Министерство образования Республики Беларусь
УО «Полесский государственный университет»

В. Н. ШТЕПА, А. Г. ШТЕПА

РАСПРЕДЕЛЁННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ. НЕЧЁТКИЕ НЕЙРОННЫЕ СЕТИ. ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

Методические рекомендации по выполнению лабораторных работ для студентов по специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)»

Пинск
ПолесГУ
2020

УДК 004.7(072)
ББК 32.973я73
Ш89

Р е ц е н з е н т ы:
кандидат технических наук Д. В. Кузёмкин;
кандидат технических наук И. В. Бубырь

У т в е р ж д е н о
научно-методическим советом ПолесГУ

Штепа, В. Н.

Ш89 Распределённые информационные системы. Нечёткие нейронные сети. Генетический алгоритм : методические рекомендации по выполнению лабораторных работ / В. Н. Штепа, А. Г. Штепа. – Пинск : ПолесГУ, 2020. – 34 с.

ISBN 978-985-516-607-9

Предназначено для студентов специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)».

УДК 004.7(072)
ББК 32.973я73

ISBN 978-985-516-607-9

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В распределённых информационных системах используются три технологии интеграции данных:

- технология «клиент – сервер»;
- технология совместного использования ресурсов в рамках глобальных сетей;
- технология универсального пользовательского обмена данными.

Основной формой взаимодействия в сети является технология «клиент – сервер». Как правило, одни персональные пользователи располагают информационно-вычислительные ресурсы (процессоры, файловую систему, почтовую службу, службу печати, базу данных), а другие пользуются ими.

Компьютер, управляющий тем или иным ресурсом, принято называть сервером этого ресурса, а компьютер, который пользуется этим ресурсом, – клиентом. Если ресурсом является база данных, то говорят о сервере баз данных, назначение которого – обслуживать запросы клиентов, связанные с обработкой данных. Если ресурс – файловая система, то говорят о файловом сервере или файл-сервере и т.д.

Один из основных принципов технологии «клиент – сервер» заключается в распределении операций обработки данных на три группы, которые имеют разную природу.

Первая группа – это ввод и отображение данных.

Вторая группа объединяет прикладные операции обработки данных, характерные для решения задач данной предметной области.

К третьей группе относятся операции сохранения и управления данными (базами данных или файловыми системами).

Согласно этой классификации, в любом технологическом процессе можно выделить программы трех видов:

- программы представления, которые реализуют операции первой группы;
- приложения, поддерживающие операции второй группы;
- программы доступа к информационным ресурсам, которые реализуют операции третьей группы.

Особое внимание в современных распределённых информационных системах уделяется решению прикладных задач (вторая группа операций обработки данных) с использованием систем искусственного интеллекта на основе нейросетевых и эволюционных методов, к которым относятся нечёткие нейронные сети и математический аппарат генетического алгоритма.

1 ГИБРИДНЫЕ (НЕЧЁТКИЕ) НЕЙРОННЫЕ СЕТИ

Каждая разновидность систем искусственного интеллекта имеет свои преимущества, например, по возможностям обучения, обобщения и выработки выводов.

Нейронные сети хороши для задач распознавания образов, но они достаточно неудобные для выяснения вопроса, как они такое распознавания осуществляют. Они могут автоматически получать знания, но процесс их обучения чаще всего происходит медленно, а анализ обученной сети достаточно сложный (обученная сеть – черный ящик для пользователя). При этом любую априорную информацию (знания эксперта) для ускорения процесса его обучения в нейронную сеть ввести невозможно. Системы с нечеткой логикой, напротив, хороши для объяснения получаемых с их помощью выводов, но они не могут автоматически получать знания для использования их в механизмах выводов. Необходимость разбивки универсальных множеств на отдельные области, как правило, ограничивает количество входных переменных в таких системах небольшим значением. В целом теоретически системы с нечеткой логикой и искусственными нейронными сетями эквивалентны друг другу, однако, в соответствии с изложенным выше, на практике у них есть свои собственные преимущества и недостатки. Такое рассуждение легло в основу разработки аппарата гибридных (нечетких) сетей, в которых выводы делаются на основе нечеткой логики, но соответствующие функции принадлежности подстраиваются с использованием алгоритмов обучения нейронных сетей, например, алгоритма обратного распространения ошибки. Такие системы не только используют априорную информацию, но могут приобретать новые знания и для пользователя являются логически прозрачными.

Для объяснения сущности гибридных сетей рассмотрим простую нейронную сеть, которая имеет два входа и только один нейрон (**Рис. 1.1**).

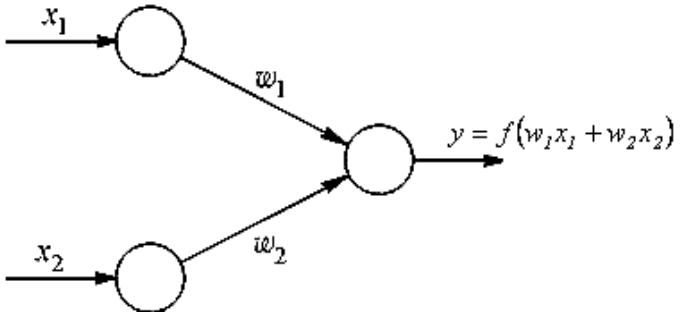

Рис. 1.1 – Элементарная НС

Здесь входные сигналы x_i «взаимодействуют» с весами w_i, создавая произведение:

$$P_i = x_i w_i, \qquad i = 1, 2. \tag{1.1}$$

Такая информация (произведения) сочетается с использованием операции суммирования, создавая вход *net* нейрона:

$$net = p_1 + p_2 = w_1x_1 + w_2x_2, \tag{1.2}$$

Выход нейрона образуется в результате преобразования входа *net* некоторой активационной функцией:

$$y = f(net) = f(w_1x_1 + w_2x_2), \tag{1.3}$$

например, сигмоидного типа:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.4}$$

Приведенную однонейронную сеть, в которой используются операции умножения, суммирования и сигмоидная функция активации, будем называть **стандартной нейронной сетью (НС)** (см. **Рис. 1.1**). В случае применения других операций, таких как *t-норма* или *t-конорма*, придем к нейронной сети, которая будет называться **гибридной**.

Определение:

Гибридная (нечеткая) нейронная сеть – это нейронная сеть с четкими сигналами, весами и активационной функцией, но с объединением x_i и w_i, p_1 и p_2 с использованием *t-нормы*, *t-конормы* или других операций.

Входы, выходы и веса гибридной нейронной сети – действительные числа, которые принадлежат отрезку [0, 1]. Рассмотрим следующие примеры элементарных гибридных нейронных сетей.

Нечеткий нейрон «И». Сигналы x_i и веса w_i в этом случае сочетаются с помощью треугольной конормы:

$$p_i = S(w_i, x_i), \quad i = 1, 2, \tag{1.5}$$

а выход образуется с применением треугольной нормы **(Рис. 1.2)**:

$$y = \mathrm{AND}(p_1, p_2) = \mathrm{T}(p_1, p_2) = \mathrm{T}(\mathrm{S}(w_1, x_1), \quad \mathrm{S}(w_2, x_2)). \tag{1.6}$$

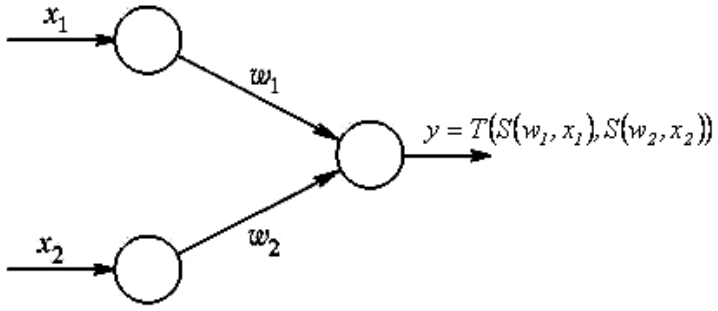

Рис. 1.2 – Структура гибридного нейрона «И»

Если принять $\mathrm{T} = \min, \ \mathrm{S} = \max,$ (1.7)

тогда нечеткий нейрон «И» реализует композицию min-max:

$$y = \min(w_1 \vee x_1, w_2 \vee x_2). \tag{1.8}$$

Нечеткий нейрон «АБО» (**Рис. 1.3**). Сигналы x_i и весы w_i сочетаются с помощью треугольной нормы:

$$p_i = \grave{O}(w_i, x_i), \quad i = 1, 2, \tag{1.9}$$

а выход образуется с применением треугольной конормы (см. **Рис. 1.3**):

$$y = \mathrm{OR}(p_1, p_2) = \mathrm{S}(p_1, p_2) = \mathrm{S}(\mathrm{T}(w_1, x_1), \quad \mathrm{T}(w_2, x_2)). \tag{1.10}$$

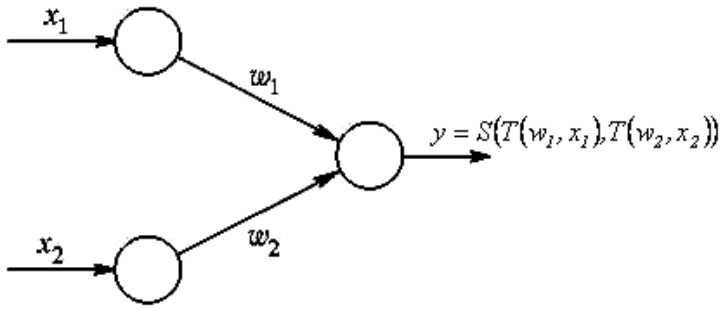

Рис. 1.3 – Нечеткий нейрон «АБО»

Если принять $T = min,\ \ S = max,$ (1.11)

тогда нечеткий нейрон «АБО» реализует композицию max-min:

$$y = \max(w_1 \wedge x_1, w_2 \wedge x_2). \tag{1.12}$$

Опишем типичный подход к построению алгоритмов обучения и использования гибридных нейронных сетей. Предположим, что гибридной сетью должно быть реализовано неизвестное отображение:

$$y^k = f(\mathrm{X}^k) = f(x_1^k, x_2^k, ..., x_n^k),\ k = 1, 2, \ldots, N, \tag{1.13}$$

при наличии обучающего множества:

$$\{(\mathrm{X}^1, y^1), ..., (\mathrm{X}^N, y^N)\}. \tag{1.14}$$

Для моделирования неизвестного отображения f используем упрощенный алгоритм нечеткого вывода, используя следующую форму записи предикатных правил:

$$\Pi_i:\ \text{если } x_1 \in A_{i1}\ i\ x_2 \in A_{i2}\ i\ \ldots\ i\ x_n \in A_{in}, \qquad \text{тогда } y = z_i,\ i = 1, 2, \ldots, m, \tag{1.15}$$

где A_{ij} – нечеткие числа треугольной формы;

z_i – действительные числа.

Определение степени истинности i-го правила осуществляется с помощью операции умножения (Larsen):

$$\alpha_i = \prod_{j=1}^{n} A_{ij}(x_j^k), \tag{1.16}$$

(здесь можно использовать и другие представления для моделирования логического оператора «И») и определяя выход нечеткой системы дискретным аналогом центроидного метода:

$$o^k = \frac{\sum_{i=1}^{m} \alpha_i z_i}{\sum_{i=1}^{m} \alpha_i}. \tag{1.17}$$

Введение функции ошибки для k-го предъявленного образца вида:

$$E_k = \frac{1}{2}(o^k - y^k)^2 \tag{1.18}$$

позволяет, как в обычных (стандартных) нейронных сетях, использовать градиентный метод для подстройки параметров заданных предикатных правил. Так, величины z_i можно корректировать по соотношению:

$$z_i := z_i - \eta \frac{\partial E_k}{\partial z_i} = z_i - \eta(o^k - y^k)\frac{\alpha_i}{\alpha_1 + \alpha_2 + \ldots + \alpha_m}, \quad i = 1, 2, \ldots, m, \tag{1.19}$$

где η – константа, характеризующая скорость обучения.

Более подробно алгоритм настройки рассмотрим на примере системы, включающей два правила:

Π_1: если $x \in A_1$, тогда $y = z_1$,

Π_2: если $x \in A_2$, тогда $y = z_2$,

при этом предполагается, что нечеткие понятия A_1 («малый») и A_2 («большой») имеют сигмоидные функции принадлежности:

$$A_1(x) = \frac{1}{1 + e^{b_1(x - a_1)}}, \qquad A_2(x) = \frac{1}{1 + e^{b_2(x - a_2)}}, \tag{1.20}$$

которые характеризуются параметрами a_1, a_2, b_1, b_2.

