ПРОГРАММНАЯ СИСТЕМА ДЛЯ
МОДЕЛИРОВАНИЯ МНОГОСЛОЙНЫХ НЕЙРОННЫХ СЕТЕЙ
С СИГМОИДАЛЬНЫМИ ФУНКЦИЯМИ АКТИВАЦИИ

«NNSys»

(ВЕРСИЯ 0.1)

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Разработчик: Бондаренко Иван Юрьевич
email: bond005@yandex.ru
skype: i_yu_bondarenko
web: http://ua.linkedin.com/pub/ivan-bondarenko/3/785/632

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Программная система **NNSys** предназначена для моделирования процессов обучения и функционирования многослойных нейронных сетей прямого распространения сигнала с сигмоидальными функциями активации. Такие сети также называются многослойными персептронами. Этот класс нейронных сетей достаточно широко освещён в научной литературе и является одним из наиболее популярных при решении задач классификации и регресии. Обучаются эти нейронные сети, как правило, с помощью алгоритма обратного распространения ошибки и его модификаций.

Программная система рассчитана на подготовленных пользователей, имеющих теоретическое представление об искусственных нейронных сетях и алгоритмах машинного обучения. Структуры и алгоритмы нейронных сетей, положенные в основу данной программной системы, хорошо описаны в работах [1, с.46-88] и, особенно, [2, с.219-339]. Кроме того, при программировании алгоритмов обучения автор опирался также на исследование [6], в котором даны общие рекомендации по эффективной настройке алгоритма обратного распространения ошибки, и работу [10], описывающую одну из наиболее эффективных модификаций алгоритма обратного распространения ошибки.

В программной системе текущей версии (0.1) пользователь имеет возможность с помощью интерфейса командной строки создавать многослойные персептроны любой сложности, использовать их для решения задач классификации и регрессии, а также обучать многослойные персептроны с использованием следующих алгоритмов:

1) классического алгоритма стохастического обратного распространения ошибки [2, с.225-232];

2) алгоритма Incremental Delta-Bar-Delta [10];

3) алгоритма пакетного обратного распространения ошибки [2, с.238-239], в котором коэффициент скорости обучения определяется в результате решения задачи одномерной оптимизации по методу Брента [8, с.402-405];

4) алгоритма «упругого» обратного распространения, известного в англоязычной литературе под названием Resilient Back-propagation [1, с.73];

5) алгоритма сопряжённых градиентов, в котором множитель масштабирования рассчитывается по формуле Полака-Рибьера [2, с.326-327], а коэффициент скорости обучения определяется в результате решения задачи одномерной оптимизации по методу Брента [8, с.402-405].

Все эти алгоритмы относятся к классу алгоритмов обучения с учителем и семейству алгоритмов обратного распространения ошибки. Они основаны на последовательном обратном (от последнего слоя к первому) вычислении градиента функции ошибки обучаемой нейронной сети и корректировки весов этой сети в направлении антиградиента.

Первые два алгоритма являются стохастическими (их ещё называют последовательными, или онлайн-алгоритмами), поскольку вычисление градиента и корректировка весов нейронной сети происходит сразу после предъявления очередного примера из обучающего множества. При этом каждый раз перед новой эпохой обучения примеры случайно перемешиваются (этим и объясняется применение термина «стохастический» для определения данных алгоритмов).

Остальные три алгоритма являются пакетными, и корректировка весов нейронной сети в этих алгоритмах происходит только после прохода по всему обучающему множеству. Новые веса рассчитываются на основе интегрального градиента функции ошибки, получаемого путём усреднения «мгновенных» градиентов функций ошибки по всем примерам обучающего множества.

Исходя из результатов исследования [12], а также из личного опыта автора, стоит отметить, что в большинстве случаев стохастические алгоритмы обучения сходятся быстрее пакетных. Среди стохастических алгоритмов, реализованных в данной программной системе, наиболее эффективным является алгоритм Incremental Delta-Bar-Delta, хотя он имеет несколько большую вычислительную сложность в сравнении с обычным алгоритмом стохастического обратного распространения ошибки.

Программная система **NNSys** написана на языке программирования C++ с использованием библиотеки Qt 5.1.1 [9]. Распараллеливание нейросетевых вычислений реализовано с помощью технологии OpenMP [7]. Программная система является кроссплатформенной по компиляции, т.е. может быть скомпилирована без изменения программного кода в операционных системах семейств Windows, Linux и MacOS (практическое тестирование программы осуществлялось в операционных системах Microsoft Windows 7 и Linux OpenSUSE 13.1). Компиляция программы требует от пользователя начальных навыков сборки Qt-проектов.

Дальнейшие направления развития программной системы **NNSys** таковы:

1) реализация параллельных вычислений с помощью технологии OpenCL [11] вместо применяемой сейчас технологии OpenMP, что позволит эффективно использовать вычислительные ресурсы современных графических процессоров для ускорения работы больших нейронных сетей в режимах обучения и эксплуатации;

2) добавление возможности предобучения многослойной нейронной сети по алгоритму глубоких (сильно многослойных) сетей доверия (англ. Deep Belief Nets) в целях снижения вероятности «застревания» в одном из локальных минимумов функции ошибки: перед запуском алгоритма обратного распространения ошибки на этапе инициализации весов многослойный персептрон представляется как глубокая сеть доверия и обучается соответствующим алгоритмом, в результате чего веса этого персептрона принимают квазиоптимальные значения, и алгоритм обратного распространения ошибки стартует из точки, близкой к глобальному минимуму функции ошибки [4];

3) включение в состав поддерживаемых алгоритмов обучения генетического алгоритма, что также повысит вероятность достижения глобального, а не локального минимума функции ошибки [1, с.81-86].

Программная система поставляется с открытым исходным кодом и руководством пользователя по лицензии GPL версии 3 [5].

