8. Учитель

Этот компонент не является столь универсальным как задачник, оценка или нейронная сеть, поскольку существует ряд алгоритмов обучения жестко привязанных к архитектуре нейронной сети. Примерами таких алгоритмов могут служить обучение (формирование синаптической карты) сети Хопфилда, обучение сети Кохонена и ряд других аналогичных сетей. Однако в главе «Описание нейронных сетей» приводится способ формирования сетей, позволяющий обучать сети Хопфилда и Кохонена методом обратного распространения ошибки. Описываемый в этой главе стандарт компонента учитель ориентирован в первую очередь на обучение двойственных сетей (сетей обратного распространения ошибки).

8.1 Что можно обучать методом двойственности

Как правило, метод двойственности (обратного распространения ошибки) используют для подстройки параметров нейронной сети. Однако, как было показано в главе «Описание нейронных сетей», сеть может вычислять не только градиент функции оценки по обучаемым параметрам сети, но и по входным сигналам сети. Используя градиент функции оценки по входным сигналам сети можно решать задачу, обратную по отношению к обучению нейронной сети.

Рассмотрим следующий пример. Пусть есть сеть, обученная предсказывать по текущему состоянию больного и набору применяемых лекарств состояние больного через некоторый промежуток времени. Поступил новый больной. Его параметры ввели сети и она выдала прогноз. Из прогноза следует ухудшение некоторых параметров состояния больного. Возьмем выданный сетью прогноз, заменим значения параметров, по которым наблюдается ухудшение, на желаемые значения. Полученный вектор ответов объявим правильным ответом. Имея правильный ответ и ответ, выданный сетью, вычислим градиент функции оценки по входным сигналам сети. В соответствии со значениями элементов градиента изменим значения входных сигналов сети так, чтобы оценка уменьшилась. Проделав эту процедуру несколько раз, получим вектор входных сигналов, порождающих правильный ответ. Далее врач должен определить, каким способом (какими лекарствами или процедурами) перевести больного в требуемое (полученное в ходе обучения входных сигналов) состояние. В большинстве случаев часть входных сигналов не подлежит изменению (например пол или возраст больного). В этом случае эти входные сигналы должны быть помечены как не обучаемые (см. использование маски обучаемости входных сигналов в главе «Описание нейронных сетей»).

Таким образом, способность сетей вычислять градиент функции оценки по входным параметрам сети позволяет решать вполне осмысленную обратную задачу: так подобрать входные сигналы сети, чтобы выходные сигналы удовлетворяли заданным требованиям.

Кроме того, использование нейронных сетей позволяет ставить новые вопросы перед исследователем. В практике группы «НейроКомп» был следующий случай. Была поставлена задача обучить сеть ставить диагноз вторичного иммунодефицита по данным анализов крови и клеточного метаболизма. Вся обучающая выборка была разбита на два класса: больные и здоровые. При анализе базы данных стандартными статистическими методами значимых отличий обнаружить не удалось. Сеть оказалась не способна обучиться. Далее у исследователя было два пути: либо увеличить число нейронов в сети, либо определить, что мешает обучению. Исследователи выбрали второй путь. При обучении сети была применена следующая процедура: как только обучение сети останавливалось из-за невозможности дальнейшего уменьшения оценки, пример, имеющий наихудшую оценку, исключался из обучающего множества. После того, как сеть обучилась решению задачи на усеченном обучающем множестве, был проведен анализ исключенных примеров. Выяснилось, что исключено около половины больных. Тогда множество больных было разбито на два класса – больные1 (оставшиеся в обучающем множестве) и больные2 (исключенные). При таком разбиении обучающей выборки стандартные методы статистики показали значимые различия в параметрах классов. Обучение сети классификации на три класса быстро завершилось полным успехом. При содержательном анализе примеров, составляющих классы больные1 и больные2, было установлено, что к классу болные1 относятся больные на завершающей стадии заболевания, а к классу больные2 – на начальной. Ранее такое разбиение больных не проводилось. Таким образом, обучение нейронной сети решению прикладной задачи поставило перед исследователем содержательный вопрос, позволивший получить новое знание о предметной области.

Подводя итоги этого раздела, можно сказать, что, используя метод двойственности в обучении нейронных сетей можно:

1. Обучать сеть решению задачи.
2. Подбирать входные данные так, чтобы на выходе нейронной сети был заданный ответ.
3. Ставить вопросы о соответствии входных данных задачника постановке нейросетевой задачи.

8.2 Описание алгоритмов обучения

Все алгоритмы обучения сетей методом обратного распространения ошибки опираются на способность сети вычислять градиент функции ошибки по обучающим параметрам. Даже правило Хебба использует вектор псевдоградиента, вычисляемый сетью при использовании зеркального порогового элемента (см. раздел «Пороговый элемент» главы «Описание нейронных сетей»). Таким образом, акт обучения состоит из вычисления градиента и собственно обучения сети (модификации параметров сети). Однако, существует множество не градиентных методов обучения, таких, как метод покоординатного спуска, метод случайного поиска и целое семейство методов Монте-Карло. Все эти методы могут использоваться при обучении нейронных сетей, хотя, как правило, они менее эффективны, чем градиентные методы. Некоторые варианты методов обучения описаны далее в этой главе.

Поскольку обучение двойственных сетей с точки зрения используемого математического аппарата эквивалентно задаче многомерной оптимизации, то в данной главе рассмотрены только несколько методов обучения, наиболее используемых при обучении сетей. Более полное представление о методах оптимизации, допускающих использование в обучении нейронных сетей, можно получить из книг по методам оптимизации (см. например [48, 103, 142]).

8.2.1 Краткий обзор макрокоманд учителя

При описании методов используется набор макросов, приведенный в табл. 2. В табл. 2 дано пояснение выполняемых макросами действий. Все макрокоманды могут оперировать с данными как пространства параметров, так и пространства входных сигналов сети. В первой части главы полагается, что объект обучения установлен заранее. В макросах используются понятия и аргументы, приведенные в табл. 1. Список макрокоманд приведен в табл. 2. При описании методов обучения все аргументы имеют

Таблица 1

Понятия и аргументы макрокоманд учителя (предварительный список)

Название	Смысл
Точка	Точка в пространстве параметров или входных сигналов. Аналогична вектору.
Вектор	Вектор в пространстве параметров или входных сигналов. Аналогичен точке.
Вектор_минимумов	Вектор минимальных значений параметров или входных сигналов.
Вектор_максимумов	Вектор максимальных значений параметров или входных сигналов.
Указатель_на_вектор	Адрес вектора. Используется для передачи векторов в макрокоманды.
Пустой_указатель	Указатель на отсутствующий вектор.

Таблица 2

Список макрокоманд учителя (предварительный).