Степени истинности правил определяются в этом случае соотношениями:

$$\alpha_1 = A_1(x) = \frac{1}{1+e^{b_1(x-a_1)}}, \qquad \alpha_2 = A_2(x) = \frac{1}{1+e^{b_2(x-a_2)}}, \tag{1.21}$$

а выход системы – выражением:

$$o = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2} = \frac{A_1(x) z_1 + A_2(x) z_2}{A_1(x) + A_2(x)}. \tag{1.22}$$

Предположим, что имеется обучающее множество $\{(x_1, y_1), ..., (x^N, y^N)\}$, отражающее неизвестную функцию *f*. Необходимо провести такую настройку параметров системы a_1, a_2, b_1, b_2, z_1, z_2, при которых обеспечивается лучшая аппроксимация такой функции.

Решение. В этом случае функция ошибки может быть записана в форме:

$$E_k = E_k(a_1, b_1, a_2, b_2, z_1, z_2) = \frac{1}{2}\left(o^k(a_1, b_1, a_2, b_2, z_1, z_2) - y^k\right)^2. \tag{1.23}$$

Используя далее тот же подход, что и при алгоритме обратного распространения ошибки, запишем:

$$z_1 := z_1 - \eta \frac{\partial E_k}{\partial z_1} = z_1 - \eta\left(o^k - y^k\right)\frac{\alpha_1}{\alpha_1 + \alpha_2} = z_1 - \eta\left(o^k - y^k\right)\frac{A_1(x^k)}{A_1(x^k) + A_2(x^k)}, \tag{1.24}$$

$$z_2 := z_2 - \eta \frac{\partial E_k}{\partial z_2} = z_2 - \eta\left(o^k - y^k\right)\frac{\alpha_2}{\alpha_1 + \alpha_2} = z_2 - \eta\left(o^k - y^k\right)\frac{A_2(x^k)}{A_1(x^k) + A_2(x^k)}. \tag{1.25}$$

Аналогичным путем могут быть получены развернутые выражения для коррекции коэффициентов a_1, a_2, b_1, b_2. Выходные соотношение такие:

$$a_1 := a_1 - \eta \frac{\partial E_k}{\partial a_1}, \qquad a_2 := a_2 - \eta \frac{\partial E_k}{\partial a_2}, \tag{1.26}$$

$$b_1 := b_1 - \eta \frac{\partial E_k}{\partial b_1}, \qquad b_2 := b_2 - \eta \frac{\partial E_k}{\partial b_2}. \tag{1.27}$$

Конечные выражения есть достаточно громоздкими, но могут быть упрощены в случае, если функции принадлежности имеют вид:

$$A_1(x) = \frac{1}{1+e^{b(x-a)}}, \qquad A_2(x) = \frac{1}{1+e^{b(x-a)}}, \tag{1.28}$$

Такие функции характеризуются всего двумя параметрами (*a* и *b*) и в определенном смысле являются симметричными и удовлетворяют уравнению:

$$A_1(x) + A_2(x) = 1. \tag{1.29}$$

Заметим, что из последнего и ранее полученных уравнений следует:

$$z_1 := z_1 - \eta \frac{\partial E_k}{\partial z_1} = z_1 - \eta\left(o^k - y^k\right)\frac{\alpha_1}{\alpha_1 + \alpha_2} = z_1 - \eta\left(o^k - y^k\right)A_1(x^k), \tag{1.30}$$

$$z_2 := z_2 - \eta \frac{\partial E_k}{\partial z_2} = z_2 - \eta\left(o^k - y^k\right)\frac{\alpha_2}{\alpha_1 + \alpha_2} = z_2 - \eta\left(o^k - y^k\right)A_2(x^k). \tag{1.31}$$

Тогда $$a := a - \eta \frac{\partial E_k(a, b)}{\partial a}, \tag{1.32}$$

где
$$\begin{cases} \dfrac{\partial E_k(a,b)}{\partial a} = (o^k - y^k)\dfrac{\partial o^k}{\partial a} = (o^k - y^k)\dfrac{\partial}{\partial a}(z_1 A_1(x^k) + z_2 A_2(x^k)) = \\ = (o^k - y^k)\dfrac{\partial}{\partial a}(z_1 A_1(x^k) + z_2(1 - A_1(x^k))) = (o^k - y^k)(z_1 - z_2)\dfrac{\partial A_1(x^k)}{\partial a} = \\ = (o^k - y^k)(z_1 - z_2)b\dfrac{e^{b(x^k - a)}}{(1 + e^{b(x^k - a)})^2} = (o^k - y^k)(z_1 - z_2)bA_1(x^k)(1 - A_1(x^k)) = \\ = (o^k - y^k)(z_1 - z_2)bA_1(x^k)A_1(x^k), \end{cases} \tag{1.33}$$

и
$$b := b - \eta \frac{\partial E_k(a,b)}{\partial b}, \tag{1.34}$$

где
$$\frac{\partial E_k(a,b)}{\partial b} = (o^k - y^k)(z_1 - z_2)\frac{\partial}{\partial b}\frac{1}{1 + e^{b(x^k - a)}} = (o^k - y^k)(z_1 - z_2)(x^k - \dot{a})A_1(x^k)(1 - A_1(x^k)) = \tag{1.35}$$
$$= (o^k - y^k)(z_1 - z_2)(x^k - \dot{a})A_1(x^k)A_1(x^k)$$

Формулы (1.20–1.35) иллюстрируют идеи алгоритмов обучения и использования гибридной сети.

Другим примером может служить система, имеющая следующую базу знаний:

П1: если $x_1 \in L_1$ и $x_2 \in L_2$ и $x_3 \in L_3$, тогда $y \in H$;
П2: если $x_1 \in H_1$ и $x_2 \in H_2$ и $x_3 \in L_3$, тогда $y \in M$;
П3: *если* $x_1 \in H_1$ и $x_2 \in H_2$ и $x_3 \in H_3$, тогда $y \in S$,

где x_1, x_2, x_3 – входные переменные;

y – выход системы;

$L_1, L_2, L_3, H_1, H_2, H_3, H, M, S$ – некоторые нечеткие множества с функциями принадлежности сигмоидного типа:

$$L_j(t) = \frac{1}{1 + e^{b_j(t - c_j)}}, \qquad H_j(t) = \frac{1}{1 + e^{-b_j(t - c_j)}}, \qquad j = 1, 2, 3, \tag{1.36}$$

$$H(t) = \frac{1}{1 + e^{-b_4(t - c_4 + c_5)}}, \qquad M(t) = \frac{1}{1 + e^{-b_4(t - c_4)}}, \qquad S(t) = \frac{1}{1 + e^{b_4(t - c_4)}}. \tag{1.37}$$

Для определения выходной переменной используется алгоритм заключения Tsukamoto (см. выше):

1) подсчитываются значения истинности предпосылок для каждого правила:

$$\begin{aligned} \alpha_1 &= L_1(a_1) \wedge L_2(a_2) \wedge L_3(a_3), \\ \alpha_2 &= H_1(a_1) \wedge H_2(a_2) \wedge L_3(a_3), \\ \alpha_3 &= H_1(a_1) \wedge H_2(a_2) \wedge H_3(a_3), \end{aligned} \tag{1.38}$$

где a_1, a_2, a_3 – текущие значения входов системы;

2) для каждого правила определяются частные выходы:

$$z_1 = B^{-1}(\alpha_1) = c_4 + c_5 + \frac{1}{b_4}\ln\frac{1-\alpha_1}{\alpha_1},$$
$$z_2 = B^{-1}(\alpha_2) = c_4 + \frac{1}{b_4}\ln\frac{1-\alpha_2}{\alpha_2}, \qquad (1.39)$$
$$z_3 = B^{-1}(\alpha_3) = c_4 + \frac{1}{b_4}\ln\frac{1-\alpha_3}{\alpha_3};$$

3) ищется общий выход системы:

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3}{\alpha_1 + \alpha_2 + \alpha_3}. \qquad (1.40)$$

Изложенный процесс демонстрируется на **Рис. 1.4**.

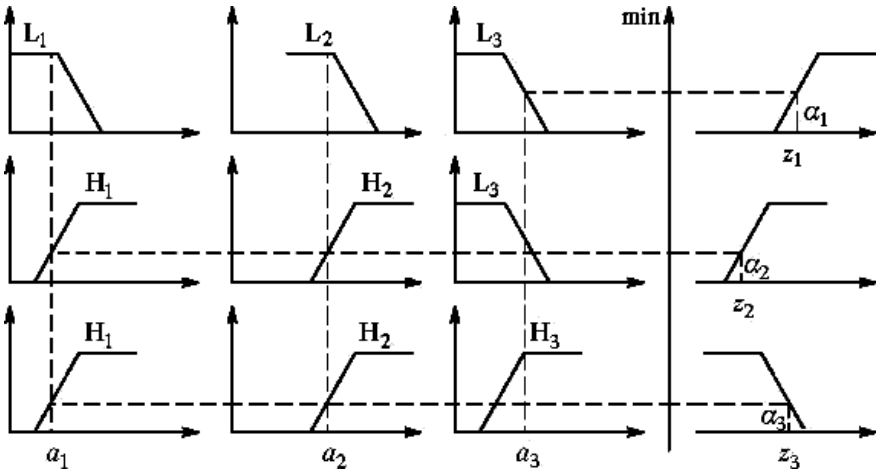

Рис. 1.4 – Демонстрация алгоритма вывода Tsukamoto

Заметим, что сети с подобной архитектурой в англоязычной литературе получили название ANFIS (Adaptive Neuro-Fuzzy Inference System). Такая сеть может быть описана следующим образом (**Рис. 1.5**):

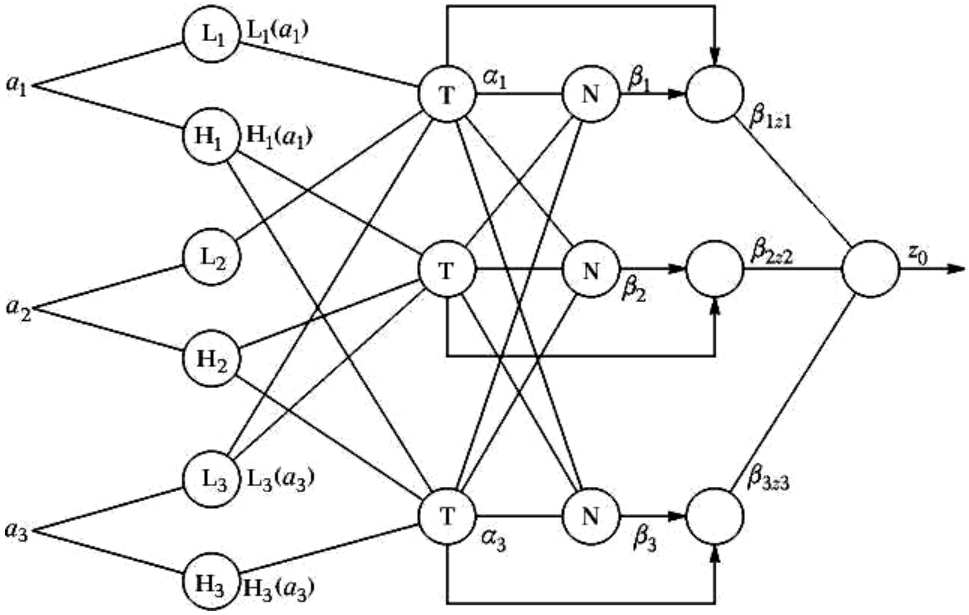

Рис. 1.5 – Структура гибридной нейронной сети (архитектура ANFIS)

1. *Слой 1* (Layer 1). Выходы узлов этого слоя представляют собой значения функций принадлежности при конкретных (заданных) значениях входов.

2. *Слой 2* (Layer 2). Выходами нейронов этого слоя являются степени истинности предпосылок каждого правила базы знаний системы, вычисляются по формулам:

$$\begin{aligned} \alpha_1 &= L_1(a_1) \wedge L_2(a_2) \wedge L_3(a_3), \\ \alpha_2 &= H_1(a_1) \wedge H_2(a_2) \wedge L_3(a_3), \\ \alpha_3 &= H_1(a_1) \wedge H_2(a_2) \wedge H_3(a_3) \end{aligned} \tag{1.41}$$

Все нейроны этого слоя обозначены буквой T, это означает, что они могут реализовывать произвольную t-норму для моделирования операции «И».

