообще говоря, может иметь различный вид, однако чаще всего используется сигмоид (на рисунке 4, слева) или скачкообразная функция.

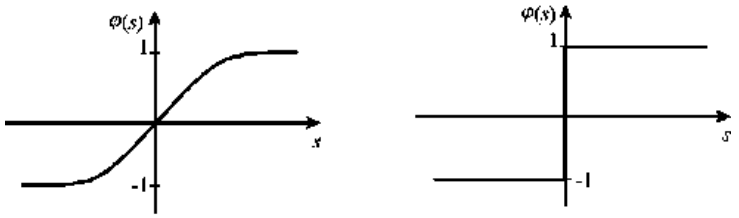

Рисунок 4. Активационные функции нейрона (из [2]).

Сигмоид обладает рядом замечательных свойств, которые делают его использование в качестве активационной функции крайне удобным, однако их рассмотрение выходит за рамки данного реферата.

Во-вторых, общим для нейронных сетей является принцип параллельной обработки сигналов, который достигается путем объединения большого числа нейронов в слои. Нейроны различных слоев определенным образом соединены, а также, в некоторых конфигурациях, соединены и нейроны одного слоя между собой, причем обработка взаимодействия всех нейронов ведется послойно.

Обучение нейронной сети. Метод обратного распространения

Очевидно, что ключевым вопросом при разработке нейронной сети является подбор весовых коэффициентов синапсов. Именно от них и будет зависеть процесс функционирования сети. Этап подбора весовых коэффициентов называется *обучением нейронной сети.*

Различают два способа обучения нейронной сети:

1. **Обучение с учителем.** Если заранее имеется значения входных данных и соответствующих им выходных сигналов, проводится так называемое обучение с учителем. Сети предоставляется эта информация, и она по некоторому правилу подстраивает веса синоптических связей таким образом, чтобы установилось необходимое соответствие входных и выходных сигналов.
2. **Обучение без учителя.** В этом случае веса определяются по алгоритму, зависящему только от входных данных.

При обучении с учителем многослойной сети разработчик может столкнуться со следующей проблемой: неизвестны оптимальные выходные значение всех слоев, кроме последнего, так что такую сеть уже невозможно обучить, руководствуясь только величинами ошибок на выходах нейронной сети. Наиболее приемлемым решением этой проблемы является использование *метода обратного распространения ошибок.* Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы. Подробнее об этом методе можно прочитать в статье (1).

Говоря об обучении без учителя, стоит выделить алгоритм обучения *по Хеббу.* Он состоит в следующем:

1. На стадии инициализации всем весовым коэффициентам присваиваются небольшие случайные значения.
2. На входы сети подается входной образ, и сигналы возбуждения распространяются по всем слоям обычным образом: для каждого нейрона рассчитывается взвешенная сумма его входов, к которой затем применяется активационная (передаточная) функция нейрона, в результате чего получается его выходное значение $y_i^{(n)}, i \in [0; M_n - 1], i \in \mathbb{Z}$, где $\mathbf{M_n}$ – число нейронов в слое **n**.
3. На основании полученных выходных значений нейронов по формулам Хебба производится изменение весовых коэффициентов. Формулы приведены в (2).
4. Цикл с шага 2, пока выходные значения сети не застабилизируются с заданной точностью. Применение этого нового способа определения завершения обучения обусловлено тем, что подстраиваемые значения синапсов фактически не ограничены.

Обучение *по Кохонену* проводится так же, с тем лишь отличием, что на шаге 3 из всего слоя выбирается нейрон, значения синапсов которого максимально походят на входной образ, и подстройка весов по формуле Кохонена (эта формула также приведена в (2) и использует значения синапсов на предыдущей итерации) проводится только для него.

Особый интерес представляют сети Хопфилда и Хэмминга, поскольку по приведенной классификации они не принадлежат, строго говоря, ни к одному из классов. О сети Хопфилда будет сказано ниже.