Название	Аргументы (типы)	Выполняемые действия
Создать_вектор	Указатель_на_вектор	Создает экземпляр вектора с неопределенными значениями. Адрес вектора помещается в Указатель_на_вектор.
Освободить_вектор	Указатель_на_вектор	Освобождает память занятую вектором, расположенным по адресу Указатель_на_вектор.
Случайный_вектор	Указатель_на_вектор	В векторе, на который указывает Указатель_на_вектор, генерируется вектор, каждая из координат которого является случайной величиной, равномерно распределенной на интервале между значениями соответствующих координат векторов Вектор_минимумов и Вектор_максимумов.
Модификация_вектора	Указатель_на_вектор Старый_Шаг Новый_Шаг	Генерирует запрос на модификацию вектора (см. раздел «Провести обучение (Modify)» главы «Описание нейронных сетей»).
Оптимизация_шага	Указатель_на_вектор Начальный_Шаг	Производит подбор оптимального шага (см. рис. 3).
Сохранить_вектор	Указатель_на_вектор	Скопировать текущий вектор в вектор, указанный в аргументе Указатель_на_вектор.
Установить_параметры	Указатель_на_вектор	Скопировать вектор, указанный в аргументе Указатель_на_вектор, в текущий вектор.
Вычислить_оценку	Оценка	Вычисляет оценку текущего вектора. Вычисленную величину складывает в аргумент Оценка.
Вычислить_градиент		Вычисляет градиент функции оценки.

тип, определяемый типом аргумента макрокоманды. Если в описании макрокоманды в табл. 2 тип аргумента не соответствует ни одному из типов, приведенных в табл. 1, то эти аргументы имеют числовой тип.

8.2.2 Неградиентные методы обучения

Среди неградиентных методов рассмотрим следующие методы, каждый из которых является представителем целого семейства методов оптимизации:

1. Метод случайной стрельбы (представитель семейства методов Монте-Карло).
2. Метод покоординатного спуска (псевдоградиентный метод).
3. Метод случайного поиска (псевдоградиентный метод).
4. Метод Нелдера-Мида.

8.2.2.1 Метод случайной стрельбы

Идея метода случайной стрельбы состоит в генерации большой последовательности случайных точек и вычисления оценки в каждой из них. При достаточной длине последовательности минимум будет найден. Запись этой процедуры на макроязыке приведена на рис. 1

Остановка данной процедуры производится по команде пользователя или при выполнении условия, что О1 стало меньше некоторой заданной величины. Существует огромное разнообразие модификаций этого метода. Наиболее простой является метод случайной стрельбы с уменьшением радиуса. Пример процедуры, реализующей этот метод, приведен на рис. 2. В этом методе есть два параметра, задаваемых пользователем:

Число_попыток – число неудачных пробных генераций вектора при одном радиусе.

Минимальный_радиус – минимальное значение радиуса, при котором продолжает работать алгоритм.

Идея этого метода состоит в следующем. Зададимся начальным состоянием вектора параметров. Новый вектор параметров будем искать как сумму начального и случайного, умноженного на радиус, векторов. Если после Число_попыток случайных генераций не произошло уменьшения оценки, то уменьшаем радиус. Если произошло уменьшение оценки, то полученный вектор объявляем начальным и продолжаем процедуру с тем же шагом. Важно, чтобы последовательность уменьшающихся радиусов образовывала расходящийся ряд. Примером такой последовательности может служить использованный в примере на рис. 2 ряд $1/n$.

Отмечен ряд случаев, когда метод случайной стрельбы с уменьшением радиуса работает быстрее градиентных методов, но это скорее исключение, чем правило.

1. Создать_вектор В1
2. Создать_вектор В2
3. Вычислить_оценку О1
4. Сохранить_вктор В1
5. Установить_параметры В1
6. Случайный_вектор В2
7. Модификация_вектора В2, 0, 1
8. Вычислить_оценку О2
9. Если О2<О1 то переход к шагу 11
10. Переход к шагу 5
11. О1=О2
12. Переход к шагу 4
13. Установить_параметры В1
14. Освободить_вектор В1
15. Освободить_вектор В2

Рис. 1. Простейший алгоритм метода случайной стрельбы

1. Создать_вектор В1
2. Создать_вектор В2
3. Вычислить_оценку О1
4. Число_Смен_Радиуса=1
5. Радиус=1/ Число_Смен_Радиуса
6. Попытка=0
7. Сохранить_вктор В1
8. Установить_параметры В1
9. Случайный_вектор В2
10. Модификация_вектора В2, 1, Радиус
11. Вычислить_оценку О2
12. Попытка=Попытка+1
13. Если О2<О1 то переход к шагу 16
14. Если Попытка<=Число_попыток то переход к шагу 8
15. Переход к шагу 18
16. О1=О2
17. Переход к шагу 6
18. Число_Смен_Радиуса= Число_Смен_Радиуса+1
19. Радиус=1/ Число_Смен_Радиуса
20. Если радиус>= Минимальный_радиус то переход к шагу 6
21. Установить_параметры В1
22. Освободить_вектор В1
23. Освободить_вектор В2

Рис. 2. Алгоритм метода случайной стрельбы с уменьшением радиуса

8.2.2.2 Метод покоординатного спуска

Идея этого метода состоит в том, что если в задаче сложно или долго вычислять градиент, то можно построить вектор, обладающий приблизительно теми же свойствами, что и градиент следующим путем. Даем малое положительное приращение первой координате вектора. Если оценка при этом уве-

личилась, то пробуем отрицательное приращение. Далее так же поступаем со всеми остальными координатами. В результате получаем вектор, в направлении которого оценка убывает. Для вычисления такого вектора потребуется, как минимум, столько вычислений функции оценки, сколько координат у вектора. В худшем случае потребуется в два раза большее число вычислений функции оценки. Время же необходимое для вычисления градиента в случае использования двойственных сетей можно оценить как 2-3 вычисления функции оценки. Таким образом, учитывая способность двойственных сетей быстро вычислять градиент, можно сделать вывод о нецелесообразности применения метода покоординатного спуска в обучении нейронных сетей.

8.2.2.3 Подбор оптимального шага

Данный раздел посвящен описанию макрокоманды Оптимизация_Шага. Эта макрокоманда часто используется в описании процедур обучения и не столь очевидна как другие макрокоманды. Поэтому ее текст приведен на рис. 3. Идея подбора оптимального шага состоит в том, что при наличии направления в котором производится спуск (изменение параметров) задача многомерной оптимизации в пространстве параметров сводится к одномерной оптимизации – подбору шага. Пусть заданы начальный шаг (Ш2) и направление спуска (антиградиент или случайное) (Н). Тогда вычислим величину О1 – оценку в текущей точке пространства параметров. Изменив параметры на вектор направления, умноженный на величину пробного шага, вычислим величину оценки в новой точке – О2. Если О2 оказалось меньше либо равно О1, то увеличиваем шаг и снова вычисляем оценку. Продолжаем эту процедуру до тех пор, пока не получится оценка, большая предыдущей. Зная три последних значения величины шага и оценки, используем квадратичную оптимизацию – по трем точкам построим параболу и следующий шаг сделаем в вершину параболы. После нескольких шагов квадратичной оптимизации получаем приближенное значение оптимального шага.

Если после первого пробного шага получилось О2 большее О1, то уменьшаем шаг до тех пор, пока не получим оценку, меньше чем О1. После этого производим квадратичную оптимизацию.