3. *Слой 3* (Layer 3). Нейроны этого слоя (обозначены буквой N) вычисляют величины:

$$\beta_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3}, \qquad \beta_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2 + \alpha_3}, \qquad \beta_3 = \frac{\alpha_3}{\alpha_1 + \alpha_2 + \alpha_3}. \tag{1.42}$$

4. *Слой* 4 *(Layer* 4). Нейроны слоя выполняют операции:

$$\beta_1 z_1 = \beta_1 H^{-1}(a_1), \qquad \beta_2 z_2 = \beta_2 M^{-1}(a_2), \qquad \beta_3 z_3 = \beta_3 S^{-1}(a_3), \tag{1.43}$$

5. *Слой* 5 *(Layer* 5). Единственный нейрон вычисляет выход сети:

$$z_0 = \beta_1 z_1 + \beta_2 z_2 + \beta_3 z_3. \tag{1.44}$$

Корректировка параметров системы осуществляется в соответствии с ранее рассмотренного подхода.

Так, например, настройки коэффициентов b_4, c_4 и c_5 осуществляются по формулам:

$$\begin{aligned} b_4 &:= b_4 - \eta \frac{\partial E_k}{\partial b_4} = b_4 - \frac{\eta}{b_4^2} \delta_k \frac{\alpha_1 + \alpha_2 - \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3}, \\ c_4 &:= c_4 - \eta \frac{\partial E_k}{\partial c_4} = c_4 - \eta \delta_k \frac{\alpha_1 + \alpha_2 + \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3} = c_4 + \eta \delta_k, \\ c_5 &:= c_5 - \eta \frac{\partial E_k}{\partial c_5} = c_5 - \eta \delta_k \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3}. \end{aligned} \tag{1.45}$$

2 СИСТЕМЫ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

В 1966 г. Л. Дж. Фогель, А. Дж. Оуэнс, М. Дж. Уолш предложили и исследовали эволюцию простых автоматов, которые прогнозируют символы в цифровых последовательностях. В 1975 г. Д. Х. Холланд предложил схему генетического алгоритма. Эти работы легли в основу главных направлений разработки эволюционных алгоритмов. Простой же генетический алгоритм был впервые описан Гольдбергом на основе работ Холланда.

Постановка задачи. Пусть дана некоторая сложная целевая функция, которая зависит от нескольких переменных, и нужно решить задачу оптимизации, т.е. найти такие значения переменных, при которых значение этой функции максимальное или минимальное.

Эту задачу можно решить, применяя известные биологические эволюционные подходы к оптимизации. Будем рассматривать каждый вариант (набор значений переменных) как особь, а значение целевой функции для этого варианта – как приспособленность данной особи. Тогда в процессе эволюции приспособленность особей будет расти, и будут появляться оптимальные варианты. Остановив эволюцию в некоторый момент и выбрав лучший вариант, можно получить достаточно хорошее решение задачи.

Генетический алгоритм (ГА) – это последовательность управляющих действий и операций, которая моделирует эволюционные процессы на основе аналогов механизмов генетического наследования и естественного отбора (**Рис. 2.1**). При этом сохраняется биологическая терминология в упрощенном виде.

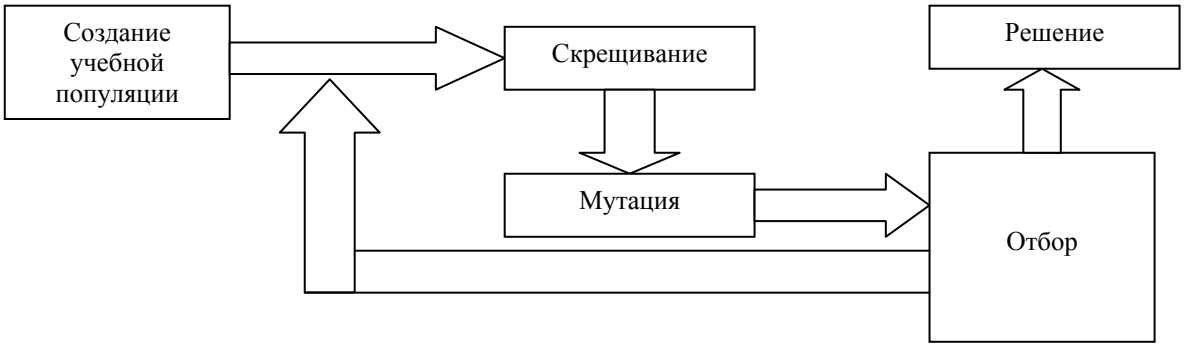

Рис. 2.1 – Вариант генетического алгоритма

Хромосома – вектор (последовательность) из нулей и единиц, каждая позиция (бит) которого называется *геном*.

Особь (индивидуум) = генетический код – набор хромосом = вариант решения задачи.

Кроссовер – операция, при которой две хромосомы обмениваются своими частями.

Мутация – случайное изменение одной или нескольких позиций в хромосоме.

Генетические алгоритмы представляют собой скорее подход, чем единственные алгоритмы. Они требуют содержательного наполнения для решения каждой конкретной задачи. На **Рис. 2.1** показан один из вариантов структуры генетического алгоритма.

Сначала генерируется случайная популяция из нескольких особей со случайным набором хромосом (числовых векторов). Генетический алгоритм имитирует эволюцию этой популяции как циклический процесс скрещивания особей, мутации и смены поколений (отбора).

В течение жизненного цикла популяции, т.е. в результате нескольких случайных скрещиваний (посредством кроссовера) и мутаций к ней добавляется определенное количество новых вариантов. Далее происходит отбор, в результате которого из старой популяции формируется новая, после чего старая популяция погибает. После отбора к новой по-

пуляции опять применяются операции кроссовера и мутации, затем опять происходит отбор, и так далее. Отбор в генетическом алгоритме тесно связан с принципами естественного отбора следующим образом:

- приспособленность особи соответствует значению целевой функции на заданном варианте;
- выживание наиболее приспособленных особей соответствует тому, что популяция следующего поколения вариантов формируется с учетом целевой функции. Чем более приспособленная особь, тем больше вероятность ее участия в кроссовере, т.е. в размножении.

Таким образом, модель отбора определяет, как следует строить популяцию следующего поколения. Как правило, вероятность участия особи в скрещивании берется пропорционально ее приспособленности. Часто используется так называемая стратегия элитизма, при которой несколько лучших особей переходят в следующее поколение без изменений, не участвуя в кроссовере и отборе. В любом случае каждое следующее поколение будет в среднем лучше предыдущего. Когда приспособленность особей перестает заметно увеличиваться, процесс останавливают и как решение задачи оптимизации берут лучший из найденных вариантов.

Математически изложенное можно представить следующим образом. Пусть имеется некоторая целевая функция многих переменных. Необходимо найти глобальный максимум или минимум: $f(x_1, \ldots, x_n)$. Представим независимые переменные в виде хромосом. Для этого выполним кодирования независимых переменных либо в двоичном формате, либо в формате с плавающей запятой. В случае двоичного кодирования используется n бит для каждого параметра, причем n может быть разным. Если параметр может изменяться между минимальным *(min)* и максимальным *(max)* значениями, то можно использовать следующие формулы для преобразования:

$$r = \frac{g(\max - \min)}{(2^n - 1)} + \min, \qquad g = \frac{(r - \min)}{(\max - \min)(2^n - 1)}, \tag{2.1}$$

где g – значение параметра в двоичном формате;
r – значение параметра в формате с плавающей запятой.

Хромосомы в формате с плавающей запятой задаются путем последовательного размещения закодированных параметров друг за другом. Наиболее хорошие результаты дает вариант представления хромосом в двоичном формате. Однако в этом случае необходимо постоянно осуществлять кодирование/декодирование параметров (генов). Рассмотрим работу генетического алгоритма более подробно. Заранее подбирается некоторое представление для рассматриваемого решения, размер и структура популяции. В первом поколении случайным образом генерируется популяция хромосом. Конечно, размер популяции постоянен. Определяется «полезность» хромосом, после чего генетический алгоритм может начинать генерировать новую популяцию. Далее осуществляется репродуцирование, состоящее из: селекции; трех генетических операторов – кроссовера, мутации, инверсии, порядок применения которых неважен. Оператор селекции (*reproduction*, *selection*) осуществляет отбор хромосом в соответствии со значениями их функции приспособленности. Существуют как минимум два популярных типа оператора селекции – рулетка и турнир. *Roulette-wheel selection* отбирает особи с помощью n «запусков» рулетки. Колесо рулетки содержит по одному сектору для каждого члена популяции.

Размер i-го сектора пропорционален соответствующей величине $P_{sel}(i)$ и вычисляется по формуле:

$$P_{sel}(i) = \frac{f(i)}{\sum_{i=1}^{n} f(i)} \tag{2.2}$$

При таком отборе члены популяции с более высокой приспособленностью будут выбираться с большей вероятностью, чем особи с низкой приспособленностью (**Рис. 2.2**).

Метод турнирного отбора (tournament selection) реализует n турниров, чтобы выбрать n особей. Каждый турнир построен на выборке k элементов из популяции и выбора лучшей особи среди них. Наиболее распространенный турнирный отбор: $k = 2$. Из трех генетических операторов кроссовер является наиболее важным. Он генерирует новую хромосому потомка, сочетая генетический материал двух родителей.

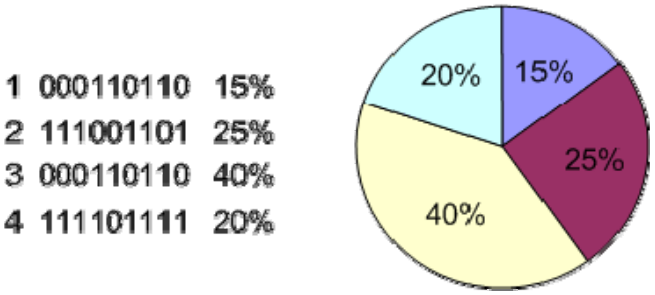

Рис. 2.2 – Оператор селекции типа колеса рулетки с пропорциональными функциями приспособленности

Существует несколько вариантов кроссовера. Наиболее простым является одноточечный, в котором берутся две хромосомы и «перерезаются» в случайно выбранной точке. Хромосома потомка выходит с начала одной и из конца другой родительской хромосомы:

$$\begin{array}{l} 001100101110010 \mid 11000 \\ 110101101101000 \mid \boldsymbol{11100} \end{array} \quad \rightarrow \quad 0011001011100101\boldsymbol{11100}.$$

Мутация представляет собой случайное изменение хромосомы (обычно простым изменением состояния одного из битов на противоположный).

Оператор позволяет, во-первых, более быстро находить локальные экстремумы и, во-вторых, перескочить в другой локальный экстремум:

$$0011001011100\boldsymbol{1}011000 \quad \rightarrow \quad 0011001011100\boldsymbol{1}11000.$$

Инверсия меняет порядок бит в хромосоме путем циклической перестановки (случайное количество раз). Много модификаций ГА обходятся без данного генетического оператора:

$$0011001011100101\boldsymbol{11000} \quad \rightarrow \quad \boldsymbol{11000}001100101110010.$$

По скорости определения оптимума целевой функции генетические алгоритмы на несколько порядков превосходят случайный поиск. Причина заключается в том, что большинство систем имеют достаточно независимые подсистемы. В результате при обмене генетическим материалом от каждого из родителей берутся гены, отвечающие наиболее удачном варианту определенной подсистемы (неудачные варианты постепенно погибают).

Генетический алгоритм позволяет накапливать удачные решения для таких систем. Данные, закодированные в генотипе, могут представлять собой команды какой-либо виртуальной машины. В таком случае говорят об эволюционном или генетическом программировании. В простейшем случае можно ничего не менять в генетическом алгоритме.

Современные алгоритмы генетического программирования функционируют в системах с переменной длиной генотипа.

Пример:

Задание. Необходимо оптимизировать некоторую функцию $F(X_1, X_2, .., X_n)$. Мы ищем ее глобальный максимум. Тогда для реализации ГА нам нужно придумать, как мы будем хранить решение. По сути, нам нужно поместить все $X_1 – X_n$ в некоторый вектор, который будет играть роль хромосомы.