РАЗДЕЛ 1
СОЗДАНИЕ ИЛИ ИЗМЕНЕНИЕ СТРУКТУРЫ НЕЙРОННОЙ СЕТИ

1.1 Необходимые действия

Пользователь имеет возможность создавать многослойные полносвязные нейронные сети любой структуры или изменять структуру уже существующих сетей с помощью сочетания параметров командной строки **–set**, **–struct** и **–ann**, указываемых в произвольном порядке:

- **–set=i**<*число*>**–**<*число*><*тип активац. ф-ции*>**–...–**<*число*><*тип активац. ф-ции*> (все числа являются целыми положительными; может быть указан один из двух типов активационной функции — линейная **lin** или сигмоидальная **sig**);
- **–struct**;
- **–mlp=**<*строка с названием файла*>.

Значением параметра **–set** является последовательность параметров структуры нейронной сети. Элементы в этой последовательности разделены тире. Первый из элементов последовательности указывает количество входов, т.е. размер входного сигнала (отсюда буква **i** перед числом), а остальные — размеры слоёв с первого по последний соответственно и типы функций активации нейронов в этих слоях. Поскольку нейронная сеть должна состоять хотя бы из одного слоя нейронов, то вышеописанная последовательность не должна иметь меньше двух элементов (количества входов и количества нейронов хотя бы в одном слое).

Значением параметра **–mlp** является строка с названием файла, из которого должна быть прочитана нейронная сеть.

Функции активации всех нейронов сети стандартны и представляют собой либо линейные функции вида $f(s) = s$, либо же биполярные сигмоиды вида:

$$f(s)=\frac{2\cdot s}{1+|s|} \quad .$$

Если заданный файл с нейронной сетью не существует, то он будет создан. Если же этот файл существует и в нём уже есть нейронная сеть, то существующая нейронная сеть будет модифицирована в соответствии с заданными значениями аргумента **`–set`**. Модификация производится по следующим правилам:

1) для размера входного сигнала:
   - если новое число входов больше старого, то для каждого нейрона первого слоя недостающие входы добавляются в конец списка входов; веса создаваемых при этом новых связей инициализируются случайными значениями;
   - если новое число входов меньше старого, то для каждого нейрона первого слоя лишние входы удаляются из конца списка входов;
2) для количества слоёв:
   - если новое количество слоёв больше старого, то новые недостающие слои добавляются после существующих слоёв; при этом веса новых межнейронных связей инициализируются случайными значениями;
   - если новое количество слоёв меньше старого, то в качестве лишних удаляются последние слои нейронной сети;
3) для размера каждого слоя:
   - если новый размер слоя больше старого, то недостающие нейроны добавляются в конец слоя; при этом веса новых межнейронных связей инициализируются случайными значениями;
   - если новый размер слоя меньше старого, то в качестве лишних удаляются последние нейроны слоя.

1.2 Примеры

Пример 1. Создадим новую трёхслойную нейронную сеть, имеющую входной сигнал из двух элементов, три нейрона с сигмоидальными функциями

активации в первом слое, четыре нейрона с сигмоидальными функциями активации во втором слое и три нейрона с линейными функциями активации в третьем слое, и сохраним созданную сеть в файле my_mlp_1.dat (файл с таким именем ещё не существует).

Для осуществления этого действия необходимо выполнить в терминале следующую команду:

```
nnsys –set=i2–3sig–4sig–3lin –mlp=my_mlp_1.dat –struct
```

После этого в текущей директории появится файл my_mlp_1.dat, содержащий нейронную сеть заданной структуры. Веса нейронной сети будут проинициализированы случайными значениями.

Пример 2. Увеличим количество нейронов в первом слое нейронной сети из предыдущего примера с трёх до пяти.

Для осуществления этого действия необходимо выполнить в терминале следующую команду:

```
nnsys –mlp=my_mlp_1.dat –struct –set=i2–5sig–4sig–3lin
```

После этого количество нейронов первого слоя нейронной сети из файла my_mlp_1.dat будет увеличено до пяти. Веса новых связей от каждого элемента входного сигнала к 4-м и 5-м нейронами первого слоя, а также от 4-го и 5-го нейронов первого слоя к каждому нейроном- второго слоя, будут проинициализированы случайными значениями. Веса остальных связей нейронной сети изменены не будут.

РАЗДЕЛ 2
ПРОСМОТР СТУКТУРЫ НЕЙРОННОЙ СЕТИ

2.1 Необходимые действия

Пользователь имеет возможность просмотреть структуру нейронной сети с помощью сочетания параметров командной строки **`-mlp`**, **`-show`** и **`-struct`**, указываемых в произвольном порядке:

- **`-mlp=`**<*`строка с названием файла`*>;
- **`-get`**;
- **`-struct`**.

Значением параметра **`-mlp`** является строка с названием файла, из которого должна быть прочитана нейронная сеть.

2.2 Примеры

Предположим, что на диске в текущей директории существует файл my_mlp_1.dat, в котором содержится нейронная сеть с двумя входами, тремя слоями и пятью, четырьмя и тремя нейронами в слоях соответственно (см. подраздел 1.2, в котором описывалось создание такой сети). Для просмотра информации о структуре нейронной сети из данного файла необходимо выполнить следующую команду:

```
nnsys -mlp=my_mlp_1.dat -get -struct
```

РАЗДЕЛ 3
ПРОСМОТР ВЕСОВ НЕЙРОННОЙ СЕТИ

3.1 Необходимые действия

Пользователь может просмотреть значение весового коэффициента заданной межнейронной связи или группы связей с помощью сочетания параметров командной строки **–mlp**, **–get** и **–w**, указываемых в произвольном порядке:

- **–mlp=**<*строка с названием файла*>;
- **–get**;
- **–w**, либо **–w=**<*целое число*>, либо **–w=**<*целое число*>**–**<*целое число*>, либо **–w=**<*целое число*>**–**<*целое число*>**–**<*целое число*>.

Значением параметра **–mlp** является строка с названием файла, из которого должна быть прочитана нейронная сеть.

Параметр **–get** показывает, что пользователю необходимо получить текущее значение весового коэффициента заданной межнейронной связи или группы связей.