1. Создать_вектор В
2. Сохранить_вктор В
3. Вычислить_оценку О1
4. Ш1=0
5. Модификация_вектора Н, 1, Ш2
6. Вычислить_оценку О2
7. Если О1<О2 то переход к шагу 15
8. Ш3=Ш2*3
9. Установить_параметры В
10. Модификация_вектора Н, 1, Ш3
11. Вычислить_оценку О3
12. Если О3>О2 то переход к шагу 21
13. О1=О2 О2=О3 Ш1=Ш2 Ш2=Ш3
14. Переход к шагу 8
15. Ш3=Ш2 О3=О2
16. Ш2=Ш3/3
17. Установить_параметры В
18. Модификация_вектора Н, 1, Ш2
19. Вычислить_оценку О3
20. Если О2>=О1 то переход к шагу 15
21. Число_парабол=0
22. Ш=((Ш3Ш3-Ш2Ш2)О1+(Ш1Ш1-Ш3Ш3)О2 +(Ш2Ш2-Ш1Ш1)О3)/(2((Ш3-Ш2)О1+(Ш1-Ш3)О2 +(Ш2-Ш1)О3))
23. Установить_параметры В
24. Модификация_вектора Н, 1, Ш
25. Вычислить_оценку О
26. Если Ш>Ш2 то переход к шагу 32
27. Если О>О2 то переход к шагу 30
28. Ш3=Ш2 О3=О2 О2=О Ш2=Ш
29. Переход к шагу 36
30. Ш1=Ш О1=О
31. Переход к шагу 36
32. Если О>О2 то переход к шагу 35
33. Ш3=Ш2 О3=О2 О2=О Ш2=Ш
34. Переход к шагу 36
35. Ш1=Ш О1=О
36. Число_парабол= Число_парабол+1
37. Если Число_парабол<Максимальное_Число_Парабол то переход к шагу 22
38. Установить_параметры В
39. Модификация_вектора Н, 1, Ш2
40. Освободить_вектор В

Рис. 3. Алгоритм оптимизации шага

8.2.2.4 Метод случайного поиска

Этот метод похож на метод случайной стрельбы с уменьшением радиуса, однако в его основе лежит другая идея – сгенерируем случайный вектор и будем использовать его вместо градиента. Этот метод использует одномерную оптимизацию – подбор шага. Одномерная оптимизация описана в разделе

«Одномерная оптимизация». Процедура случайного поиска приведена на рис. 4. В этом методе есть два параметра, задаваемых пользователем.

Число_попыток – число неудачных пробных генераций вектора при одном радиусе.

Минимальный_радиус – минимальное значение радиуса, при котором продолжает работать алгоритм.

Идея этого метода состоит в следующем. Зададимся начальным состоянием вектора параметров. Новый вектор параметров будем искать как сумму начального и случайного, умноженного на радиус, векторов. Если после Число_попыток случайных генераций не произошло уменьшения оценки, то уменьшаем радиус. Если произошло уменьшение оценки, то полученный вектор объявляем начальным и продолжаем процедуру с тем же шагом. Важно, чтобы последовательность уменьшающихся радиусов образовывала расходящийся ряд. Примером такой последовательности может служить использованный в примере на рис. 4 ряд $1/n$.

1. Создать_вектор H
2. Число_Смен_Радиуса=1
3. Попытка=0
4. Радиус=1/ Число_Смен_Радиуса
5. Случайный_вектор H
6. Оптимизация шага H Радиус
7. Попытка=Попытка+1
8. Если Радиус=0 то Попытка=0
9. Если Попытка<=Число_попыток то переход к шагу 4
10. Число_Смен_Радиуса= Число_Смен_Радиуса+1
11. Радиус=1/ Число_Смен_Радиуса
12. Если Радиус>= Минимальный_радиус то переход к шагу 3
13. Освободить_вектор H

Рис. 4. Алгоритм метода случайного поиска

8.2.2.5 Метод Нелдера-Мида

Этот метод является одним из наиболее быстрых и наиболее надежных не градиентных методов многомерной оптимизации. Идея этого метода состоит в следующем. В пространстве оптимизируемых параметров генерируется случайная точка. Затем строится n-мерный симплекс с центром в этой точке, и длиной стороны l. Далее в каждой из вершин симплекса вычисляется значение оценки. Выбирается вершина с наибольшей оценкой. Вычисляется центр тяжести остальных n вершин. Проводится оптимизация шага в направлении от наихудшей вершины к центру тяжести остальных вершин. Эта процедура повторяется до тех пор, пока не окажется, что оптимизация не изменяет положения вершины. После этого выбирается вершина с наилучшей оценкой и вокруг нее снова строится симплекс с меньшими размерами (например $l/2$). Процедура продолжается до тех пор, пока размер симплекса, который необходимо построить, не окажется меньше требуемой точности.

Однако, несмотря на свою надежность, применение этого метода к обучению нейронных сетей затруднено большой размерностью пространства параметров.

8.2.3 Градиентные методы обучения

Изучению градиентных методов обучения нейронных сетей посвящено множество работ [47, 64, 90] (сослаться на все работы по этой теме не представляется возможным, поэтому дана ссылка на работы, где эта тема исследована наиболее детально). Кроме того, существует множество публикаций, посвященных градиентным методам поиска минимума функции [48, 103] (как и в предыдущем случае, ссылки даны только на две работы, которые показались наиболее удачными). Данный раздел не претендует на какую-либо полноту рассмотрения градиентных методов поиска минимума. В нем приведены только несколько методов, применявшихся в работе группой «НейроКомп». Все градиентные методы объединены использованием градиента как основы для вычисления направления спуска.

8.2.3.1 Метод наискорейшего спуска

Наиболее известным среди градиентных методов является метод наискорейшего спуска. Идея этого метода проста: поскольку вектор градиента указывает направление наискорейшего возрастания функции, то минимум следует искать в обратном направлении. Последовательность действий приведена на рис. 5.

1. Вычислить_оценку O2
2. O1=O2
3. Вычислить_градиент
4. Оптимизация шага Пустой_указатель Шаг
5. Вычислить_оценку O2
6. Если O1-O2<Точность то переход к шагу 2

Рис. 5. Метод наискорейшего спуска

Этот метод работает, как правило, на порядок быстрее методов случайного поиска. Он имеет два параметра – Точность, показывающий, что если изменение оценки за шаг метода меньше чем Точность, то обучение останавливается; Шаг – начальный шаг для оптимизации шага. Заметим, что шаг постоянно изменяется в ходе оптимизации шага.

Остановимся на основных недостатках этого метода. Во-первых, эти методом находится тот минимум, в область притяжения которого попадет начальная точка. Этот минимум может не быть глобальным. Существует несколько способов выхода из этого положения. Наиболее простой и действенный – случайное изменение параметров с дальнейшим повторным обучение методом наискорейшего спуска. Как правило, этот метод позволяет за несколько циклов обучения с последующим случайным изменением параметров найти глобальный минимум.