Решение. Один из самых распространенных способов – кодировать значения переменных в битном векторе. Например, выделим каждому X по 8 бит. Тогда наш вектор будет длиной $L = 8 \cdot n$. Для простоты будем считать, что биты лежат в массиве $X[0..L-1]$.

Пусть каждая особь состоит из массива *X* и значения функции *F*. Тогда ГА будет состоять из следующих шагов:

1. Генерация начальной популяции – заполнение популяции особями, в которых элементы массива *X* (биты) заполнены случайным образом. Для выбора родительской пары используем элитный отбор, т.е. берем *K* особей, с максимальными значениями функции *F*, и составляем из них все возможные пары *(K*(K – 1)/2)*.

2. Кроссинговер – берем случайную точку t на массиве *X (0..L–1)*.

3. **Все элементы массива** с индексами *0–t* новой особи (потомка) заполняем элементами с теми же индексами, но из массива X первой родительской особи. Другие элементы заполняются из массива второй родительской особи. Для второго потомка делается наоборот – элементы *0–t* берут от второго отпрыска, а другие – от первого.

4. Новые особи с некоторой вероятностью мутируют – инвертируется случайный бит массива X этой особи. Вероятность мутации – около 1 %.

5. Полученные особи-потомки добавляются в популяцию после переоценки. Обычно новую особь добавляют вместо плохой старой особи – при условии, что значение функции на новой особи выше значения функции на старой («плохой») особи.

6. Если самое лучшее решение в популяции нас не удовлетворяет, то переходим на шаг 2.

Если строго придерживаться правил, то ГА должен содержать еще такие шаги, как отбор особей для размножения и генерация пар из отобранных особей. При этом каждая особь может быть задействована в одной и более паре в зависимости от используемого алгоритма.

Понятие *шаблона* было введено Холландом и используется для анализа работы ГА. В частности, рассматриваются процессы конструирования и разрушения определенного шаблона в течение развития популяции (schema dynamics).

Например, для двух бинарных строк «111000111000» и «110011001100» шаблон будет выглядеть следующим образом: «11*0****1*00». Символом «*» в некотором разряде обозначается то, что там может быть как 1, так и 0. Т.е. с помощью шаблонов можно как бы выделять общие участки двоичных строк и маскировать расхождение. Имея в составе шаблона *m* символов «*», можно закодировать (обобщить) *2m* двоичных строк. Так, например, шаблон «01*0*1» описывает набор строк {«010001», «010011», «011001», «011011»}.

Определяющей длиной шаблона (schema defining length) называется расстояние между двумя крайними символами «0» и/или «1». Для шаблона «01*0*1» определяющая длина равна 5, а для шаблона «**0**1*» определяющая длина равна 3.

Порядок шаблона (schema order) – это еще одна характеристика шаблона, равна она числу фиксированных позиций в строке, т.е. общему числу «0» и «1». Для шаблонов «01*0*1» и «**0**1*» порядки равны 4 и 2 соответственно.

Хромосома является, по сути, двоичной строкой. В то же время особь, которой принадлежит хромосома, содержащая набор генов-параметров задачи, поставлена в соответствие величине, характеризующей ее приспособленность.

Т.е. шаблон является обобщением нескольких бинарных строк (хромосом) – можно говорить, что особи, обладающие хромосомами, соответствующие одному шаблону, более приспособлены, а особи с хромосомами, которые соответствуют другому шаблону, менее приспособлены.

Можно сказать, что сущность работы ГА состоит в поиске двоичной строки определенного вида со всего множества бинарных строк. Пространство поиска составляет *2L* строк, а его размерность равна *L* (*L*-мерное пространство), где *L* – длина хромосомы. Шаблон соответствует некоторой гиперплоскости в этом пространстве. Данное утверждение можно продемонстрировать следующим образом. Пусть разрядность хромосомы равна 3, тогда всего можно закодировать 23 = 8 строк. Представим куб в трехмерном про-

странстве (**Рис. 2.3**). Обозначим вершины этого куба 3-разрядными бинарными строками так, чтобы метки соседних вершин отличались ровно на 1 разряд, причем вершина с меткой «000» находилась бы в начале координат.

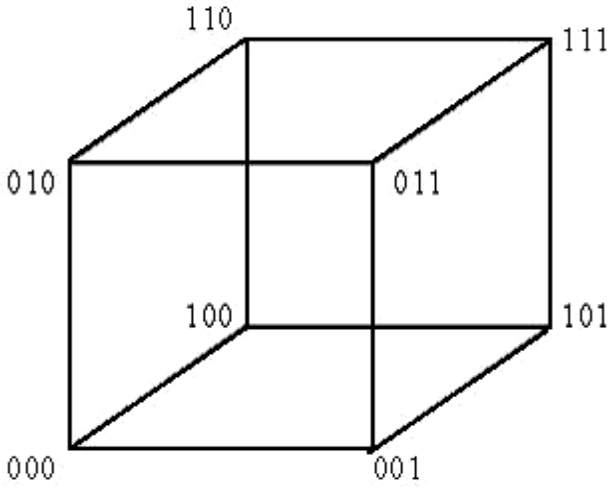

Рис. 2.3 – 3-мерный куб

Если взять шаблон вида «**0», он опишет левую грань куба, а шаблон «*10» – верхнее ребро этой грани. Очевидно, что шаблон «***» соответствует всему пространству.

Если взять двоичные строки длиной 4 разряда, то разбивку пространства шаблонами можно изобразить на примере 4-мерного куба с соответствующими вершинами (**Рис. 2.4**):

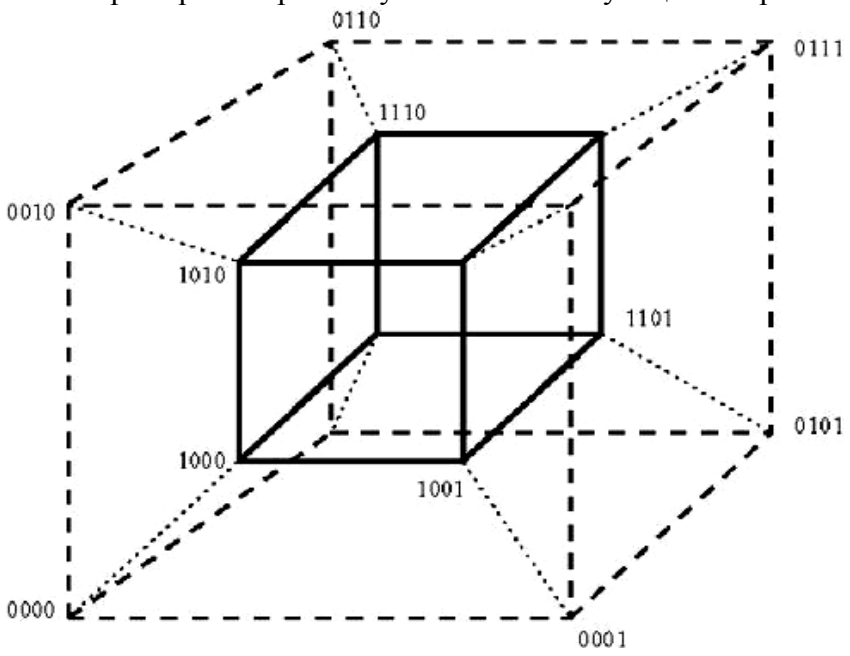

Рис. 2.4 – 4-мерный куб

Здесь шаблону «*1**» соответствует гиперплоскость, которая включает задние грани внешнего и внутреннего куба, а шаблону «**10» – гиперплоскость с верхними ребрами левых граней обоих кубов.

Разбивку пространства поиска можно представить по-другому. Представим координатную плоскость, в которой по одной оси мы будем откладывать значения двоичных строк, а по другой – значение целевой функции (**Рис. 2.5**).

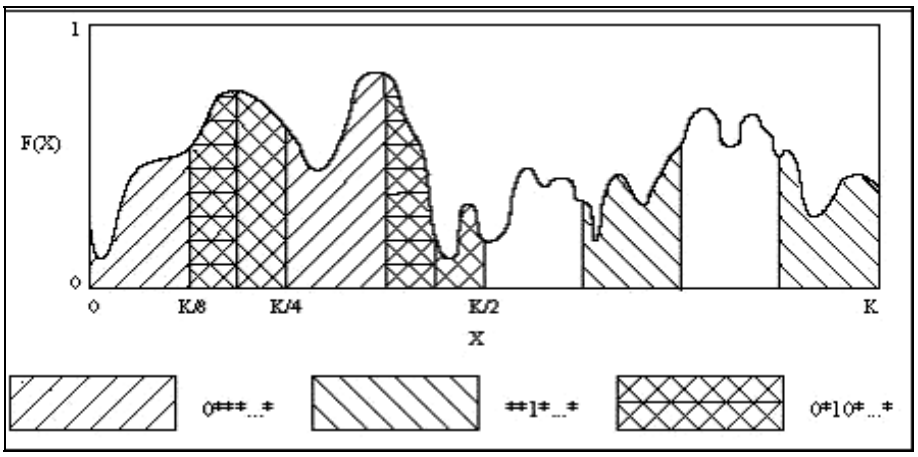

Рис. 2.5 – Разбивка пространства

Участки пространства заштрихованы различным стилем, соответствуют различным шаблонам. Число *K* в правой части горизонтальной оси соответствует максимальному значению бинарной строки «111...111». Из рисунка видно, что шаблон «0***...*» покрывает всю левую половину отрезка, шаблон «**1*...*» – четыре участка шириной в одну восьмую часть, а шаблон «0*10*...*» – левые половины участков, находящихся на пересечении первых двух шаблонов. Таким образом, происходит разбивка пространства.

Теорема шаблонов Холланда

Теорема шаблонов является одной из основных теорем теории ГА. Она была получена для канонического ГА. Для других вариантов ГА, которые реализуют отличные от канонического стратегии отбора и формирования нового поколения, теорема практически недейственна. С развитием теории ГА теорема шаблонов незначительно менялась и оправдывалась. Ниже приводится первоначальный вариант теоремы шаблонов 1975 г. и исправленный вариант 1992 г. Обе эти теоремы принадлежат Д. Холланду.

Для начала представим, что есть популяция двоичных строк длины *L*. Вероятность проведения кроссовера равна *Pc*. Тип кроссовера: одноточечный. Пусть определяющая длина шаблона *H* равна *d (H)*, а его приспособленность *f (H)*. Часть строк, соответствующих шаблону H в текущем поколении T, равна *m (H, t)*. Нас интересует, какая часть строк, соответствующих шаблону H, будет присутствовать в популяции в следующем поколении – *m (H, t + 1)*. Прежде чем идти дальше, остановимся на одном моменте, а именно: с какой вероятностью кроссовер разрушит уже имеющийся шаблон?

Очевидно, что если точка разрыва не попадает внутрь уже имеющегося шаблона, то шаблон будет разрушен. Т.е. если *PC * d (H) / (L-1)* – вероятность того, что точка разрыва попадет внутрь шаблона, то *1-pc * d (H) / (L-1)* – вероятность того, что кроссовер не разрушит шаблон. Для канонического ГА шансы особи принять участие в скрещивании исчисляются в соответствии с отношением *f / fcp*, где *f* – значение приспособленности данной особи, а *fcp* – средняя приспособленность. Таким образом, вероятность того, что строка соответствующего шаблона H будет участвовать в скрещивании равна *m (H, t) * f (H, t) / fcp (t)*.

Учитывая вероятность разрушения шаблона кроссовером, можно сформулировать первоначальную теорему шаблонов (Холланд, 1975 г.):

$$\langle m(H, t+1)\rangle \geq m(H, t)\frac{f(H, t)}{\bar{f}(t)}\left[1 - p_c \frac{\delta(H)}{l-1}\right]. \qquad (2.3)$$

Одним из недостатков теоремы (2.3) является то, что в ней отсутствует влияние мутации на создание и разрушение шаблонов. Если считать, что вероятность мутации одного разряда равна Pm и порядок шаблона H равна *o (H)*, то вероятность того, что мутация не разрушит шаблон, равна *(1-Pm) o (H)*.