Параметр **–w** указывает, информация о какой межнейронной связи или группе связей должна быть получена:

1) если для параметра **–w** не задано значение, т.е. он указан просто как **–w**, то на экран будет выведена информация обо всех весах и смещениях нейронной сети;

2) если значение параметра **–w** указано как одно целое число, т.е. **–w=**<*целое число*>, то это число — номер слоя, информация о весам и смещениям всех нейронов которого будет выведена на экран;

3) если значение параметра **–w** указано как последовательность двух целых чисел, разделённых тире, т.е. **–w=**<*целое число*>**–**<*целое число*>, то первое из этих чисел — это номер слоя, а второе — номер нейрона в слое, информация о всем весах и смещении которого будет выведена на экран;

4) если значение параметра **–w** указано как последовательность трёх целых чисел, разделённых тире, т.е. **–w=**<*целое число*>**–**<*целое число*>**–**<*целое число*>, то первое из этих чисел — это номер слоя, второе — номер нейрона в слое, а третье — номер входа в этот нейрон; на экран будет выведена информация о весовом коэффициенте именно этого входа.

Нумерация слоёв, нейронов и входов начинается с единицы. Если в качестве номера входа в нейрон указывается число, на единицу превосходящее количество входов данного нейрона, то считается, что пользователь указал, что необходимо получить информацию о смещении нейрона.

3.2 Примеры

Для того, чтобы вывести значение смещения 3-го нейрона 2-го слоя нейросети my_mlp_1.dat (см. подразд. 1.2 и 2.2), необходимо выполнить команду:

```
nnsys –get –w=2-3-6 –mlp=my_mlp_1.dat
```

Поскольку каждый нейрон 2-го слоя имеет по пять входов (столько нейронов содержится в предыдущем слое), то указанный в данном примере вес 6-го входа 3-го нейрона 2-го слоя — это и есть вес смещения.

Для просмотра 1-го веса 3-го нейрона 2-го слоя нейросети my_mlp_1.dat необходимо выполнить команду:

```
nnsys –mlp=my_mlp_1.dat –get –w=2-3-1
```

Для просмотра весов и смещений всех нейронов 2-го слоя нейросети my_mlp_1.dat необходимо выполнить команду:

```
nnsys –mlp=my_mlp_1.dat –w=2 –get
```

РАЗДЕЛ 4
РУЧНОЕ ИЗМЕНЕНИЕ ВЕСОВ НЕЙРОННОЙ СЕТИ

4.1 Необходимые действия

Пользователь может изменить значение весового коэффициента заданной межнейронной связи или значения нескольких связей с помощью сочетания параметров командной строки **–mlp**, **–set** и **–w**, указываемых в произвольном порядке:

- **–mlp=**<*строка с названием файла*>;
- **–set=**<*вещественное число*>;
- **–w**, либо **–w=**<*целое число*>, либо **–w=**<*целое число*>**–**<*целое число*>, либо **–w=**<*целое число*>**–**<*целое число*>**–**<*целое число*>.

Значением параметра **–mlp** является строка с названием файла, содержащего нейронную сеть, в которую должны быть внесены изменения.

Параметр **–set** определяет новое значение заданной межнейронной связи или группы связей.

Параметр **–w** указывает, какая межнейронная связь или группа связей меняется:

1) если для параметра **–w** не задано значение, т.е. он указан просто как **–w**, то значение параметра **–set** будет присвоено всем весам и смещениям нейронной сети;

2) если значение параметра **–w** указано как одно целое число, т.е. **–w=**<*целое число*>, то это число — номер слоя, весам и смещениям всех нейронов которого будет присвоено значение параметра **–set**;

3) если значение параметра **–w** указано как последовательность двух целых чисел, разделённых тире, т.е. **–w=**<*целое число*>**–**<*целое число*>, то первое из этих чисел — это номер слоя, а второе — номер нейрона в слое, всем весам и смещению которого будет присвоено значение параметра **–set**;

4) если значение параметра **–w** указано как последовательность трёх целых чисел, разделённых тире, т.е. **–w=**<*целое число*>**–**<*целое число*>**–**<*целое число*>, то первое из этих чисел — это номер слоя, второе — номер нейрона в слое, а третье — номер входа в этот нейрон; весовому коэффициенту именно этого входа будет присвоено значение параметра **–set**.

Нумерация слоёв, нейронов и входов начинается с единицы. Если в качестве номера входа в нейрон указывается число, на единицу превосходящее количество входов данного нейрона, то считается, что пользователь указал, что необходимо изменить смещение нейрона.

4.2 Примеры

Для того, чтобы присвоить значение 0,5 смещению 3-го нейрона 2-го слоя нейросети my_mlp_1.dat (см. подразд. 1.2 и 2.2), необходимо выполнить команду:

```
nnsys –mlp=my_mlp_1.dat –set=0,5 –w=2-3-6
```

Поскольку каждый нейрон 2-го слоя имеет по пять входов (столько нейронов содержится в предыдущем слое), то указанный в данном примере 6-й вход 3-го нейрона 2-го слоя — это и есть смещение.

Для того, чтобы присвоить значение 1,23 1-му весу 3-го нейрона 2-го слоя нейросети my_mlp_1.dat, необходимо выполнить команду:

```
nnsys –mlp=my_mlp_1.dat –w=2-3-1 –set=1.23
```

Для того, чтобы присвоить нулевое значение всем весам и смещениям нейронов 2-го слоя нейросети my_mlp_1.dat, необходимо выполнить команду:

```
nnsys –mlp=my_mlp_1.dat –set=0.0 –w=2
```

РАЗДЕЛ 5
ИНИЦИАЛИЗАЦИЯ ВЕСОВ СЛУЧАЙНЫМИ ЗНАЧЕНИЯМИ

Пользователь может инициализировать весовые коэффициенты нейронной сети случайными значениями. Для каждого нейрона эти случайные значения выбираются из множества равномерно распределённых чисел с математическим ожиданием $M_w = 0$, и среднеквадратичным отклонением, определённым следующей формулой:

$$\sigma_w = \frac{1}{\sqrt{N_{inp}+1}} \quad ,$$

где N_{inp} — это число входов в нейрон (единица добавляется к этому числу входов для того, чтобы учесть ещё один дополнительный вход — смещение)[2,с.251-252].

Для выполнения инициализации весов при вызове программы **nnsys** необходимо указать следующие параметры командной строки (порядок указания про`извольный):

- **−mlp=<***строка с именем файла***>**;
- **−init**.