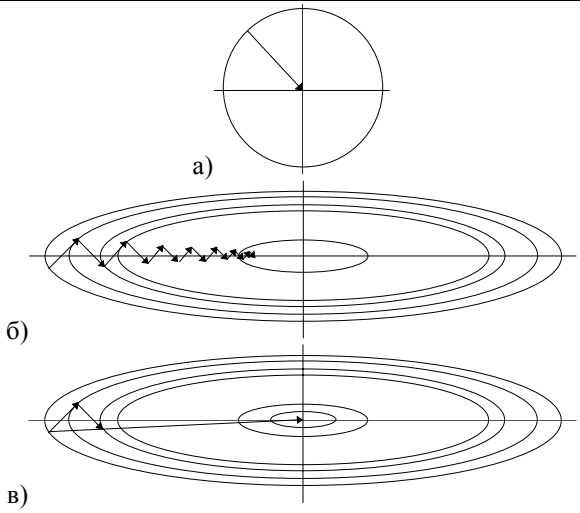

Рис. 6. Траектории спуска при различных конфигурациях окрестности минимума и разных методах оптимизации.

Вторым серьезным недостатком метода наискорейшего спуска является его чувствительность к форме окрестности минимума. На рис. 6а проиллюстрирована траектория спуска при использовании метода наискорейшего спуска, в случае, если в окрестности минимума линии уровня функции оценки являются кругами (рассматривается двумерный случай). В этом случае минимум достигается за один шаг. На рис. 6б приведена траектория метода наискорейшего спуска в случае эллиптических линий уровня. Видно, что в этой ситуации за один шаг минимум достигается только из точек, расположенных на осях эллипсов. Из любой другой точки спуск будет происходить по ломаной, каждое звено которой ортогонально к соседним звеньям, а длина звеньев убывает. Легко показать что для точного достижения минимума потребуется бесконечное число шагов метода градиентного спуска. Этот эффект получил название овражного, а методы оптимизации, позволяющие бороться с этим эффектом – антиовражных.

8.2.3.2 kParTan

Одним из простейших антиовражных методов является метод kParTan. Идея метода состоит в том, чтобы запомнить начальную точку, затем выполнить k шагов оптимизации по методу наискорейшего спуска, затем сделать шаг оптимизации по направлению из начальной точки в конечную. Описание метода приведено на рис 7. На рис 6в приведен один шаг оптимизации по методу 2ParTan. Видно, что после шага вдоль направления из первой точки в третью траектория спуска привела в минимум. К сожалению, это верно только для двумерного случая. В многомерном случае направление kParTan не ведет прямо в точку минимума, но спуск в этом направлении, как правило, приводит в окрестность минимума меньшего радиуса, чем при еще одном шаге метода наискорейшего спуска (см. рис. 6б). Кроме того, следует отметить, что для выполнения третьего шага не потребовалось вычислять градиент, что экономит время при численной оптимизации.

1. Создать_вектор B1
2. Создать_вектор B2
3. Шаг=1
4. Вычислить_оценку O2
5. Сохранить_вектор B1
6. O1=O2
7. N=0
8. Вычислить_градиент
9. Оптимизация_шага Пустой_указатель Шаг
10. N=N+1
11. Если N<k то переход к шагу 8
12. Сохранить_вектор B2
13. B2=B2-B1
14. ШагParTan=1
15. Оптимизация шага B2 ШагParTan
16. Вычислить_оценку O2
17. Если O1-O2<Точность то переход к шагу 5

Рис. 7. Метод kParTan

8.2.3.3 Квазиньютоновские методы

Существует большое семейство квазиньютоновских методов, позволяющих на каждом шаге проводить минимизацию в направлении минимума квадратичной формы. Идея этих методов состоит в том, что функция оценки приближается квадратичной формой. Зная квадратичную форму, можно вычислить ее минимум и проводить оптимизацию шага в направлении этого минимума. Одним из наиболее часто используемых методов из семейства одношаговых квазиньютоновских методов является BFGS метод. Этот метод хорошо зарекомендовал себя при обучении нейронных сетей (см. [29]). Подробно ознакомиться с методом BFGS и другими квазиньютоновскими методами можно в работе [48].

8.3 Стандарт первого уровня компонента учитель

В этом разделе приводится стандарт языка описания компонента учитель. Поскольку часть алгоритмов обучения жестко привязана к архитектуре сети, то в следующем разделе предложен способ опознания «своих» сетей.

8.3.1 Способ опознания сети для методов, привязанных к архитектуре сети

Для опознания типа сети рекомендуется использовать первый параметр сети. Для этого архитектуре сети приписывается уникальный номер, типа Long. Уникальность может поддерживаться, например, за счет использования генератора случайных чисел. Кроме того, при описании параметров сети следует задать отдельный тип параметров для первого параметра и указать минимальную границу равной максимальной и равной номеру архитектуры сети. Также необходимо указать в маске параметров, что этот параметр является необучаемым. Учитель, прежде чем выполнить любую операцию с сетью, читает параметры сети, и проверяет первый параметр сети, интерпретируемый как переменная типа Long, на совпадение с хранимым в учителе номером архитектуры. В случае несовпадения номера в параметрах сети с номером в учителе, учитель генерирует внутреннюю ошибку 601 – несовместимость сети и учителя.

Если учитель работает с сетями любой архитектуры, то процедура опознания архитектуры сети не нужна.

8.3.2 Список стандартных функций

В этом разделе описаны стандартные функции, специфические для компонента учитель. Эти функции соответствуют макросам, использованным в первой части главы. Заголовки функций даны на языке описания учителя.

8.3.2.1 Установить объект обучения (SetInstructionObject)

Заголовок функции:

Function SetInstructionObject (What : Integer; Net : PString) : Logic;

Описание аргументов

What может принимать следующие значения (предопределенные константы, приведенные в табл. 11 главы «Общий стандарт»):

Parameters – для обучения параметров сети;

InSignals – для обучения входных сигналов.

Net – имя нейронной сети, которая будет обучаться.

Возможно обучение одного из двух объектов – параметров сети или входных сигналов. Объект обучения должен быть задан до начала собственно обучения. По умолчанию обучается первая сеть в списке нейронных сетей компонента сеть. При необходимости в качестве объекта обучения может быть задана часть сети (см. раздел «Описание нейронных сетей»). При сохранении учителя в файле сети объект обучения хранится вместе с учителем. Функция возвращает значение истина, если ее выполнение завершено успешно. В противном случае (например, указанная сеть отсутствует в списке сетей компонента сеть) возвращается значение ложь.

8.3.2.2 Создание массива (CreateArray)

Заголовок функции:

Function CreateArray : PRealArray;

Аргументов нет.

Функция возвращает указатель на массив, пригодный для хранения массива обучаемых параметров (входных сигналов) сети. Если массив создать не удалось, то возвращается пустой указатель.

8.3.2.3 Освободить массив (EraseArray)

Заголовок функции:

Function EraseArray(Vec : PRealArray) : Logic;

Описание аргументов

Vec – указатель на массив. При вызове содержит адрес освобождаемого массива.

После выполнения функции в аргументе Vec содержится пустой указатель. В случае невозможности освобождения памяти функция генерирует внутреннюю ошибку 604 – некорректная работа с памятью, передает управление обработчику ошибок, выполнение функции завершается, возвращается значение ложь. В противном случае возвращается значение истина.