Т.е. если мутирующий разряд не попадает ни на одну фиксированную позицию внутри шаблона, то она не меняется. С учетом этого исправленная теорема шаблонов выглядит следующим образом (Холланд, 1992 г.):

$$\langle m(H,t+1)\rangle \geq m(H,t)\frac{f(H,t)}{\bar{f}(t)}\left[1-p_c\frac{\delta(H)}{l-1}\right](1-p_m)^{o(H)}. \qquad (2.4)$$

Как уже было отмечено, классическая теорема шаблонов несовершенна для ГА, модели которых отличаются от канонического ГА. Кроме этого, в теореме шаблонов практически не учитывается то обстоятельство, что кроссовер и мутация могут не только разрушать шаблон, но создавать его из других шаблонов. Поэтому в теореме шаблонов присутствует знак неравенства.

Еще одним недостатком теоремы шаблонов является то, что она позволяет рассчитать долю шаблонов в популяции только для следующего поколения. Т.е. при попытке подсчитать число строк, которые соответствуют данному шаблону через несколько поколений с использованием теоремы шаблонов, мы не получим верного ответа. Однако приведенные аргументы не уменьшают значения теоремы шаблонов. Это была, наверное, первая серьезная и успешная попытка понять, как и почему работают ГА. Сама же теория шаблонов на сегодняшний день является одним их самых распространенных инструментов анализа ГА.

Ниже приведено небольшое описание некоторых моделей генетического алгоритма. При реализации собственного алгоритма придерживаться этих моделей необязательно, поскольку многое зависит от решаемой задачи, однако описанные принципы (например, параллелизм) могут оказаться полезными.

1. Canonical GA (J. Holland)

Такая модель алгоритма является классической. Она была предложена Джоном Холландом в его работе «Адаптация в естественных и искусственных средах» (1975 г.). Часто можно встретить описание простого ГА (**Simple GA, D. Goldberg**), он отличается от канонического тем, что использует либо метод рулетки, либо турнирный отбор.

Модель канонического ГА имеет следующие характеристики:

– Фиксированный размер популяции.
– Фиксированную разрядность генов.
– Пропорциональный отбор.
– Особи для скрещивания выбираются случайным образом.
– Одноточечный кроссовер и одноточечная мутация.
– Следующее поколение формируется из потомков текущего поколения без «элитизма». Потомки занимают места своих родителей.

2. Genitor (D. Whitley)

В такой модели используется специфическая стратегия отбора: сначала популяция инициализируется, ее особи оцениваются. Затем выбираются случайным образом две особи, скрещиваются, причем получается только один потомок, который оценивается и занимает место менее приспособленной особи. После этого вновь случайным образом выбираются две особи, и их потомок занимает место особи с низкой приспособленностью. Таким образом, на каждом шагу в популяции обновляется только одна особь.

Можно выделить следующие характерные черты:

– Фиксированный размер популяции.
– Фиксированная разрядность генов.
– Особи для скрещивания выбираются случайным образом.
– Ограничений на тип кроссовера и мутации нет.
– В результате скрещивания особей выходит один потомок, который занимает место наименее приспособленной особи.

3. Hybrid algorithm (L. «Dave» Davis)

Использование гибридного алгоритма позволяет объединить преимущества ГА с преимуществами классических методов. Дело в том, что ГА являются робастными алгоритмами. Поэтому возникла идея использовать ГА на начальном этапе для эффективного сужения пространства поиска вокруг глобального экстремума, а затем, взяв лучшую особь, применить один из «классических» методов оптимизации.

Характеристики алгоритма:

– Фиксированный размер популяции.

– Фиксированная разрядность генов.

– Любые комбинации стратегий отбора и формирования следующего поколения.

– Ограничений на тип кроссовера и мутации нет.

– ГА применяется на начальном этапе, а затем в работу включается классический метод оптимизации.

4. Island Model GA

Представим себе следующую ситуацию. В некотором океане есть группа близлежащих островов, на которых живут популяции особей одного вида. Эти популяции развиваются независимо, и лишь изредка происходит обмен представителями между популяциями. «Островная модель» ГА использует описанный принцип для поиска решения. Вариант, безусловно, интересен и является одной из разновидностей параллельных ГА.

Такая модель генетического алгоритма имеет следующие свойства:

– Наличие нескольких популяций, как правило, одинакового фиксированного размера.

– Фиксированная разрядность генов.

– Любые комбинации стратегий отбора и формирования следующего поколения в каждой популяции. Можно сделать так, что в разных популяциях будут использоваться различные комбинации стратегий, хотя даже один вариант дает разнообразные решения на разных «островах».

– Ограничений на тип кроссовера и мутации нет.

– Случайный обмен особями между «островами». Если миграция будет слишком активной, то особенности «островной» модели будут сглажены, она будет не очень сильно отличаться от моделей ГА без параллелизма.

5. CHC (Eshelman)

CHC расшифровывается как Cross-population selection, Heterogenous recombination and Cataclysmic mutation. Данный алгоритм достаточно быстро сходится из-за того, что в нем нет мутаций, используются популяции небольшого размера, и отбор особей в следующее поколение ведется и между родительскими особями, и между их потомками. В силу этого после установления некоторого решения алгоритм для перезагрузки, подавляющей особь, копируется в новую популяцию, а особи, которые остались, сильной мутацией мутируют примерно треть битов, существующих в хромосоме, и поиск повторяется. Специфической чертой является стратегия скрещивания: все особи разбиваются на пары, причем скрещиваются только те пары, в которых хромосомы особей существенно различны. При скрещивании используется так называемый HUX-оператор (Half Uniform Crossover) – это разновидность однородного кроссовера, но в нем к каждому потомку попадает ровно половина битов хромосомы от каждого родителя.

Таким образом, модель обладает следующими свойствами:

– Фиксированный размер популяции.

– Фиксированная разрядность генов.

– Перезапуск алгоритма после нахождения решения.

– Небольшая популяция.

– Особи для скрещивания разбиваются на пары и скрещиваются при условии существенных различий.

– Отбор в следующее поколение проводится между родительскими особями и потомками.

– Используется половинный однородный кроссовер (HUX).

– Макромутации при перезапуске.

Обучение нечетких нейронных сетей на основе генетических алгоритмов

Одной из наиболее актуальных областей применения генетических алгоритмов являются задачи обучения нейронных сетей, в том числе и нечетких, путем подбора адекватных параметров. Общими этапами такого обучения являются:

ШАГ 1. Выделение управляющих параметров задачи обучения.

ШАГ 2. Получение решения при фиксированных значениях параметров.

ШАГ 3. Определение несогласованности полученного и необходимого решений.

ШАГ 4. Выбор новых значений параметров на основе работы генетического алгоритма.

ШАГ 5. Остановка в случае получения удовлетворительного / несогласованного решения, иначе – переход к шагу 2.

Управляющими параметрами обучения нечетких нейронных сетей, которые влияют на качество решения, могут быть выбраны параметры функций принадлежности, а также различная формализация логических правил.

Генетические алгоритмы – не единственный способ решения задач оптимизации. Кроме него, существуют два основных подхода для решения таких задач – перебор и локально-градиентный, каждый из которых имеет свои преимущества и недостатки.

Сравним стандартные подходы с генетическими алгоритмами на примере задачи коммивояжера (TSP – Travelling Salesman Problem), суть которой заключается в нахождении кратчайшего замкнутого пути обхода городов, заданных своими координатами.

Уже для 30 городов поиск оптимального пути представляет собой сложную задачу, побудившей к развитию новых методов (в том числе нейронных сетей и генетических алгоритмов) (**Рис. 2.6**).

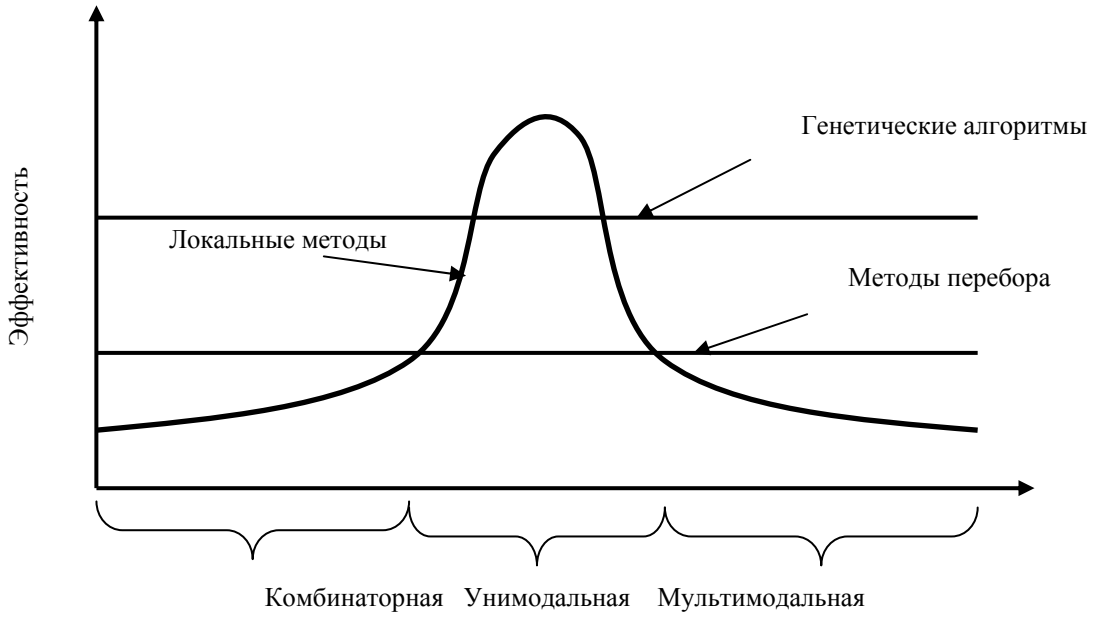

Рис. 2.6 – Эффективность генетических алгоритмов

Каждый вариант решения (для 30 городов) – это числовой ряд. Таким образом, в этой задаче 30 параметров, причем не все комбинации значений допустимы. Метод перебора – наиболее простой в программировании. Для поиска оптимального решения (максимума целевой функции) нужно последовательно вычислить значения целевой функции

во всех возможных точках, запоминая максимальное из них. Недостатком метода является большая вычислительная сложность: нужно просчитать длины более 1030 вариантов путей, это практически невозможно. Однако если перебор всех вариантов за разумное время возможен, то такое найденное решение является оптимальным.

Второй подход базируется на методе градиентного спуска. Сначала выбираются некоторые случайные значения параметров, а затем эти значения постепенно изменяют, добиваясь наибольшей скорости роста целевой функции. При достижении локального максимума такой метод останавливается, поэтому для поиска глобального оптимума нужны дополнительные меры. Градиентные методы работают быстро, но не гарантируют оптимальности найденного решения. Они идеальны для применения так называемых унимодальных задач, где целевая функция имеет единственный локальный максимум (он же – глобальный). Однако задача коммивояжера таковой не является.

Практические задачи – как правило, мультимодальные и многомерные, т.е. содержат много параметров. Для них не существует универсальных методов, которые позволяют достаточно быстро найти точные решения. Комбинируя методы перебора, можно получить приближенные решения, точность которых будет расти с увеличением времени расчета. Генетический алгоритм представляет собой именно такой комбинированный метод. Механизмы скрещивания и мутации реализуют переборную часть метода, а отбор лучших решений – градиентный спуск. На **Рис. 2.6** показано, что такое сочетание обеспечивает устойчиво хорошую эффективность генетического поиска для любых типов оптимизационных задач.

Таким образом, если на некотором множестве задана сложная функция нескольких переменных, то генетический алгоритм за разумное время находит значение функции, достаточно близкое к оптимальному. Задавая время расчета, можно получить одно из лучших решений, которое реально получить за это время. При этом реализация ГА на существующих языках программирования – относительно несложная.

3 САМОСТОЯТЕЛЬНОЕ ОСВОЕНИЕ СИНТЕЗА ГИБРИДНЫХ (НЕЧЁТКИХ) НЕЙРОННЫХ СЕТЕЙ

Графический интерфейс гибридных (нечетких) нейронных систем в CASE-системе MatLAB вызывается функцией (из режима командной строки) **anfisedit**.