Значением параметра **−mlp** является строка с названием файла, содержащего инициализируемую нейронную сеть.

РАЗДЕЛ 6
ОБУЧЕНИЕ НЕЙРОННОЙ СЕТИ

6.1 Необходимые действия

Пользователь имеет возможность обучать созданную нейронную сеть на заданном обучающем множестве с помощью сочетания следующих параметров командной строки, указываемых в произвольном порядке:

- **–mlp=**<*строка с именем файла*>;
- **–train=**<*строка с именем файла*>;
- **–control=**<*строка с именем файла*> (необязательный параметр, который указывается не всегда и только в паре с параметром **–esmooth**);
- **–goal=**<*неотрицательное вещественное число*> (необязательный параметр, который указывается в том случае, если мы хотим указать порог ошибки обучения);
- **–alg=bp_s**, **–alg=idbd**, **–alg=bp_b**, **–alg=rp** или **–alg=cg** (разные виды алгоритмов обучения);
- **–earlystop** (необязательный параметр, который указывается не всегда и только при наличии параметра **–control**);
- **–esmooth=**<*положительное целое число*> (необязательный параметр, который указывается не всегда и только в паре с параметром **–earlystop**);
- **–maxepochs=**<*положительное целое число*>;
- **–restarts=**<*положительное целое число*>;
- **–log=**<*строка с именем файла*> или просто **–log** (необязательный параметр);
- **–etr**;
- **–eg**.

Кроме того, пользователь может указать также и некоторые специальные параметры, зависящие от выбранного типа алгоритма обучения и описанные в пункте 6.1.2.

Значением параметра **`-mlp`** является строка с названием файла, содержащего обучаемую нейронную сеть.

Значением параметра **`-train`** является строка с названием файла, содержащего обучающее множество. Формат такого файла с обучающим множеством описан в пункте 6.1.1. Размер входного сигнала в примерах обучающего множества должен совпадать с размером входного сигнала обучаемой нейронной сети, а размер желаемого выходного сигнала — с размером выходного сигнала (количеством нейронов выходного слоя) обучаемой нейронной сети.

Необязательный параметр **`-control`** задаёт необходимость использования специального контрольного множества, не пересекающегося с обучающим, для вычисления ошибки обобщения.

Если значением параметра **`-control`** является вещественное число больше нуля и меньше единицы, то контрольное множество «на лету» генерируется из обучающего путём случайного переноса (не копирования, а именно переноса!) части примеров из исходного обучающего множества. В этом случае число — значение параметра **`-control`** — определяет долю примеров обучающего множества, которые будут изъяты для формирования контрольного множества.

Если же значение параметра **`-control`** не удалось распознать как вещественное число, то данное значение рассматривается как строка с названием файла, содержащего контрольное множество, заранее сформированное каким-либо способом (например, способом, описанным в разд. 9). Как и в случае с обучающим множеством, размер входного сигнала в примерах контрольного множества должен совпадать с размером входного сигнала обучаемой нейронной сети, а размер желаемого выходного сигнала — с размером выходного сигнала (количеством нейронов выходного слоя) обучаемой нейронной сети.

С помощью ключа **`-alg`** пользователь может выбрать один из пяти алгоритмов обучения нейронной сети:

1) классический алгоритм стохастического обратного распространения (**`-alg=bp_s`**) [2, с.225-232];

2) алгоритм Incremental Delta-Bar-Delta (**`-alg=idbd`**), являющийся одним из лучших градиентных алгоритмов обучения нейронных сетей за счёт использования индивидуальных адаптивных коэффициентов скорости обучения для каждого весового коэффициента обучаемой сети [10];

3) алгоритм пакетного обратного распространения (**`-alg=bp_b`**) [2, с.238-239], в котором коэффициент скорости обучения определяется в результате решения задачи одномерной оптимизации по методу Брента [8, с.402-405];

4) алгоритм «упругого» обратного распространения (**`-alg=rp`**), известный в англоязычной литературе под названием Resilient Back-propagation [1, с.73];

5) алгоритм сопряжённых градиентов (**`-alg=cg`**), в котором множитель масштабирования рассчитывается по формуле Полака-Рибьера [2, с.326-327], а коэффициент скорости обучения определяется в результате решения задачи одномерной оптимизации по методу Брента [8, с.402-405].

Первые два алгоритма работают в последовательном режиме (такой режим называют ещё стохастическим, или онлайн), т. е. корректировка весов сети проводится после подачи каждого примера из обучающего множества, а сами примеры после каждой эпохи обучения (прохода по обучающему множеству) случайным образом перемешиваются.

Остальные три алгоритма работают в пакетном режиме, т. е. сначала рассчитывается суммарный градиент функции ошибки как среднее по градиентам, полученным для каждого примера из обучающего множества, а потом один раз в эпоху обучения происходит корректировка весов обучаемой сети.

Обучение завершается, когда будет выполнено максимально допустимое количество эпох обучения, которое задаётся значением ключа **`-maxepochs`**. Кроме того, можно использовать метод раннего останова, когда обучение прекращается в точке минимума траектории ошибки обобщения E_g, а ошибка обобщения вычисляется после каждой эпохи обучения на контрольном (проверочном) множестве, не пересекающемся с обучающим множеством [2, с.291-294]. Чтобы

использовать метод раннего останова, необходимо указать ключ **`-earlystop`** без значения, при этом ключ **`-control`**, определяющий использование контрольного множества, также должен быть указан.

Траектория ошибки обобщения не всегда является гладкой и может содержать небольшие локальные подъёмы даже при сохранении общей тенденции к убыванию. Поэтому при анализе траектории в методе раннего останова есть риск прекратить обучение слишком рано. Чтобы предотвратить этот риск, можно сглаживать траекторию с помощью медианного фильтра, порядок которого задаётся значением ключа **`-esmooth`**. По понятным причинам интервал сглаживания должен быть меньше, чем максимально допустимое число эпох обучения.

Процесс обучения может быть протоколирован. Протоколирование определяется наличием ключа **`-log`**. Если данный ключ не имеет значения, то протокол выводится прямо на экран. Если же данный ключ имеет значение, то протокол будет выводиться в текстовый файл, название которого задано значением ключа.