8.3.2.4 Случайный массив (RandomArray)

Заголовок функции:

Function RandomArray(Vec : PRealArray) : Logic;

Описание аргументов

Vec – указатель на массив. При входе в макрос содержит адрес существующего массива.

В ходе выполнения функции для каждого элемента массива параметров генерируется случайное значение. Для генерации используется генератор случайных чисел, равномерно распределенных на отрезке от нуля до единицы. После получения случайной величины a она преобразуется по формуле $a' = a(a_{\max} - a_{\min}) - a_{\min}$ к случайной величине, распределенной на отрезке $[a_{\min}, a_{\max}]$. Величины $a_{\min}$ и $a_{\max}$ для параметров сети определяются их типом (см. раздел «Описание элементов»). Для входных сигналов принимается $a_{\min} = -1$, $a_{\max} = 1$. Если обучаемым объектом являются параметры, то генерация случайного массива производится путем генерации запроса RandomDirection компонента сеть. Если при выполнении функции возникла ошибка, то генерируется внутренняя ошибка 605 – ошибка при исполнении внешнего запроса, управление передается обработчику ошибок, функция возвращает значение ложь. В противном случае возвращается значение истина.

8.3.2.5 Модификация массива (Modify)

Заголовок функции:

Function Modify(Direct : PRealArray; OldStep, NewStep : Real) : Logic;

Описание аргументов

Direct – указатель на массив направления модификации сети.

OldStep – вес старого массива параметров в модифицированном.

NewStep – вес массива направления модификации в модифицированном массиве параметров.

Эта функция генерирует запрос на модификацию параметров сети (см. раздел «Провести обучение (Modify)» главы «Описание нейронных сетей»). Вызов запроса имеет вид:

Modify(Net, OldStep, NewStep, Tipe, Direct)

Аргументами запроса являются:

Net – указатель на пустую строку (используется сеть по умолчанию).

OldStep, NewStep – аргументы функции.

Tipe – значение аргумента What в запросе InstructionObject.

Direct – аргумент функции.

Аргумент функции Direct может быть пустым указателем. В этом случае для модификации используется массив градиента, хранящийся вместе с сетью. В случае возникновения ошибки в ходе модификации сети (запрос Modify возвращает значение ложь) генерируется внутренняя ошибка 605 – ошибка при исполнении внешнего запроса, управление передается обработчику ошибок, функция возвращает значение ложь. В противном случае возвращается значение истина.

8.3.2.6 Оптимизация шага (Optimize)

Заголовок функции:

Function Optimize (Direct : PRealArray; Step : Real) : Real;

Описание аргументов

Direct – указатель на массив направления модификации сети.

Step – начальный шаг в направлении Direct.

Действия, выполняемые функцией Optimize, описаны в разделе «Подбор оптимального шага» этой главы. В случае возникновения ошибки при выполнении функции она генерирует внутреннюю ошибку 605 – ошибка при исполнении внешнего запроса, передает управление обработчику ошибок, функция возвращает значение 0. В противном случае возвращается значение оценки при оптимальном

шаге. Следует отметить, что после завершения выполнения функции, параметры сети соответствуют результату выполнения функции Modify(Direct, 1, Step), где Step – значение оптимального шага.

8.3.2.7 Сохранить массив (SaveArray)

Заголовок функции:

Function SaveArray(Vec : PRealArray) : Logic;

Описание аргументов

Vec – указатель на массив.

Функция генерирует запрос nwGetData. После выполнения функции в массиве, на который указывает аргумент Vec, содержится текущий массив параметров. В случае возникновения ошибки в ходе выполнения функции генерируется внутренняя ошибка 605 – ошибка при исполнении внешнего запроса, управление передается обработчику ошибок, функция возвращает значение ложь. В противном случае возвращается значение истина.

8.3.2.8 Установить параметры (SetArray)

Заголовок функции:

Function SetArray(Vec : PRealArray) : Logic;

Описание аргументов

Vec – указатель на массив, содержащий параметры, которые необходимо установить.

Функция генерирует запрос nwSetData.После выполнения функции параметры сети совпадают с параметрами, содержащимися в массиве, на который указывает аргумент Vec. В случае возникновения ошибки в ходе выполнения функции генерируется внутренняя ошибка 605 – ошибка при исполнении внешнего запроса, управление передается обработчику ошибок, функция возвращает значение ложь. В противном случае возвращается значение истина.

8.3.2.9 Вычислить оценку (Estimate)

Заголовок функции:

Function Estimate(Handle : Integer; All : Logic) : Real;

Описание аргументов

Handle – номер сеанса задачника.

All – признак обучения по всему обучающему множеству.

Функция генерирует запрос к исполнителю на вычисление оценки. Если аргумент All содержит значение истина, то обучение производится по всему обучающему множеству, в противном случае – позадачно. В случае возникновения ошибки при выполнении функции он генерирует внутреннюю ошибку 605 – ошибка при исполнении внешнего запроса, передает управление обработчику ошибок, функция возвращает значение 0. В противном случае возвращается значение вычисленной оценки.

8.3.2.10 Вычислить градиент (CalcGradient)

Заголовок функции:

Function CalcGradient(Handle : Integer; All : Logic) : Real;

Описание аргументов

Handle – номер сеанса задачника.

All – признак обучения по всему обучающему множеству.

Функция генерирует запрос к исполнителю на вычисление градиента. Если аргумент All содержит значение истина, то обучение производится по всему обучающему множеству, в противном случае – позадачно. В случае возникновения ошибки при выполнении функции он генерирует внутреннюю ошибку 605 – ошибка при исполнении внешнего запроса, передает управление обработчику ошибок, функция возвращает значение 0. В противном случае возвращается значение вычисленной оценки.

8.3.2.11 Запустить запрос (GenerateQuest)

Заголовок функции:

Function GenerateQuest(Name : PString; Arguments : PRealArray) : Logic

Описание аргументов

Name – указатель на символьную строку, содержащую имя запроса.

Arguments – массив, содержащий адреса аргументов запроса.

Функция генерирует запрос к макрокомпоненту нейрокомпьютер на исполнение запроса, имя которого указано в аргументе Name, с аргументами, адреса которых указаны в аргументе Arguments. Действуют следующие ограничения. В строке, содержащей имя запроса должно содержаться только одно слово – имя запроса. Ведущие и хвостовые пробелы подавляются. В массиве Arguments должно

содержаться ровно столько элементов, сколько аргументов у генерируемого запроса. В массив Arguments всегда складываются адреса аргументов, даже если в запрос данный аргумент передается по значению.

8.3.3 Язык описания учителя

В отличие от таких компонентов как оценка, сеть и интерпретатор ответа, учитель не является составным объектом. Однако учитель может состоять из множества функций, вызывающих друг друга. Собственно учитель – это процедура, управляющая обучением сети. Ключевые слова, специфические для языка описания учителя приведены в табл. 3

Таблица 3.