Выполнение функции приводит к появлению окна редактора гибридных систем (**ANFIS Editor** или **ANFIS-редактор**) (**Рис. 3.1**):

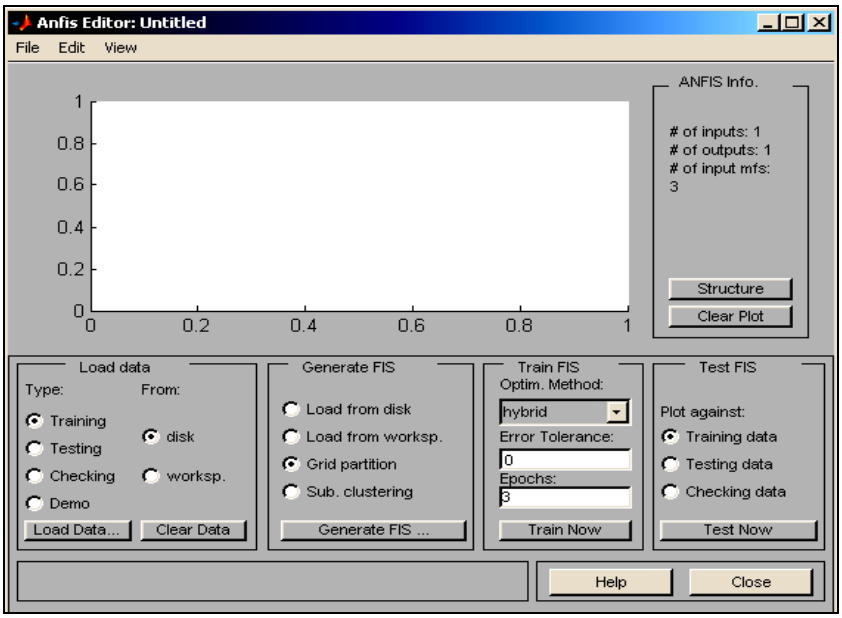

Рис. 3.1 – Окно редактора гибридных (нечетких) нейронных систем

С помощью данного редактора осуществляется создание или загрузка структуры гибридной системы, просмотр структуры, настройки ее параметров, проверка функционирования такой системы. Создание структуры, настройки параметров и проверка осуществляются по выборкам (наборам данных) – **учебной (Training data)**, **проверочной (Checking data)** и **тестовой (Testing data)**, которые предварительно должны быть представлены в виде текстовых файлов (с расширением dat и разрешением табуляциями). Рекомендуется создавать их в текстовом редакторе «Блокнот». Первые столбики соответствуют входным переменным, а последний (левый) – единственной исходной переменной; количество строк в таких файлах равно количеству образцов (примеров) (**Рис. 3.2**). Четких рекомендаций по объемам указанных выборок не существует, очевидно, лучше исходить из принципа «чем больше, тем лучше». Учебные и проверочная выборки непосредственно задействуются в процессе настройки параметров гибридной сети.

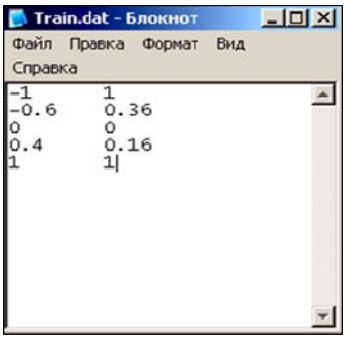

Рис. 3.2 – Создание в текстовом редакторе «Блокнот» выборок для ANFIS-editor

Проверочная выборка применяется для выяснения ситуации: имеет место переобучение сети, при котором ошибка для учебной последовательности стремится к нулю, а для проверочной – возрастает. Впрочем, наличие проверочной выборки не является строго необходимой, она только желательна. Тестирующая выборка служит для проверки качества функционирования обученной сети.

Меню и опции редактора. Пункты меню **File** и **View** в общем идентичны аналогичным пунктам **FIS-редактора**, за тем исключением того, что здесь работа может происходить только с алгоритмом нечеткого вывода **Sugeno**.

Пункт меню **Edit** содержит единственный подпункт – **Undo** (отменить выполненное действие).

Набор опций **Load data** (загрузить данные) в нижней левой части окна редактора включает в себя:

– тип (**Type**) загружаемых данных (для обучения – **Training,** для тестирования – **Testing,** для проверки – **Checking,** демонстрационные – **Demo**);

– место, откуда должны загружаться данные (с диска – **disk** или рабочей области **MATLAB-workspace**).

К данным опциям относятся две кнопки, нажатие на которые приводит к необходимым действиям: **Load Data ...** (загрузить данные) и **Clear Data** (очистить, т.е. стереть введенные данные).

Следующая группа опций (в средине нижней части окна **ANFIS-редактора**) объединена под именем **Generate FIS** (создание нечеткой системы вывода).

Данная группа включает в себя опции:

– загрузка структуры системы с диска (**Load from disk**);

– загрузка структуры системы с рабочей области MATLAB (**Load from worksp**);

– разбивка (распределение) областей определения входных переменных (аргументов) на подобласти – независимо для каждого аргумента (**Grid partition**);

– разбивка всей области определения аргументов (входных переменных) на подобласти – в комплексе для всех аргументов (**Subtract clustering** или **Sub.clustering**), а также кнопку **Generate FIS**, нажатие которой приводит к процессу создания гибридной системы.

Следующая группа опций – **Train FIS** (обучение нечеткой системы вывода) – позволяет определить метод «обучения» (**Optim. Method**) системы (т.е. метод настройки ее параметров) – гибридный (**hybrid**) или обратного распространения ошибки (**backprop**), установить уровень текущей суммарной (по всем образцам) ошибки обучения (**Error Tolerance**), при достижении которой процесс обучения заканчивается; при достижении определенного уровня ошибки обучения или при проведении заданного количества циклов.

Кнопка **Train Now** (начать обучение) активизирует процесс настройки параметров гибридной сети. В правом верхнем углу окна **ANFIS-редактора** выдается информация (**ANFIS Info**) относительно спроектированной системы: количество входов, выходов, функций принадлежности входов; нажатие кнопки **Structure** (структура) позволяет увидеть структуру сети.

Кнопка **Clear** (очистить) позволяет стереть все результаты.

Опции **Test FIS** в правом нижнем углу окна позволяют провести проверку и тестирование созданной и обученной системы с выводом результатов в виде графиков.

Соответствующие графики: для обучающей выборки – **Training data**; тестовой выборки – **Testing data**; проверочной выборки – **Checking data**.

Кнопка **Test Now** позволяет запустить указанные процессы. Работу с редактором рассмотрим на примере восстановления зависимости $y = x^2$ (см. **Рис. 3.2**).

Предположим, что эти данные сохранены в файле **Train.dat**.

Создание и проверку системы, как и раньше, проведем поэтапно:

В окне **ANFIS-редактора** выберем тип загружаемых данных **Training** и нажмем кнопку **Load data**.

В следующем стандартном окне диалога укажем местоположение и имя файла. Его открытие приводит к появлению в графической части окна редактора набора точек, отвечающих введенным данным (**Рис. 3.3**):

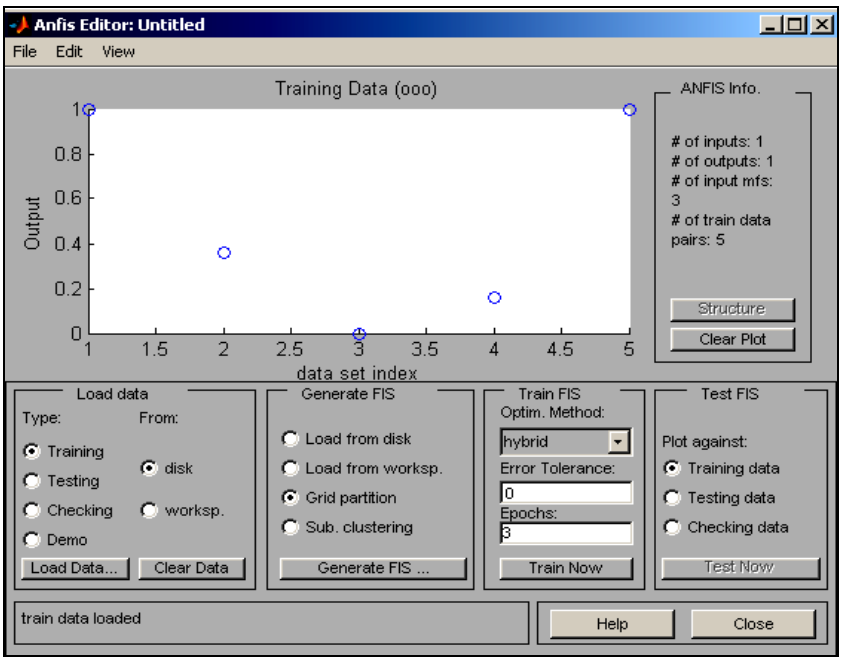

Рис. 3.3 – Окно ANFIS-редактора после загрузки обучающей выборки

В группе опций **Generate FIS** по умолчанию активизирована опция **Grid partition**. Не будем ее менять и нажмем кнопку **Generate FIS**, после чего появится диалоговое окно для задания числа и типов функций принадлежности (**Рис. 3.4**):

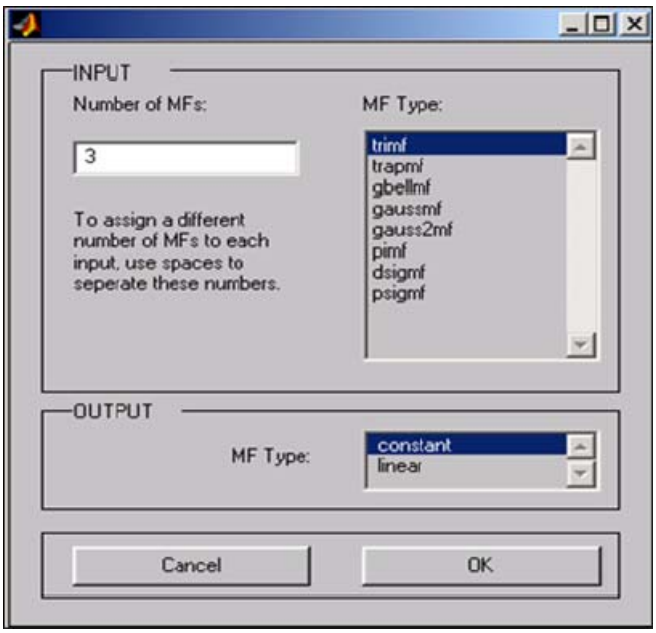

Рис. 3.4 – Окно задания функций принадлежности

Сохраним все установки по умолчанию, согласившись с ними нажатием кнопки **ОК**. Произойдет возврат в основное окно **ANFIS-редактора**.

Теперь структура гибридной сети создана, ее графический вид можно просмотреть с помощью кнопки **Structure** (**Рис. 3.5**):

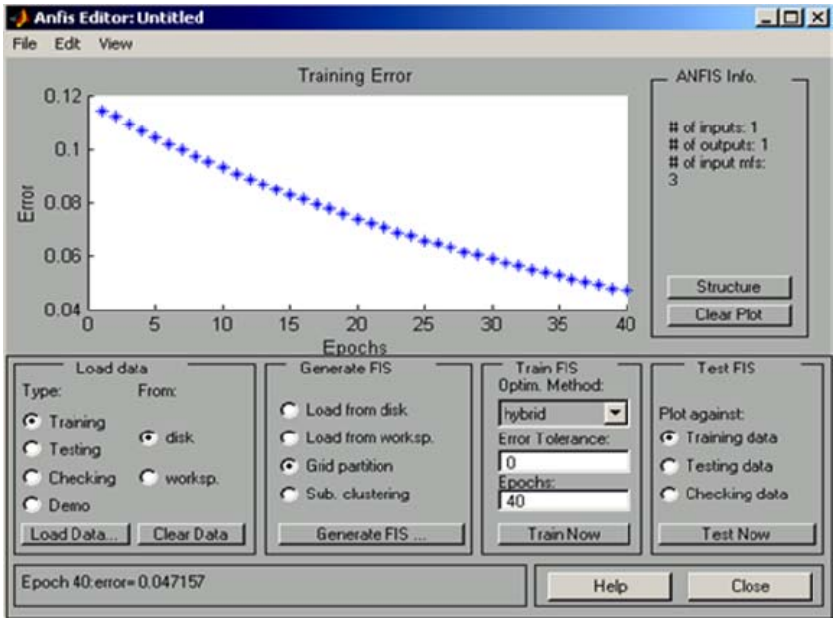

Рис. 3.5 – Структура созданной гибридной сети

Перейдем к опциям **Train FIS**. Не будем изменять настройки параметров, которые задаются по умолчанию: метод (**hybrid** – гибридный) и уровень ошибки (0), но количество циклов обучения изменим на 40, после чего нажмем кнопку начала процесса обучения (**Train Now**).