Пользователь может указать, надо ли выводить значения ошибок обучения и обобщения в процессе протоколирования. За вывод ошибки обучения отвечает ключ **`-etr`**, а за вывод ошибки обобщения — ключ **`-eg`**.

Процесс обучения может быть несколько раз перезапущен (каждый раз — с новыми начальными значениями весовых коэффициентов обучаемой нейронной сети), а после завершения всех рестартов процесса обучения будет выбран тот вариант обученной нейронной сети, который показал минимальную ошибку обобщения (если доступно контрольное множество) или обучения (если контрольное множество не доступно, а задано лишь обучающее). Количество таких рестартов задаётся целым положительным числом — значением ключа **`-restarts`**.

6.1.1 Формирование обучающего множества

Нейросети, моделируемые в данной программе, обучаются с учителем, поэтому обучающее множество для любой из таких нейросетей представляет собой множество обучающих пар $\{(X_k, Y_k), k = 1...N\}$, где X — это множество входных сигналов, а Y — множество соответствующих им желаемых выходных сигналов нейросети. Каждый входной сигнал — это вектор размерности M, где M совпадает с количеством входов обучающей нейронной сети. Каждый выходной сигнал — это вектор размерности L, где L совпадает с количеством нейронов последнего слоя, т.е. с количеством выходов нейросети. Поэтому как X, так и Y удобно представить в виде матриц следующего вида:

$$X = \begin{pmatrix} x_{11} & x_{12} & \dots & x_{1i} & \dots & x_{1M} \\ x_{21} & x_{22} & \dots & x_{2i} & \dots & x_{2M} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_{k1} & x_{k2} & \dots & x_{ki} & \dots & x_{kM} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_{N1} & x_{N2} & \dots & x_{Ni} & \dots & x_{NM} \end{pmatrix}, \quad Y = \begin{pmatrix} y_{11} & y_{12} & \dots & y_{1j} & \dots & y_{1L} \\ y_{21} & y_{22} & \dots & y_{2j} & \dots & y_{2L} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ y_{k1} & y_{k2} & \dots & y_{kj} & \dots & y_{kL} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ y_{N1} & y_{N2} & \dots & y_{Nj} & \dots & y_{NL} \end{pmatrix}$$

Обучающее множество содержится в двоичном файле, формат которого описан на рис. 6.1. В качестве порядка байтов используется порядок от старшего к младшему (big-endian) вне зависимости от текущей аппаратной платформы.

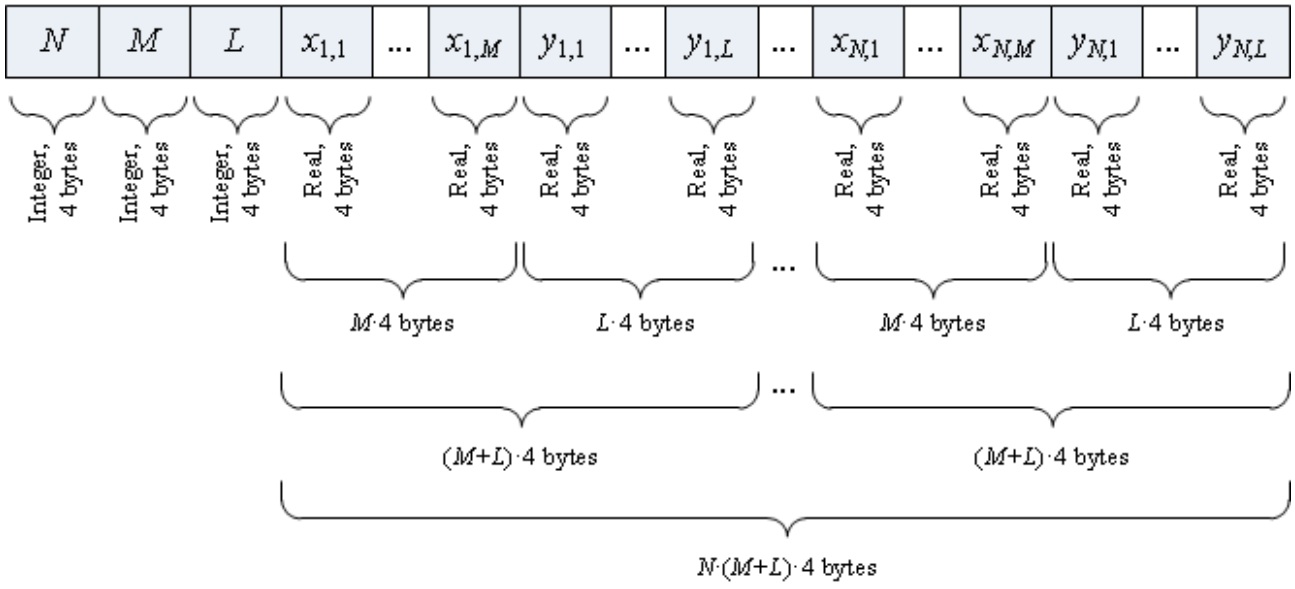

Рис. 6.1. Формат двоичного файла с обучающим множеством

6.1.2 Параметры алгоритмов обучения

6.1.2.1 Параметры алгоритма стохастического обратного распространения

Пользователь имеет возможность задать параметры алгоритма стохастического обратного распространения (**`-alg=bp_s`**) с помощью ключа **`-lr`** либо же сочетания ключей **`-lr_init`** и **`-lr_fin`**, указываемых в произвольном порядке. Значением каждого из этих ключей должно являться положительное вещественное число, определяющее коэффициент скорости обучения a. В первом случае коэффициент скорости обучения является постоянным на протяжении всех эпох обучения, а во втором случае на каждой t-й эпохе обучения пересчитывается по линейному закону $a(t) = a_1 \cdot t + a_2$. При этом коэффициенты a_1 и a_2 рассчитываются, исходя из того, что на первой эпохе коэффициент скорости обучения $a(t)$ должен быть равен значению, указанному для ключа **`-lr_init`**, а на последней — значению, указанному для ключа **`-lr_fin`**.