Нейронная сеть Хопфилда

Сеть Хопфилда состоит из одного слоя нейронов. Число нейронов равно числу входов и выходов сети. Каждый нейрон связан синапсами со всеми остальными нейронами, а также с одним из входов. Структурная схема сети приведена на рисунке 5:

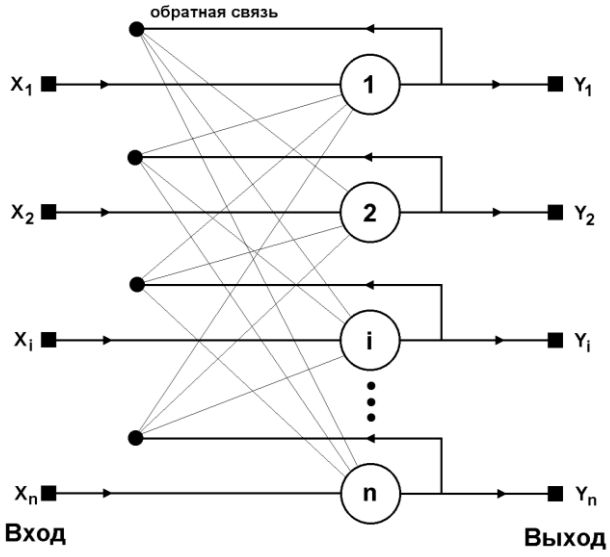

Рисунок 5. Структурная схема сети Хопфилда (из [1]).

Сеть Хопфилда позволяет решать следующую задачу: пусть имеется набор двоичных сигналов, которые считаются образцовыми. Сеть должна уметь по неидеальному («зашумленному») сигналу распознать образцовый, либо же сообщить о том, что входные данные не соответствуют ни одному из образцов.

Представим такой сигнал в виде вектора $X = \{x_i : i \in [0; n-1], i \in \mathbb{Z}\}, x_i \in \{-1; +1\}$, где **n** — число нейронов сети (отсюда видно, что размерность входного и выходного сигнала должна быть равна числу нейронов сети). Обозначим вектор, соответствующий k-ому образцу, за X^k, а его элемент — за $x_i^k, k \in [0; m-1], k \in \mathbb{Z}$. Если сеть сможет распознать зашумленный сигнал, выходной вектор **Y** будет равен соответствующему образцу $\mathbf{X^k}$, в противном случае, выходной вектор не совпадет ни с одним из образцов (скорее всего, на выходе окажется сигнал, составленный из различных кусков образцовых).

На стадии инициализации сети весовые коэффициенты синапсов устанавливаются следующим образом:

$$w_{ij} = \begin{cases} \sum_{k=0}^{m-1} x_i^k x_j^k, i \neq j \\ 0, \quad i = j \end{cases},$$

где **i** и **j** — индексы, соответственно, предсинаптического и постсинаптического нейронов; x_i^k, x_j^k — i-ый и j-ый элементы вектора k-ого образца. Нулевая проводимость синапса при **i = j** обозначает на самом деле отсутствие синапса, ведущего на тот же нейрон, с которого он берет начало.

Далее сеть функционирует по следующему алгоритму:

1. На входы сети подается неизвестный сигнал.
2. Рассчитывается новое состояние нейронов $s_j(p+1) = \sum_{i=0}^{n-1} w_{ij} y_i(p)$, *j = 0…n – 1* и новые значения аксонов $y_j(p+1) = f\left[s_j(p+1)\right]$, где f — активационная функция в виде скачка (рисунок 4, справа).
3. Проверка, изменились ли выходные значения аксонов за последнюю итерацию. Если да — переход к пункту 2, иначе (если выходы застабилизировались) — конец. При этом выходной вектор представляет собой образец, наилучшим образом сочетающийся с входными данными.

Возможности сети Хопфилда по запоминанию образов ограничены величиной, которую можно приблизить как $\frac{n}{2log_2 n}$, где n — число нейронов. Кроме того, сеть Хопфилда состоит из одного слоя, а значит, она не может решать *линейно неразрешимые* задачи. Грубо, работа всех сетей сводится к разбиению гиперпространства гиперплоскостями. Каждая полученная область является областью определения одного класса. Таким образом, однослойная сеть делит плоскость на области прямыми. Однако, например, не существует такой прямой, которая делила бы входные сигналы на классы A и B (см. рисунок 6):

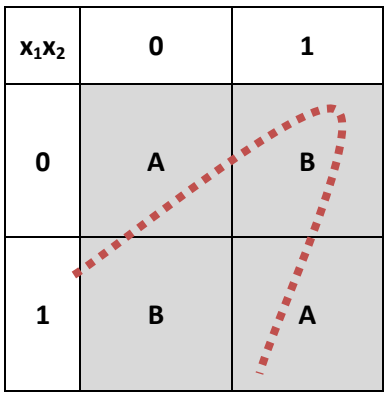

Рисунок 6. Невозможность линейного разделения классов нейронной сетью.