Результат в виде графика ошибки сети, в зависимости от числа проведенных циклов обучения (из которого следует, что фактически обучение закончилось после пятого цикла), представлен на **Рис. 3.6, 3.7, 3.8**.

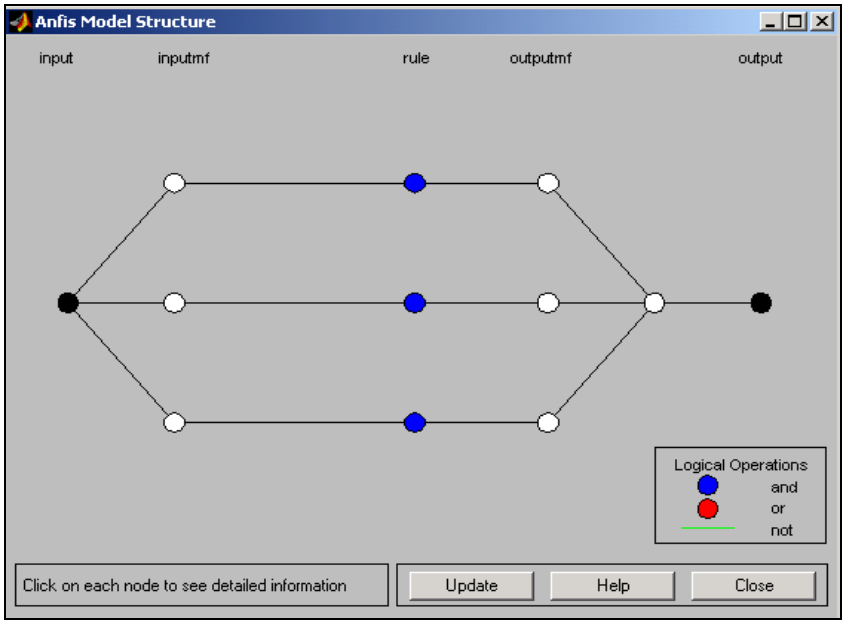

Рис. 3.6 – Результат обучения сети

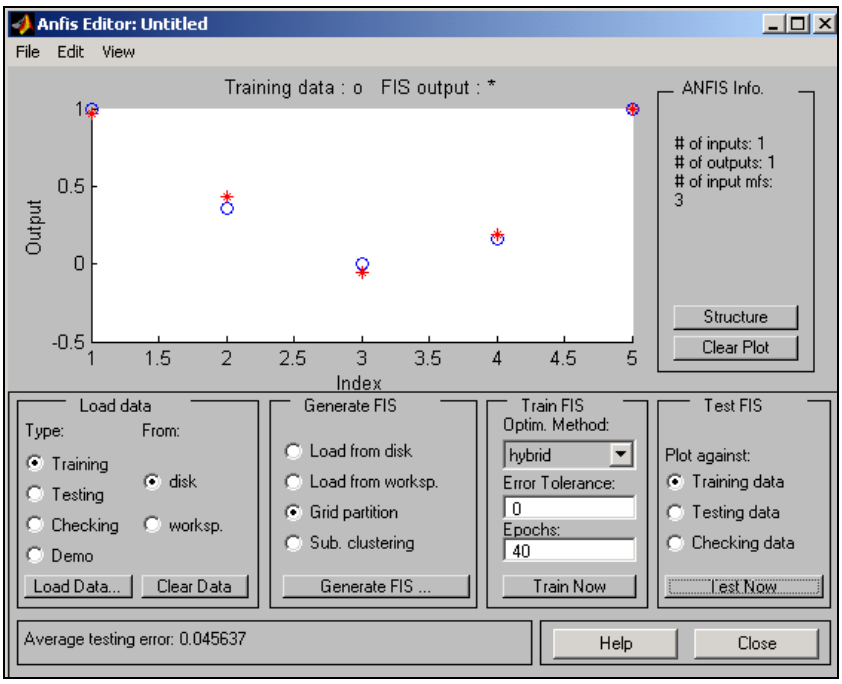

Рис. 3.7 – Результат тестирования обученной системы

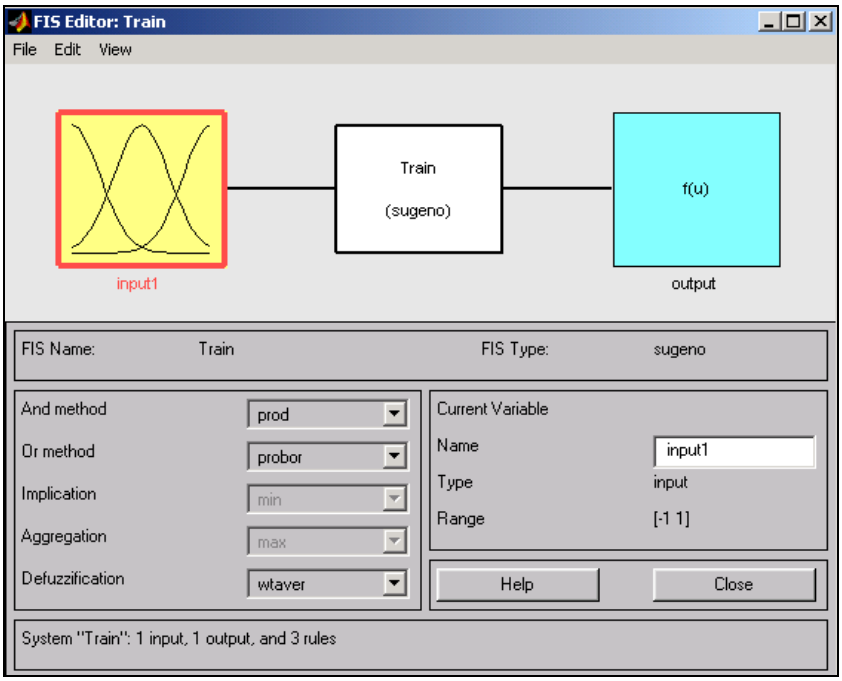

Рис. 3.8 – Загрузка созданного файла Train командой fuzzy из командной строки

Создание и исследование нечетких нейронных сетей из командной строки

Функция, которая создает и/или учит гибридные нейронные сети с архитектурой **ANFIS**, записывается **anfis**.

Значение возможных аргументов функции:

trnData – матрица данных для обучения сети (обучающая выборка), последний столбец соответствует исходной переменной, другие столбцы – входным переменным;

имя – идентификатор создаваемой гибридной сети; если структура системы нечеткого вывода с таким идентификатором уже создана, то она будет использована для

настройки числовых параметров, в противном случае структура будет создана при выполнении функции;

trnOpt – вектор опций обучения с элементами: **trnOpt (l)**: количество циклов обучения (по умолчанию 10), **trnOpt (2)**: целевой уровень ошибки обучения (по умолчанию 0), **trnOpt (3)**: начальный шаг алгоритма обучения (по умолчанию 0,01), **trnOpt (4)**: коэффициент уменьшения шага (по умолчанию 0,9), **trnOpt (5)**: коэффициент увеличения шага (по умолчанию 1,1);

dispOpt – вектор опций вида выводимой информации (по умолчанию все элементы – единичные с элементами: **dispOpt (l)** (ANFIS-информация функции принадлежности, такая, как число входов), **dispOpt (2)** (ошибка), **dispOpt (S)** (шаг восстановления (корректировки) по каждому параметру), **dispOpt (4)** (конечные результаты).

Процесс обучения сети заканчивается, когда выполнено заданное число циклов обучения или ошибка обучения уменьшилась до заданного уровня. При отсутствии некоторых аргументов их значения принимаются по умолчанию.

Выходные параметры функции:

ошибка_об – массив (вектор) значений ошибок для обучающей выборки;

ошибка_п – массив (вектор) значений ошибок для проверочной выборки;

шаг – массив значений шага в алгоритме обучения;

имя! – идентификатор (имя) сети, созданной, исходя из минимальной ошибки обучения;

имя2 – идентификатор (имя) сети, созданной, исходя из минимальной ошибки для проверочной выборки.

Пример:

```
>> x=(0:0.1:10)';
>> y=sin(2*x)./exp(x/5);
>> trnData=[x y];
>> a=anfis(trnData);
ANFIS info:
Number of nodes: 12
Number of linear parameters: 4
Number of nonlinear parameters: 6
Total number of parameters: 10
Number of training data pairs: 101
Number of checking data pairs: 0
Number of fuzzy rules: 2
Start training ANFIS ...
   1  0.313942
   2  0.31388
   3  0.313817
   4  0.313753
   5  0.313689
Step size increases to 0.011000 after epoch 5.
   6  0.313624
   7  0.313552
   8  0.313479
   9  0.313406
Step size increases to 0.012100 after epoch 9.
  10  0.313332
Designated epoch number reached --> ANFIS training completed at epoch 10.
>> plot(x,y,x,evalfis(x,a),'-.');
>> legend('Real','ANFIS') (Рис. 3.9):
```

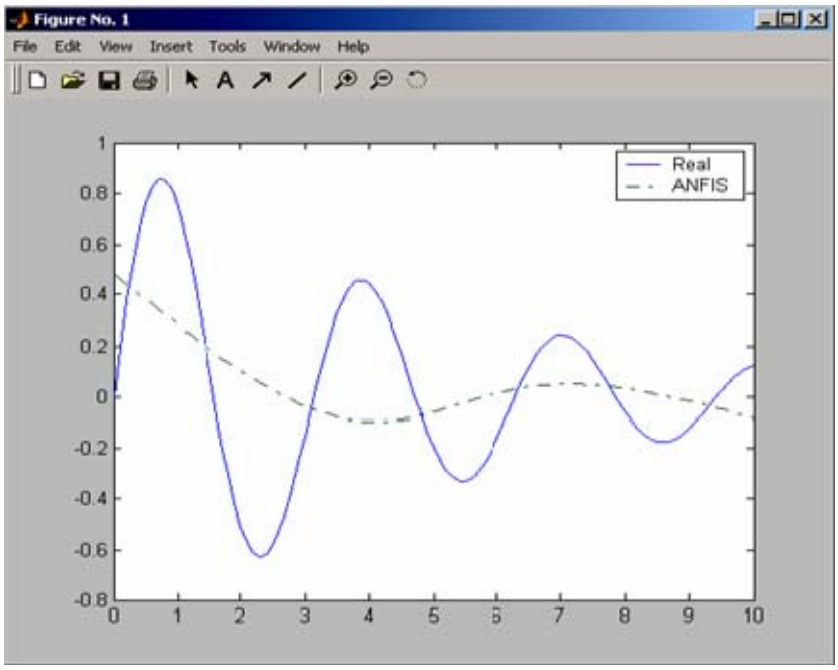

Рис. 3.9 – Обучающая выборка и выход системы

Пример:

```
>> plot(x,y,x,evalfis(x,a),'-.');
>> legend('ANFIS')
>> legend('ANFIS','Real')
>> legend('Real','ANFIS')
>> x=(0:0.1:10)';
>> y=sin(2*x)./exp(x/5);
>> trnData=[x y];
>> numMFs=5;
>>
>> mfType='gbellmf';
>> epoch_n=20;
>> in_fismat=genfis1(trnData, numMFs, mfType);
>> out_fismat=anfis(trnData, in_fismat, 20);
Start training ANFIS ...
   1  0.0694086
   2  0.0680259
   3  0.066663
   4  0.0653198
   5  0.0639961
Step size increases to 0.011000 after epoch 5.
   6  0.0626917
   7  0.0612787
   8  0.0598881
   9  0.0585193
Step size increases to 0.012100 after epoch 9.
  10  0.0571712
  11  0.0557113
  12  0.0542741
  13  0.052858
```

Step size increases to 0.013310 after epoch 13.
14 0.0514612
15 0.0499449
16 0.0484475
17 0.0469667
Step size increases to 0.014641 after epoch 17.
18 0.0455003
19 0.0439022
20 0.0423184
Designated epoch number reached --> ANFIS training completed at epoch 20.
>> plot(x,y,x,evalfis(x,a),'-.');
>> legend('Real','ANFIS') (**Рис. 3.10**).