Ключевые слова специфические для языка описания учителя

Ключевое слово	Краткое описание
1. Main	Начало главной процедуры
2. Instructor	Заголовок описания учителя
3. InstrLib	Заголовок описания библиотеки функций
4. Used	Подключение библиотек функций
5. Init	Начало блока инициации
6. InstrStep	Начало блока одного шага обучения
7. Close	Начало блока завершения обучения

8.3.3.1 Библиотеки функций учителя

Библиотеки функций учителя содержат описание функций, необходимых для работы одного или нескольких учителей. Использование библиотек позволяет избежать дублирования функций в различных учителях. Описание библиотеки функций аналогично описанию учителя, но не содержит главной процедуры.

8.3.3.2 БНФ языка описания учителя

Обозначения, принятые в данном расширении БНФ и описание ряда конструкций приведены в главе «Общий стандарт» в разделе «Описание языка описания компонент».

<Описание библиотеки> ::= <Заголовок библиотеки> <Описание глобальных переменных> <Описание функций> <Конец описания библиотеки>

<Заголовок библиотеки> ::= **InstrLib** <Имя библиотеки> [**Used** <Список имен библиотек>]

<Имя библиотеки> ::= <Идентификатор>

<Список имен библиотек> ::= <Имя используемой библиотеки> [,<Список имен библиотек>]

<Имя используемой библиотеки> ::= <Идентификатор>

<Конец описания библиотеки> ::= **End InstrLib**

<Описание учителя> ::= <Заголовок учителя> <Описание глобальных переменных> <Описание функций> <Главная процедура> <Конец описания учителя>

<Заголовок учителя> ::= **Instructor** <Имя библиотеки> [**Used** <Список имен библиотек>]

<Главная процедура> ::= **Main** <Описание статических переменных> <Описание переменных> <Блок инициации> <Блок шага обучения> <Блок завершения>

<Блок инициации> ::= **Init** <Тело функции>

<Блок шага обучения> ::= **InstrStep** <Выражение типа ***Logic***> <Тело функции>

<Блок завершения> ::= **Close** <Тело функции>

<Конец описания учителя> **End Instructor**

8.3.3.3 Описание языка описания учителя

Язык описания учителя является наиболее простым из всех языков описания компонент. Фактически все синтаксические конструкции этого языка описаны в главе «Общий стандарт». В теле функции, являющемся частью главной процедуры недопустим оператор возврата значения, поскольку главная процедура не является функцией. Три раздела главной функции – блок инициации, блок одного шага обучения и блок завершения являются фрагментами одной процедуры. Выделение этих разделов необходимо для выполнения запроса «Выполнить N шагов обучения». Выполнение главной процедуры происходит следующим образом. Выполняется блок инициации. Выполнение блока одного шага обучения сети производится до тех пор, пока не наступит одно из следующих событий:

1. программа выйдет из блока одного шага обучения сети прямым переходом на метку в другом разделе;
2. нарушится условие, указанное в конструкции InstStep;
3. компонент учитель получит запрос «Прервать обучение сети»;
4. в случае выполнения запроса «Выполнить N шагов обучения» блок одного шага обучения сети выполнен N раз.

   Далее выполняется блок завершения обучения.

8.3.3.4 Пример описания учителя

В данном разделе приведены описания некоторых методов обучения, описанных в разделе «Описание алгоритмов обучения».

Пример 1.

```
Instructor RandomFire;{Метод случайной стрельбы с уменьшением радиуса}
Main                    {Обучение ведется по всему обучающему множеству}
  Label Exit, Exit1;
  Static
    Integer Try Name "Число попыток при одном радиусе" Default 5;
    Real MinRadius Name "Минимальный радиус, при котором продолжается работа" Default 0.001;
    String NetName Name "Имя сети" Default "";
    Integer What Name "Что обучать" Default Parameters;
    Color InstColor Name "Цвет примеров обучающего множества" Default HFFFF; {По умолчанию}
    Integer OperColor Name "Операция для отбора цветов" Default CIn; {все примеры, в цвете ко-}
  Var                                                    {торых есть хоть один единичный бит}
    PRealArray Map, DirectMap; {Для хранения текущего и случайного массивов параметров}
    Real Est1, Est2;           {Для хранения текущей и случайной оценки}
    Real Radius;               {Текущий радиус}
    Integer TryNum, RadiusNum; {Число попыток, номер использованного радиуса}
    Integer Handle;            {Номер сеанса задачника}
    String QName;              {Имя запроса}
Init
  Begin
    If Not SetInstructionObject (What, @NetName) Then GoTo Exit;    {Задаем объекты обучения}
    QName = "InitSession";                                          {Задаем имя запроса}
    Map = NewArray(mRealArray, 3);                                  {Создаем массив для аргументов запроса}
    If Map = Null Then GoTo Exit;
    TPointer(Map^[1]) = @InstColor;                                 {Заносим адрес первого аргумента}
    TPointer(Map^[2]) = @OperColor;                                 {Заносим адрес второго аргумента}
    TPointer(Map^[3]) = @Handle;                                    {Заносим адрес третьего аргумента}
    If Not GenerateQuest(@QName, Map) Then GoTo Exit;{Открываем сеанс работы с задачником}
    If Not FreeArray(mRealArray, Map) Then GoTo Exit;  {Освобождаем массив для аргументов}
{Собственно начало обучения}
    Map = CreateArray;                                              {Создаем вспомогательные массивы}
    DirectMap= CreateArray;
    If Map = Null Then GoTo Exit;
    If DirectMap= Null Then GoTo Exit;
    Est1 = Estimate(Handle, True);
    If Error <> 0 Then GoTo Exit;
    RadiusNum = 1;                                                  {Обрабатываем первый радиус}
    Radius = 1 / RadiusNum;                                         {Вычисляем первый радиус}
    If Not SaveArray(Map) Then GoTo Exit;                           {Сохраняем начальный массив параметров}
  End
InstrStep Radius > MinRadius                                        {Обработка с одним радиусом – один шаг обучения}
  Begin
    TryNum = 0;
    While TryNum < Try Do Begin
      If Not SetArray(Map) Then GoTo Exit;                          {Устанавливаем лучший массив параметров}
      If Not RandomArray(DirectMap) Then GoTo Exit;  {Генерируется новый массив параметров}
      If Not Modify(DirectMap, 1, Radius) Then GoTo Exit; {Модифицируем массив параметров}
      Est2 = Estimate(Handle, True);
      If Error <> 0 Then GoTo Exit;
      If Est1>Est2 Then Begin
        If Not SaveArray(Map) Then GoTo Exit; {Сохраняем лучший массив параметров}
        Est1 = Est2;
        TryNum = 0;
      End Else TryNum = TryNum + 1;                                 {Увеличиваем счетчик отказов}
    End
```

```
        RadiusNum = RadiusNum + 1;                    {Обрабатываем следующий радиус}
        Radius = 1 / RadiusNum;                       {Вычисляем следующий радиус}
    End
Close
    Begin
Exit:
        If Not SetArray(Map) Then;                {Восстанавливаем лучший массив параметров}
        If Not EraseArray(Map1) Then;             {Освобождаем вспомогательные массивы}
        If Not EraseArray(Map2) Then;
        QName = "CloseSession";                                  {Задаем имя запроса}
        Map = NewArray(mRealArray, 1);                {Создаем массив для аргументов запроса}
        If Map = Null Then GoTo Exit1;
        TPointer(Map^[1]) = @Handle;                          {Заносим адрес единственного аргумента}
        If Not GenerateQuest(@QName, Map) Then;{Открываем сеанс работы с задачником}
        If Not FreeArray(mRealArray, Map) Then;   {Освобождаем массив для аргументов}
Exit1:
    End
End Instructor
```