Видно, что это разбиение реализует функцию исключающего ИЛИ. Таким образом, функция исключающего ИЛИ нереализуема с помощью однослойной сети.

Реализация сети Хопфилда на языке C++ приведена в Приложении, вместе с примерами распознаваемых сигналов.

Список литературы

1. **Короткий С.** Нейронные сети: алгоритм обратного распространения. [В Интернете] 1996. http://www.gotai.net/documents/doc-nn-003.aspx.
2. **Короткий С.** Нейронные сети: обучение без учителя. [В Интернете] 1996. http://www.gotai.net/documents/doc-nn-004.aspx.
3. **Уоссермен Ф.** *Нейрокомпьютерная техника: Теория и практика.* 1992.
4. **Осовский С.** *Нейронные сети для обработки информации.* Москва: Финансы и статистика, 2004.
5. **Короткий С.** Нейронные сети: основные положения. [В Интернете] 1996. http://www.gotai.net/documents/doc-nn-002.aspx.
6. **Короткий С.** Нейронные сети Хопфилда и Хэмминга. [В Интернете] 1996. http://www.gotai.net/documents/doc-nn-005.aspx.

Приложение. Реализация сети Хопфилда на языке C++

Файлы neuron.h и neuron.cpp

```
#include <vector>
#include <exception>

/** Нейрон сети Хопфилда. © Матвей Андриенко, 2010 */

class HopfieldNeuron
{
public:

    explicit HopfieldNeuron(size_t syn_n);

    /** Подача входного сигнала */
    void in(const std::vector<int>& signal);

    /** Установка весового коэффициента синапса */
    void set_synapse(size_t n, int val);

    /** Получение значения аксона (выходного сигнала) */
    int out() const;

private:

    void update_status();
    void update_out();

    /** Активационная функция f(x) = sign(x) */
    int activation_f(int x);

    std::vector<int> ins_;
    std::vector<int> synapses_;
    int status_;
    int out_;
};

HopfieldNeuron::HopfieldNeuron(size_t syn_n)
:   ins_(syn_n, 0),
    synapses_(syn_n, 0),
    status_(0),
    out_(0)
{}
```

```
void HopfieldNeuron::in(const std::vector<int>& signal)
{
    if (signal.size() != synapses_.size())
        throw std::exception("signal length must be equal "
                             "to number of neuron's synapses");

    ins_ = signal;
    update_out();
}

void HopfieldNeuron::set_synapse(size_t n, int val)
{
    synapses_.at(n) = val;
}


int HopfieldNeuron::out() const
{
    return out_;
}

void HopfieldNeuron::update_status()
{
    status_ = 0;

    for (size_t i = 0; i < ins_.size(); i++)
        status_ += synapses_.at(i) * ins_.at(i);
}

void HopfieldNeuron::update_out()
{
    update_status();
    out_ = activation_f(status_);
}

int HopfieldNeuron::activation_f(int x)
{
    if      (x > 0) return  1;
    else if (x < 0) return -1;
    else            return  0;
}
```