6.1.2.2 Параметры алгоритма Incremental Delta-Bar-Delta

Пользователь имеет возможность задать параметры алгоритма Incremental Delta Bar Delta (**`-alg=dbd`**) с помощью ключа сочетания ключей **`-lr`** и **`-theta`**, указываемых в произвольном порядке:

- **`-lr=`**<*вещественное число в диапазоне от 10^{-6} до 50*>;
- **`-theta=`**<*положительное вещественное число*>.

Идея алгоритма Incremental Delta-Bar-Delta заключается в том, что, в отличие от обычного стохастического обратного распространения, здесь для каждого весового коэффициента обучаемой нейронной сети используется индивидуальный адаптивный коэффициент скорости обучения [10]. Изначально коэффициенты скорости обучения всех весов нейронов одного слоя одинаковы и рассчитываются по формуле:

$$\alpha(i)=\alpha\cdot 1{,}2^{1-i}\cdot\frac{\sqrt{1+Inp(i)}}{\sqrt{(1+\frac{1}{N}\cdot\sum_{k=1}^{N}Inp(k))}},\quad i=\overline{1..\mathrm{N}}$$

где:

i — это номер слоя нейронной сети;

N — количество слоёв в нейронной сети;

$\alpha(i)$ — рассчитанный коэффициент скорости обучения всех весов нейронов i-го слоя;

α — исходный коэффициент скорости обучения, заданный ключом **`-lr`**;

$Inp(i)$ — размер входного сигнала для нейронов i-го слоя.

Данная формула означает, что стартовые значения коэффициентов скорости обучения, во-первых, прямо пропорциональны количеству входов $Inp(i)$ в нейрон i-го слоя, во-вторых, обратно пропорциональны номеру самого слоя i (чем дальше слой, т. е. чем больше его порядковый номер, тем сильнее занижаются коэффициенты скорости обучения весов нейронов этого слоя относительно исходного коэффициента скорости обучения α).

Затем после предъявления очередного обучающего примера перед расчётом новых значений весов и смещений нейронной сети происходит пересчёт всех коэффициентов скорости обучения. При этом изменение конкретного коэффициента скорости обучения для соответствующего веса (или смещения) нейрона зависит как от частной производной функции ошибки по данному весу (или смещению), так и от управляющего параметра θ, задающего скорость адаптации [10].

Величина управляющего параметра θ задаётся значением ключа **`-theta`**. Чем больше это значение, тем выше будет скорость адаптации индивидуальных коэффициентов скорости обучения (т. е. тем сильнее они будут изменяться на каждом шаге).

6.1.2.3 Параметры алгоритма пакетного обратного распространения

Пользователь имеет возможность задать параметры алгоритма пакетного обратного распространения (**`-alg=bp_b`**) с помощью ключа сочетания ключей **`-lr_max`**, **`-lr_iters`** и **`-eps`**, указываемых в произвольном порядке:

- **`-lr_max=`**<*положительное вещественное число*>;
- **`-lr_iters=`**<*положительное целое число*>;
- **`-eps=`**<*положительное вещественное число не больше единицы*>.

В отличие от стохастического варианта алгоритма обратного распространения (см. подпункт 6.1.2.1), в пакетном варианте этого алгоритма весовые коэффициенты нейронов сети пересчитываются не на основе «мгновенного» градиента функции ошибки, вычисленной на одном конкретном примере обучающего множества, а на основе интегрального градиента функции ошибки, полученного путём усреднения всех «мгновенных» градиентов по всему обучающему множеству. Соответственно, весовые коэффициенты в пакетном алгоритме изменяются не после предъявления каждого примера обучающего множества, а один раз в обучающую эпоху [2, с.238-239].

При этом коэффициент скорости обучения на каждой эпохе подбирается, исходя из решения задачи одномерной оптимизации: какой длины должен быть шаг изменения весов в направлении антиградиента, чтобы минимизировать ошибку обучения. Приближённое решение такой задачи может быть получено по алгоритму Брента [8, с.402-405], что обеспечивает компромисс между точностью оптимизации и скоростью вычислений. Значения ключей **`-lr_max`** и **`-lr_iters`** определяют параметры алгоритма Брента — соответственно, наиболее длинный шаг оптимизации, т. е. максимально допустимый коэффициент скорости обучения, и наибольшее количество итераций алгоритма.

Значение ключа **`-eps`** определяет дополнительный критерий останова (кроме описанных в начале подразд. 6.1) для всего алгоритма обучения в целом. Если норма интегрального градиента после очередной эпохи стала меньше, чем

исходная норма интегрального градиента, умноженная на данное число, то процесс обучения завершается.

6.1.2.4 Параметры алгоритма «упругого» обратного распространения

Пользователь имеет возможность задать параметры алгоритма «упругого» обратного распространения (**`-alg=rp`**) с помощью ключа сочетания ключей **`-lr`** и **`-eps`**, указываемых в произвольном порядке:

- **`-lr=`**<*вещественное число в диапазоне от 10^{-6} до 50*>;
- **`-eps=`**<*положительное вещественное число не больше единицы*>.

Алгоритм «упругого» обратного распространения во многом похож пакетному варианту обычного алгоритма обратного распространения, за исключением двух моментов: 1) при уточнении весов учитывается только знак соответствующих компонентов интегрального градиента, а их значения игнорируются; 2) коэффициент скорости обучения подбирается индивидуально для каждого веса с учётом изменения значения градиента. [1, с. 73] При этом начальные коэффициенты скорости обучения для всех весов нейронной сети одинаковы и равны числу, указанному как значение ключа **`-lr`**.

Значение ключа **`-eps`** определяет дополнительный критерий останова (кроме описанных в начале подразд. 6.1) для всего алгоритма обучения в целом. Если норма интегрального градиента после очередной эпохи стала меньше, чем исходная норма интегрального градиента, умноженная на данное число, то процесс обучения завершается.