Пример 2. Библиотека функций

```
InstrLib Library1;                            {Библиотека содержит функции для следующего учителя}
    Function SDM( Handle : Integer; Step : Real) : Real;          {Метод наискорейшего спуска}
    Label Exit, Endd;
    Var
        Real Est;
    Begin
        Est = CalcGradient(Handle, True);
        If Error <> 0 Then GoTo Exit;
        Est = Optimize(Null, Step);               {Вызываем функцию подбора оптимального шага}
        If Error <> 0 Then GoTo Exit;
        SDM = Est;
        GoTo Endd;
Exit:
        SDM = 0;
Endd:
    End

    Function RDM( Handle : Integer; Step : Real) : Real;          {Метод случайного поиска}
    Label Exit, Endd;
    Var
        Real Est;
        PRealArray : Direction;
    Begin
        Direction = CreateArray;                              {Создаем вспомогательный массив}
        If Direction = Null Then GoTo Exit;
        If Not RandomArray(Direction) Then GoTo Exit;   {Генерируется новый массив параметров}
        If Error <> 0 Then GoTo Exit;
        Est = Optimize(Direction, Step);          {Вызываем функцию подбора оптимального шага}
        If Error <> 0 Then GoTo Exit;
        RDM = Est;
        GoTo Endd;
Exit:
        RDM = 0;
Endd:
    End
End InstrLib
```

Пример 3. Антиовражная процедура обучения.

```
Instructor kParTan Used Library1;       { Антиовражная процедура обучения kParTan }
Main                                    {Обучение ведется по всему обучающему множеству}
    Label Exit, Exit1;
```

```
    Static
        Color InstColor Name "Цвет примеров обучающего множества" Default HFFFF; {По умолчанию}
        Integer OperColor Name "Операция для отбора цветов" Default CIn; {все примеры, в цвете ко-}
        String NetName Name "Имя сети" Default ""; {торых есть хоть один единичный бит}
        Integer What Name "Что обучать" Default Parameters;
        Integer k Name "Число шагов между ParTan шагами" Default 2;  {По умолчанию kParTan}
        Real Accuracy Name "Требуемый минимум оценки" Default 0.00001;
        Logic Direction Name "Случайное направление или антиградиент" Default True; {Если истина,}
    Var                                                                {то антиградиент}
        Integer Handle;              {Номер сеанса задачника}
        String QName;                {Имя запроса}
        PRealArray Map1, DirectMap;  {Для текущего массива параметров и ParTan направления}
        Real Step, ParTanStep;       {Длины шагов для оптимизации шага}
        Real Est1, Est2;             {Для хранения текущей и случайной оценки}
        Long I;
Init
    Begin
        If Not SetInstructionObject (What, @NetName) Then GoTo Exit;   {Задаем объекты обучения}
        QName = "InitSession";                                  {Задаем имя запроса}
        Map1 = NewArray(mRealArray, 3);                         {Создаем массив для аргументов запроса}
        If Map = Null Then GoTo Exit;
        TPointer(Map^[1]) = @InstColor;                         {Заносим адрес первого аргумента}
        TPointer(Map^[2]) = @OperColor;                         {Заносим адрес второго аргумента}
        TPointer(Map^[3]) = @Handle;                            {Заносим адрес третьего аргумента}
        If Not GenerateQuMap(@QName, Map) Then GoTo Exit;{Открываем сеанс работы с задачником}
        If Not FreeArray(mRealArray, Map) Then GoTo Exit;       {Освобождаем массив для аргументов}
{Собственно начало обучения}
        Map = CreateArray;                                  {Создаем вспомогательные массивы}
        DirectMap= CreateArray;
        If Map = Null Then GoTo Exit;
        If DirectMap = Null Then GoTo Exit;
        Est1 = Accuracy*10;          {Задаем оценку, не удовлетворяющую требованию точности}
        Step = 0.005;                    {Задаем начальное значение шагу}
    End
InstrStep Est > Accuracy
    Begin
        If Not SaveArray(Map1) Then GoTo Exit;          {Сохраняем начальный массив параметров}
        For I = 1 To k Do Begin                         {Выполняем k межпартанных шагов}
            If Direct Then Est = SDM(Handle, Step) Else Est = RDM(Handle, Step);
            If Error <> 0 Then GoTo Exit;
        End;
        If Not SaveArray(DirectMap) Then GoTo Exit;     {Сохраняем конечный массив параметров}
        For I = 1 To TLong(Map^[0]) Do
            DirectMap^[I] = DirectMap^[I] - Map^[I];    {Вычисляем направление ParTan шага}
        ParTanStep = 1;                                 {Задаем начальное значение ParTan шагу}
        Est = Optimize(DirectMap, ParTanStep);      {Вызываем функцию подбора оптимального шага}
        If Error <> 0 Then GoTo Exit;
    End
Close
    Begin
Exit:
        If Not EraseArray(Map) Then;        {Освобождаем вспомогательные массивы}
        If Not EraseArray(DirectMap) Then;
        QName = "CloseSession";                                 {Задаем имя запроса}
        Map = NewArray(mRealArray, 1);                      {Создаем массив для аргументов запроса}
        If Map = Null Then GoTo Exit1;
        TPointer(Map^[1]) = @Handle;                        {Заносим адрес единственного аргумента}
        If Not GenerateQuest(@QName, Map) Then;{Открываем сеанс работы с задачником}
        If Not FreeArray(mRealArray, Map) Then;  {Освобождаем массив для аргументов}
```

```
Exit1:
    End
End Instructor
```

8.4 Стандарт второго уровня компонента учитель

Компонент учитель одновременно работает только с одним учителем. Запросы к компоненту учитель можно разбить на следующие группы.

1. Обучение сети.
2. Чтение/запись учителя.
3. Инициация редактора учителя.
4. Работа с параметрами учителя.

8.4.1 Обучение сети

К данной группе относятся три запроса – обучить сеть (InstructNet), провести N шагов обучения (NInstructSteps) и прервать обучение (CloseInstruction).

8.4.1.1 Обучить сеть (InstructNet)

Описание запроса:
Pascal:

```
Function InstructNet : Logic;
```

C:

```
Logic InstructNet()
```

Аргументов нет.

Назначение – производит обучение сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Выполняется главная процедура загруженного учителя.
4. Если во время выполнения запроса возникает ошибка, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 605 – ошибка исполнения учителя, управление передается обработчику ошибок, а обработка запроса прекращается.
5. Если во время выполнения запроса возникает ошибка, а значение переменной Error не равно нулю, то обработка запроса прекращается.

8.4.1.2 Провести N шагов обучения (NInstructSteps)

Описание запроса:
Pascal:

```
Function NInstructNet( N : Integer ) : Logic;
```

C:

```
Logic NInstructNet(Integer N)
```

Описание аргумента:
N – число выполнений блока одного шага обучения сети.