Во втором примере при проектировании гибридной системы заранее заданы некоторые значения (количество и тип функций принадлежности, количество циклов обучения); в первом примере для той же обучающей выборки все настройки при проектировании системы выбраны по умолчанию. Отличия функционирования двух систем наглядно отображаются на **Рис. 3.9**, **3.10**.

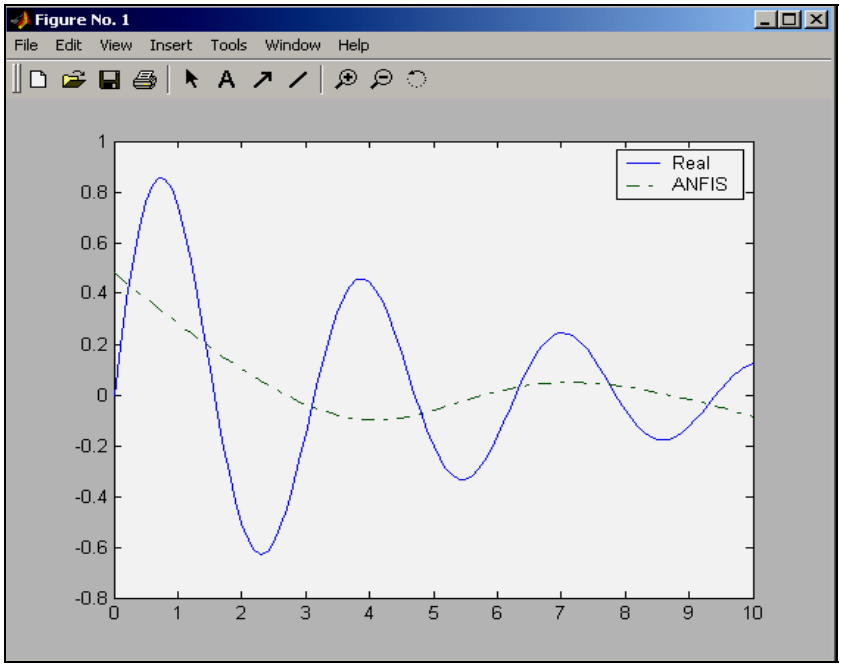

Рис. 3.10 – Обучающая выборка и выход системы flsinat

Задание. Выбрать тему финального проекта и согласовать её с преподавателем.

Примерный перечень тем:

Моделирование и проектирование нечёткой нейронной сети учета деятельности системы «Терминал приема платежей» с помощью CASE-средств MatLAB;

Моделирование и проектирование нечёткой нейронной сети «Разработка ПО для оптимизации складского учета» с помощью CASE-средств;

Моделирование и проектирование нечёткой нейронной сети учета деятельности строительной компании с помощью CASE-средств MatLAB;

Моделирование и проектирование нечёткой нейронной сети учета валютных операций в банке с помощью CASE-средств MatLAB;

Моделирование и проектирование нечёткой нейронной сети учёта деятельности интернет-сервиса по созданию сайтов с помощью CASE-средств MatLAB;

Моделирование и проектирование нечёткой нейронной сети аналитического отдела банка с помощью CASE-средств MatLAB;

Моделирование и проектирование нечёткой нейронной сети учета ценных бумаг с помощью CASE-средств MatLAB;

Моделирование и проектирование нечёткой нейронной сети поиска маршрутов городского транспорта с помощью CASE-средств MatLAB.

Построить таблицу данных с числовыми показателями, характеризующими процесс (**Рис. 3.11**):

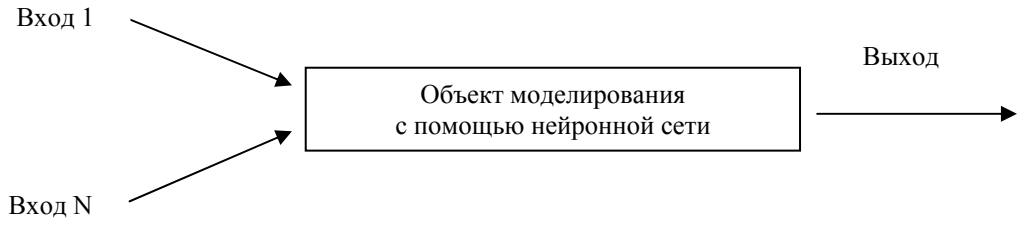

Рис. 3.11 – Структура данных для построения нечёткой нейронной сети

Синтезировать систему поддержки принятия решений на основе нечёткой нейронной сети.

Сделать выводы касательно адекватности созданной нечёткой нейронной сети (критерий – относительная среднеквадратическая ошибка).

4 САМОСТОЯТЕЛЬНОЕ ОСВОЕНИЕ СИСТЕМ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Пусть *a* и *b* – это два индивидуума в популяции, т.е. векторы, от которых наша функция зависит.

Тогда потомок *c* вычисляется по формуле:

$$c = a + k * (a–b),$$

где *k* – это некоторый коэффициент (который может зависеть от || *a-b* || и расстояния между векторами). В этой модели мутация – это добавление к индивидууму случайного вектора малой длины.

Листинг программной реализации ГА:

v1, v2 – показатели на родительские особи.

c1, c2 – потомки.

```
// классический кроссинговер
q=rand()*(EBitSize*4-1)/RAND_MAX;
for(t=0;t<q;t++)
{
  SetBit(c1->r,t,GetBit(v1->r,t));
  SetBit(c2->r,t,GetBit(v2->r,t));
}
for(t=q;t<EBitSize*4;t++)
{
  SetBit(c1->r,t,GetBit(v2->r,t));
  SetBit(c2->r,t,GetBit(v1->r,t));
}
// 2х точковий кроссинговер
t1=rand()*(EBitSize*4-1)/RAND_MAX;
t2=rand()*(EBitSize*4-1)/RAND_MAX;
t=t1;
while(t!=t2)
{
  SetBit(c1->r,t,GetBit(v1->r,t));
  SetBit(c2->r,t,GetBit(v2->r,t));
  t++;
  if(t==EBitSize*4)t=0;
}
while(t!=t1)
{
  SetBit(c1->r,t,GetBit(v2->r,t));
  SetBit(c2->r,t,GetBit(v1->r,t));
  t++;
  if(t==EBitSize*4)t=0;
}
// уніфікований кроссинговер
for(t=0;t<EBitSize*4;t++)
{
  if(rand()<(RAND_MAX/2))
  {
    SetBit(c1->r,t,GetBit(v1->r,t));
```

```
      SetBit(c2->r,t,GetBit(v2->r,t));
    }
    else
    {
      SetBit(c1->r,t,GetBit(v2->r,t));
      SetBit(c2->r,t,GetBit(v1->r,t));
    }
  }
```

Объяснение: EBitSize – количество **битов** на одну переменную.

Процедуры **GetBit**, **SetBit** и **InvertBit** – соответственно читают, устанавливают и инвертируют указанный бит в векторе (хромосоме). Если Вы храните каждый бит в **byte**, и хромосома соответственно представлена массивом таких **byte**, процедуру можно сразу изменить на обращение к массиву.

Задание. Выбрать тему финального проекта и согласовать её с преподавателем.

Примерный перечень тем:

Моделирование и проектирование системы генетического алгоритма оптимизации учёта деятельности грузоперевозок товаров;

Моделирование и проектирование нейронной сети учёта деятельности отдела маркетинга предприятия;

Моделирование и проектирование системы генетического алгоритма оптимизации работы букмекерской конторы;

Моделирование и проектирование системы генетического алгоритма оптимизации функционирования структуры «Банк Инноваций»;

Моделирование и проектирование системы генетического алгоритма оптимизации поиска вакансий;

Моделирование и проектирование системы генетического алгоритма оптимизации учета инновационных продуктов региона;

Моделирование и проектирование системы генетического алгоритма оптимизации грузоперевозок логистического центра.

Построить таблицу данных с числовыми показателями, характеризующими процесс (**Рис. 4.1**):

Рис. 4.1 – Структура данных для построения системы генетического алгоритма

Синтезировать систему генетического алгоритма на любой языке программирования.

Сделать выводы касательно адекватности созданной системы генетического алгоритма (критерий – относительная среднеквадратическая ошибка).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Мацяшек, Л. А. Анализ требований и проектирование систем: разработка информационных систем с использованием UML : научно-популярная литература / Л. А. Мацяшек ; пер. с англ. – М. : Вильямс, 2002. – 432 с.
2. Амосов, Н. М. Нейрокомпьютеры и интеллектуальные роботы / Н. М. Амосов [и др.]. – М. : АН УССР Ин-т кибернетики. – К. : Наукова думка, 1991. – 272 с.
3. Малпас, Дж. Реляционный язык ПРОЛОГ и его применение / Дж. Малпас. – М. : Наука, 1990. – 464 с.
4. Адаменко, А. Н. Логическое программирование и Visual Prolog / А. Н. Адаменко, А. М. Кучуков. – СПб. : БХВ-Петербург, 2003. – 993 с.
5. Усков, А. А. Интеллектуальные технологии управления. Искусственные нейронные сети и нечеткая логика / А. А. Усков, А. В. Кузьмин. – М. : Горячая линия. – Телеком, 2004. – 143 с.
6. Рутковский, Ляшек Методы и технологии искусственного интеллекта / Ляшек Рутковский. – М. : Горячая линия – Телеком, 2010. – 520 с.
7. Ясницкий, Л. Н. Введение в искусственний интеллект / Л. Н. Ясницкий. – М. : Академия, 2008. – 176 с.
8. Трофимов, С. А. CASE-технологии: практическая работа в Rational Rose : научно-техническое издание / С. А. Трофимов. – 2-е изд. – М. : Бином-Пресс, 2002. – 288 с.
9. Грабауров, В. А. Информационные технологии для менеджеров : производственно-практическое издание / В. А. Грабауров. – М. : Финансы и статистика, 2001. – 368 с.
10. Пилецкий, И. И. Проектирование, разработка и сопровождение баз данных с использованием CASE-средств: пособие по курсу «Методы и технологии программирования» для студентов специальности 1-31 03 04 «Информатика» всех форм обучения / И. И. Пилецкий ; Министерство образования Республики Беларусь, Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники». – Минск : БГУИР, 2009. – 116 с.
11. Корчемний, М. О. Нейроні мережі : навч. посіб. / М. О. Корчемний, В. П. Лисенко, М. В. Чапний, В. М. Штепа. – К. : «Аграр Медіа Груп», 2010. – 136 с.
12. Ручкин, В. Н. Универсальный искусственный интеллект и экспертные системы / В. Н. Ручкин, В. А. Фулин. – СПб. : БХВ-Петербург, 2009. – 240 с.
13. Озерова, В. П. Менеджер и управленческая информация : учебное пособие по курсу «Компьютерные информационные технологии» для студентов заочной формы обучения / В. П. Озерова. – Министерство образования Республики Беларусь, Белорусский государственный экономический университет. – Минск : БГЭУ, 2003. – 51 с.
14. Кратчен,°Ф. Введение в Rational Unified Process. – 2-е изд. ; пер. с англ. – М. : Вильямс, 2002. – 240 с.
15. Нейронные сети. STATISTICA Neural Networks: Методология и технологии современного анализа данных / под редакцией В. П. Боровикова. – 2-е изд., перераб. и доп. – М. : Горячая линия – Телеком, 2008. – 288 с.
16. Круглов, В. В. Нечеткая логика и искусственные нейронные сети / В. В. Круглов, М. И. Дли, Р. Ю. Голунов. – М. : Физматлит, 2000. – 224 с.

Учебное издание

Владимир Николаевич Штепа
Алёна Григорьевна Штепа

**Распределённые информационные системы.
Нечёткие нейронные сети. Генетический алгоритм**

Методические рекомендации по выполнению лабораторных работ

Ответственный за выпуск *П. Б. Пигаль*

Редактор *Т. И. Андросюк*
Корректор *Ю. В. Цвикевич*

Подписано в печать 20.05.2019 г. Формат 60×84/8.
Бумага офсетная. Гарнитура «Таймс». Ризография.
Усл. печ. л. 3,95. Уч.-изд. л. 1,39.
Тираж 47 экз. Заказ № 288.

Отпечатано в редакционно-издательском отделе
Полесского государственного университета.
225710, г. Пинск, ул. Днепровской флотилии, 23.