6.1.2.5 Параметры алгоритма сопряжённых градиентов

Параметры алгоритма сопряжённых градиентов такие же, как и параметры алгоритма пакетного обратного распространения, описанные в 6.1.2.3. Отличием алгоритма сопряжённых градиентов является то, что направление изменения

весов обучаемой нейронной сети определяется не только на основе вектора антиградиента функции ошибки, но также и на основе множества сопряжённых векторов направлений функции ошибки. Это позволяет повысить скорость сходимости процесса обучения. [2, с. 319-327]

6.2 Пример обучения нейронной сети решению задачи классификации типа XOR

Для демонстрации использования программного эмулятора в режиме обучения нейронной сети рассмотрим простейший пример.

Предположим, мы хотим обучить нейронную сеть решению задачи типа «исключающее или», т. е. задачи двухклассовой классификации, в которой образы разных классов (назовём их «класс синих точек» и «класс красных точек») так расположены в пространстве признаков, что они приближенно соответствуют логической функции «исключающего или».

Подготовим обучающее, контрольное и тестовое множество и сохраним их в CSV-формате [3] в текстовых файлах xor_train_set.txt, xor_control_set.txt и xor_test_set.txt (см. рис.6.2-6.4). Входные сигналы в этих множествах имеют размерность 2, поскольку мы проводим классификацию образов по двум признакам: x_1 и x_2. Желаемый выходной сигнал в этих множествах имеет размерность 1 и для образов класса «синих точек» всегда равен -1, а для образов класса «красных точек» всегда равен +1.

Преобразуем подготовленные данные из формата CSV в формат обучающего множества (об этом более подробно в разд. 10) следующими командами:

```
nnsys -to_ts -csv=xor_train_set.txt -i=2 -o=1 -trainset=xor_train_set.dat
nnsys -to_ts -csv=xor_control_set.txt -i=2 -o=1 -trainset=xor_control_set.dat
nnsys -to_ts -csv=xor_test_set.txt -i=2 -o=1 -trainset=xor_test_set.dat
```

```
 0.1000, -0.0500, -1
-0.3000, -0.0010, -1
 0.1230,  0.2000, -1
 0.0590,  0.0010, -1
 0.0800,  1.0000,  1
-0.2700,  0.9400,  1
-0.0058,  1.3900,  1
-0.2600,  1.0070,  1
 1.1980,  1.2000, -1
 0.7500,  1.1000, -1
 0.9800,  0.8600, -1
 1.0900,  0.9900, -1
 0.7700,  0.0700,  1
 0.8200, -0.0030,  1
 1.0700,  0.0010,  1
 1.1800, -0.1900,  1
```

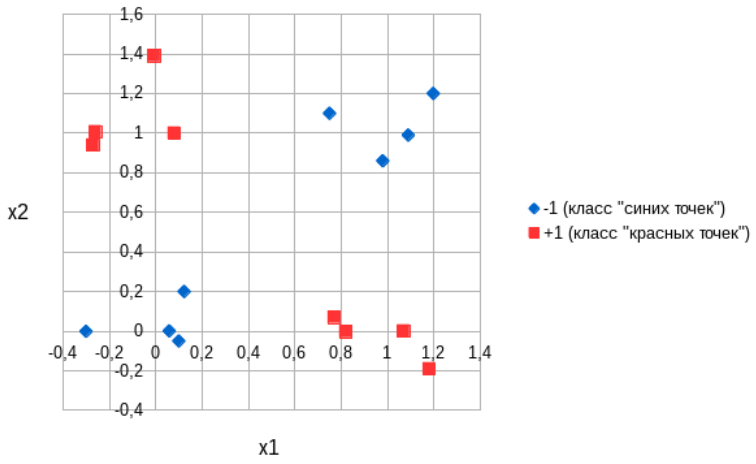

Рис. 6.2. Образы классов «синих точек» и «красных точек», включенные в CSV-файл обучающего множества xor_train_set.txt

```
-0.0010, -0.0800, -1
-0.2480,  1.0100,  1
 1.0910,  0.8370, -1
 0.8200,  0.0600,  1
```

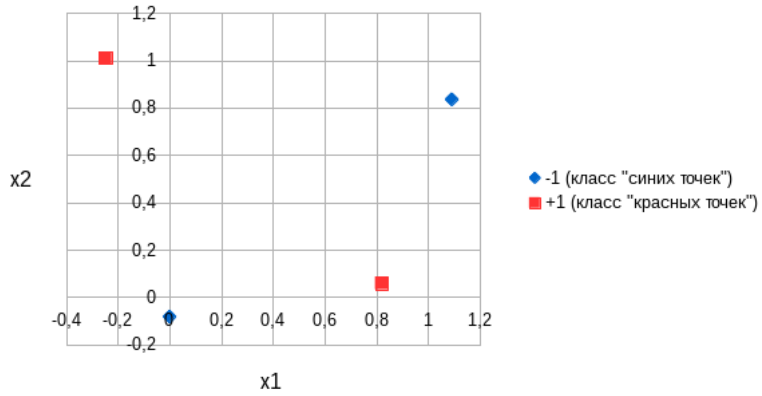

Рис. 6.3. Образы классов «синих точек» и «красных точек», включенные в CSV-файл контрольного множества xor_control_set.txt

```
 0.1200, -0.0300, -1
-0.4000, -0.0610, -1
-0.0840,  1.1000,  1
-0.1300,  1.0400,  1
 1.2580,  1.1670, -1
 0.7710,  1.0130, -1
 0.8750, -0.1800,  1
 1.0260, -0.1230,  1
```

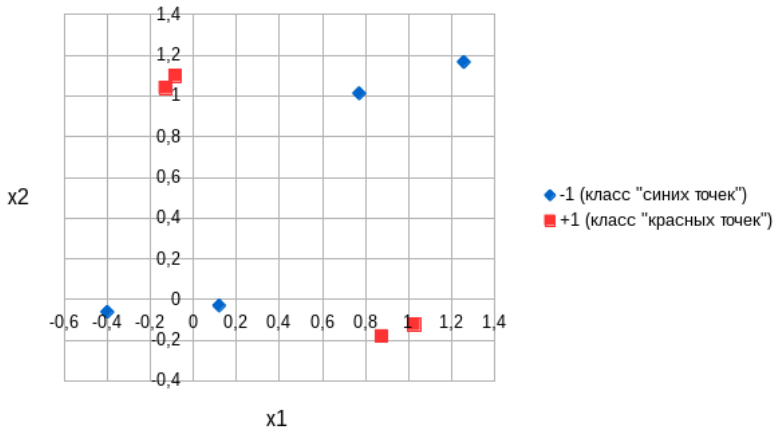

Рис. 6.4. Образы классов «синих точек» и «красных точек», включенные в CSV-файл тестового множества xor_test_set.txt