Назначение – производит обучение сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Выполняется блок инициации главной процедуры загруженного учителя, N раз выполняется блок одного шага обучения, выполняется блок завершения обучения.
4. Если во время выполнения запроса возникает ошибка, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 605 – ошибка исполнения учителя, управление передается обработчику ошибок, а обработка запроса прекращается.
5. Если во время выполнения запроса возникает ошибка, а значение переменной Error не равно нулю, то обработка запроса прекращается.

8.4.1.3 Прервать обучение (CloseInstruction)

Описание запроса:
Pascal:

Function CloseInstruction: Logic;

C:

Logic CloseInstruction()

Аргументов нет.

Назначение – прерывает обучение сети.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Если в момент получения запроса не выполняется ни один из запросов обучить сеть (InstructNet) или провести N шагов обучения (NInstructSteps), то возникает ошибка 606 – неверное использование запроса на прерывание обучения, управление передается обработчику ошибок, а обработка запроса прекращается.
4. Завершается выполнение текущего шага обучения сети.
5. Выполняется блок завершения обучения сети.
6. Если во время выполнения запроса возникает ошибка, а значение переменной Error равно нулю, то генерируется внутренняя ошибка 605 – ошибка исполнения учителя, управление передается обработчику ошибок, а обработка запроса прекращается.
7. Если во время выполнения запроса возникает ошибка, а значение переменной Error не равно нулю, то обработка запроса прекращается.

8.4.2 Чтение/запись учителя

В данном разделе описаны запросы, позволяющие загрузить учителя с диска или из памяти, выгрузить учителя и сохранить текущего учителя на диске или в памяти.

8.4.2.1 Прочитать учителя (inAdd)

Описание запроса:
Pascal:

Function inAdd(CompName : PString) : Logic;

C:

Logic inAdd(PString CompName)

Описание аргумента:
CompName – указатель на строку символов, содержащую имя файла компонента или адрес описания компонента.

Назначение – читает учителя с диска или из памяти.

Описание исполнения.

1. Если в качестве аргумента CompName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя компонента и после пробела имя файла, содержащего компоненту. В противном случае считается, что аргумент CompName содержит указатель на область памяти, содержащую описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то допускается включение в текст описания компонента ключевого слова Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.
2. Если в данный момент загружен другой учитель, то выполняется запрос inDelete. Учитель считывается из файла или из памяти.
3. Если считывание завершается по ошибке, то возникает ошибка 602 – ошибка считывания учителя, управление передается обработчику ошибок, а обработка запроса прекращается.

8.4.2.2 Удаление учителя (inDelete)

Описание запроса:
Pascal:

Function inDelete : Logic;

C:

Logic inDelete()

Аргументов нет.

Назначение – удаляет загруженного в память учителя.
Описание исполнения.

1. Если список в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя учителя, управление передается обработчику ошибок, а обработка запроса прекращается.

8.4.2.3 Запись компонента (inWrite)

Описание запроса:
Pascal:

Function inWrite(Var FileName : PString) : Logic;

C:

Logic inWrite(PString* FileName)

Описание аргументов:
CompName – указатель на строку символов, содержащую имя компонента.
FileName – имя файла или адрес памяти, куда надо записать компонент.

Назначение – сохраняет учителя в файле или в памяти.
Описание исполнения.

1. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
2. Если в качестве аргумента FileName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя файла для записи компонента. В противном случае FileName должен содержать пустой указатель. В этом случае запрос вернет в нем указатель на область памяти, куда будет помещено описание компонента в формате для записи на диск. Если описание не вмещается в одну область памяти, то в текст будет включено ключевое слово Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.
3. Если во время сохранения компонента возникнет ошибка, то возникает ошибка 603 – ошибка сохранения компонента, управление передается обработчику ошибок, а обработка запроса прекращается.

8.4.3 Инициация редактора учителя

К этой группе запросов относится запрос, который инициирует работу не рассматриваемого в данной работе компонента – редактора учителя.

8.4.3.1 Редактировать компонент (inEdit)

Описание запроса:
Pascal:

Procedure inEdit(CompName : PString);

C:

void inEdit(PString CompName)

Описание аргумента:

CompName – указатель на строку символов – имя файла или адрес памяти, содержащие описание учителя.

Если в качестве аргумента CompName дана строка, первые четыре символа которой составляют слово File, то остальная часть строки содержит имя учителя и после пробела имя файла, содержащего описание учителя. В противном случае считается, что аргумент CompName содержит указатель на область памяти, содержащую описание учителя в формате для записи на диск. Если описание не вмещается в одну область памяти, то допускается включение в текст описания ключевого слова Continue, за которым следует четыре байта, содержащие адрес следующей области памяти.

Если в качестве аргумента CompName передан пустой указатель или указатель на пустую строку, то редактор создает нового учителя.

8.4.4 Работа с параметрами учителя

В данном разделе описаны запросы, позволяющие изменять параметры учителя.

8.4.4.1 Получить параметры (inGetData)

Описание запроса:
Pascal:

Function inGetData(Var Param : PRealArray) : Logic;

C:

Logic inGetData(PRealArray* Param)

Описание аргумента:
Param – адрес массива параметров.

Назначение – возвращает вектор параметров учителя.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. В массив, адрес которого передан в аргументе Param, заносятся значения параметров. Параметры заносятся в массив в порядке описания в разделе описания статических переменных.

8.4.4.2 Получить имена параметров (inGetName)

Описание запроса:
Pascal:

Function inGetName(Var Param : PRealArray) : Logic;

C:

Logic inGetName(PRealArray* Param)

Описание аргумента:
Param – адрес массива указателей на названия параметров.

Назначение – возвращает вектор указателей на названия параметров учителя.
Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. В массив, адрес которого передан в аргументе Param, заносятся адреса символьных строк, содержащих названия параметров.

8.4.4.3 Установить параметры (inSetData)

Описание запроса:
Pascal:

Function inSetData(Param : PRealArray) : Logic;

C:

Logic inSetData(PRealArray Param)

Описание аргументов:
Param – адрес массива параметров.

Назначение – заменяет значения параметров учителя на значения, переданные, в аргументе Param.

Описание исполнения.

1. Если Error <> 0, то выполнение запроса прекращается.
2. Если в момент получения запроса учитель не загружен, то возникает ошибка 601 – неверное имя компонента, управление передается обработчику ошибок, а обработка запроса прекращается.
3. Параметры, значения которых хранятся в массиве, адрес которого передан в аргументе Param, передаются учителю.

8.4.5 Обработка ошибок

В табл. 4 приведен полный список ошибок, которые могут возникать при выполнении запросов компонентом учитель, и действия стандартного обработчика ошибок.

Таблица 4.

Ошибки компонента учитель и действия стандартного обработчика ошибок.

№	Название ошибки	Стандартная обработка
601	Несовместимость сети и учителя	Занесение номера в Error
602	Ошибка считывания учителя	Занесение номера в Error
603	Ошибка сохранения учителя	Занесение номера в Error
604	Некорректная работа с памятью	Занесение номера в Error
605	Ошибка исполнения учителя	Занесение номера в Error
606	Неверное использование запроса на прерывание обучения	Занесение номера в Error