Файлы net.h и net.cpp

```
#include <vector>
#include "debug.h"
#include "neuron.h"

/** Нейронная сеть Хопфилда. © Матвей Андриенко, 2010 */

class HopfieldNet
{
public:

    HopfieldNet(size_t neuro_n, std::vector<std::vector<int> > models);

    /** Распознает сигнал и возвращает номер образца, либо выбрасывает
          Исключение, если распознать не удается */
    size_t recognize(const std::vector<int>& signal);

    /** Приводит зашумленный сигнал к одному из образцов, либо возвращает
          Произвольный сигнал (если распознать сигнал не удалось) */
    std::vector<int> reduce_noise(const std::vector<int>& signal);

private:

    /** Инициализация сети */
    void init_network();
```

```
    size_t neuro_n_;
    std::vector<HopfieldNeuron> neurons_;
    std::vector<std::vector<int> > models_;
};



HopfieldNet::HopfieldNet(size_t neuro_n, std::vector<std::vector<int> > models)
:   neuro_n_(neuro_n),
    models_(models)
{
    init_network();
}

size_t HopfieldNet::recognize(const std::vector<int>& signal)
{
    std::vector<int> out = reduce_noise(signal);

    DEBUG(std::cout << "Trying to recognize signal...\n");

    for (size_t i = 0; i < models_.size(); i++)
        if (out == models_.at(i)) return i;

    throw std::exception("Cannot recognize signal");
}

std::vector<int> HopfieldNet::reduce_noise(const std::vector<int>& signal)
{
    DEBUG(std::cout << "Trying to reduce noise in signal...\n");

    std::vector<int> prev_out = signal;

    /* Здесь происходят основные вычисления: */
    for (long iterations = 1; ; iterations++)
    {
        /* Подача входного воздействия на сеть */
        for (size_t i = 0; i < neuro_n_; i++)
             neurons_.at(i).in(prev_out);

        /* Получения значения аксона каждого из нейронов */
        std::vector<int> out;
        for (size_t i = 0; i < neuro_n_; i++)
             out.push_back(neurons_.at(i).out());

        /* Если выходы застабилизировались - конец */
        if (out == prev_out)
        {
             DEBUG(std::cout << "It took " << iterations <<
                                " iteration(s) to stabilize the net.\n");
             return out;
        }

        prev_out = out;
    }
}


void HopfieldNet::init_network()
{
    DEBUG(std::cout << "Creating neural net with " << neuro_n_ << " neurons...\n");
    DEBUG(std::cout << "Counting values for synapses...\n");

    /* Вычисление весовых коэффициентов синапсов */
    for (size_t neuro_i = 0; neuro_i < neuro_n_; neuro_i++)
    {
        neurons_.push_back(HopfieldNeuron(neuro_n_));

        for (size_t syn_i = 0; syn_i < neuro_n_; syn_i++)
        {
             int synapse = 0;
```

```
            if (syn_i != neuro_i)
                  for (size_t mod_i = 0; mod_i < models_.size(); mod_i++)
                        synapse += models_.at(mod_i).at(syn_i) *
                                          models_.at(mod_i).at(neuro_i);

            neurons_.at(neuro_i).set_synapse(syn_i, synapse);
        }
    }
}
```

Пример использования (часть файла application.cpp)

```
/** Начать выполнение программы. © Матвей Андриенко, 2010 */

void Application::run()
{
    HopfieldNet net(signal_width_ * signal_height_, models_);

    try {
        size_t res = net.recognize(signal_);

        std::cout << "This is model " << res << ": \n\n";
        for (size_t y = 0; y < signal_height_; y++)
        {
            for (size_t x = 0; x < signal_width_; x++)
                std::cout << (models_.at(res).at(signal_width_ * y + x) >
                                                                0 ? 'X' : ' ');

            std::cout << '\n';
        }
    }
    catch (std::exception) {
        std::cout << "Signal was not recognized";
    }
}
```

Входной сигнал	Вывод сети

XX XXXXXXXXXXXXXXXX
X X   X X
X XX XXXXXXXX XX XX
X X   X   X   X
  X XX X XXXXX XXX X
X X X   X   X X
X X X XX XXXXXX X X
X X X X X   X X X
 X XXX X XXX X X X
X X X X   X X X X
  X X XXXX XX X X X
X X X   X X X
X X XXXXXXXXXXXXX X
  X   XX   X X
X XXXX   XXXXXXXX X
X   X   X   X
XXXXXXXXXX XXXXXXXXX
	Загрузка образцов из списка...
	Загрузка сигнала...
	Создание нейронной сети из 340 нейронов...
	Вычисление весовых коэффициентов синапсов...
	Попытка избавиться от шума в сигнале...
	Сеть застабилизировалась в 2 итерацию(и).
	Попытка распознать сигнал...
	Это образец 4:
	
	XXXXXXXXXXXXXXXXXXX
	X
	X XXXXXXXXXXXXXXXXX
	X X X
	X X XXXXXXXXXXXXX X
	X X X X X
	X X X XXXXXXXX X X
	X X X X X X X
	X X X X XXXX X X X
	X X X X X X X X
	X X X XXXXXX X X X
	X X X X X X
	X X XXXXXXXXXX X X
	X X X X
	X XXXXXXXXXXXXXXX X
	X X
	XXXXXXXXXXXXXXXXXXX