Поскольку данная задача относится к классу линейно неразделимых выпуклых задач классификации, для её решения понадобится нейронная сеть с двумя слоями и сигмоидальными функциями активации нейронов в слоях. В выходном (втором) слое будет один нейрон (поскольку размер выходного сигнала равен 1), а в скрытом (первом) слое нам достаточно двух нейронов. Создадим и инициализируем такую сеть следующими командами (см. разд.1 и разд.5):

```
nnsys -mlp=mlp_for_xor.dat -struct -set=i2-2sig-1sig
nnsys -mlp=mlp_for_xor.dat -init
```

Теперь обучим созданную нейронную сеть по алгоритму сопряжённых градиентов. Для предотвращения переобучения будем использовать критерий раннего останова. Для сглаживания траектории ошибки обобщения при вычислении критерия раннего останова будем использовать медианный фильтр 11-го порядка. Ход процесса обучения выведем в текстовый файл trainset.txt. Чтобы осуществить всё это, выполним следующую команду (команда должна быть записана в одну строку):

```
nnsys -mlp=mlp_for_xor.dat -train=xor_train_set.dat -control=xor_control_set.dat
-alg=cg -restarts=1 -maxepochs=1000 -earlystop -esmooth=11 -eps=0.001 -lr_max=2.0
-lr_iters=50 -log -etr -eg > trainlog.txt
```

Траектории ошибок обучения и обобщения показаны на рис. 6.5. Для наглядности ось ординат, по которой откладываются значения среднеквадратичной ошибки, представлена в логарифмическом масштабе.

Из рисунка видно, что, если пренебречь случайными колебаниями, ошибка обучения всё время убывает, а ошибка обобщения сначала убывает, а потом, после 34-й эпохи, начинает возрастать, и это приводит, в конце концов, к срабатыванию критерия раннего останова. В файл сохраняется версия нейронной сети, показавшая минимальную ошибку обобщения (т. е. версия, полученная на 34-й эпохе обучения).

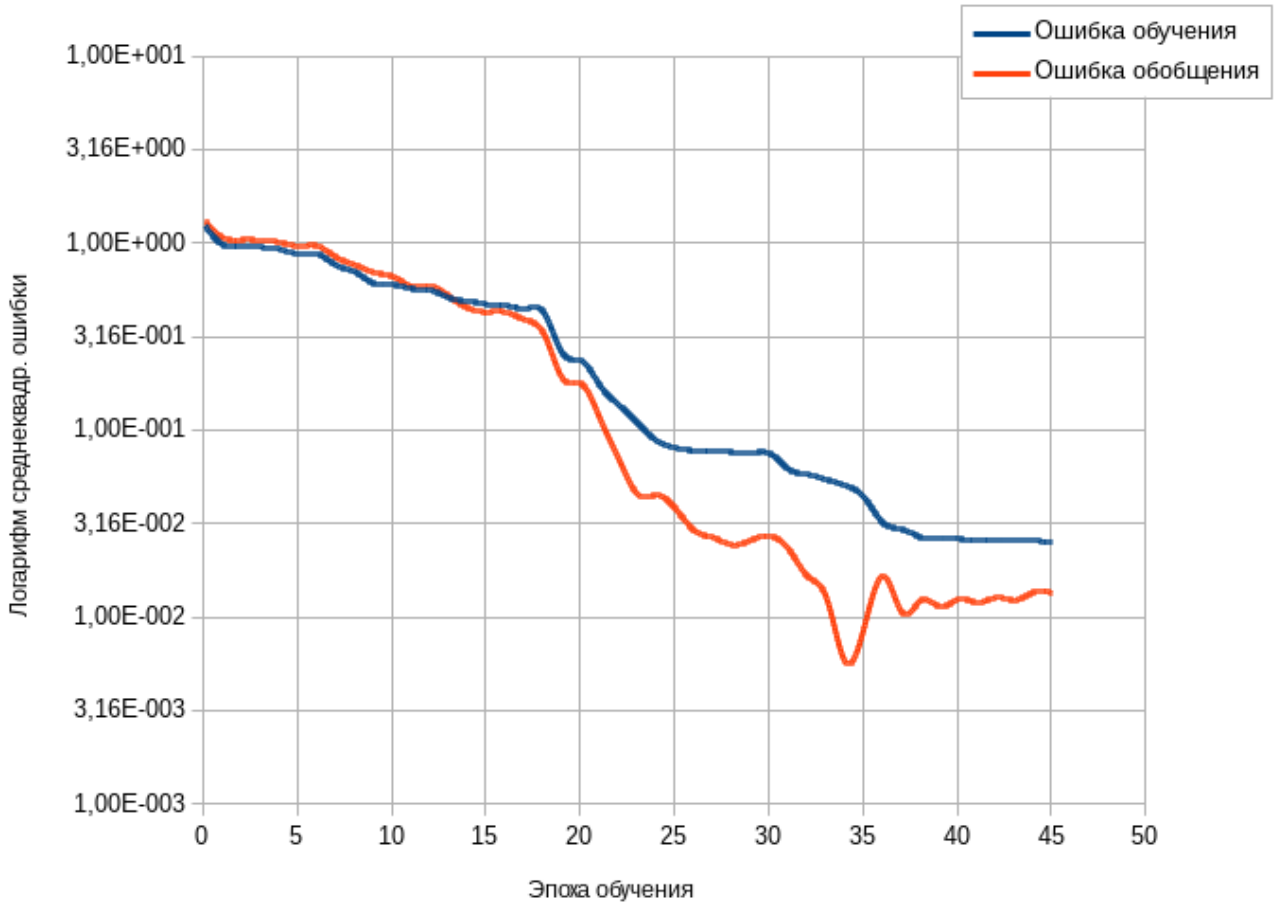

Рис. 6.5. Траектории ошибок обучения и обобщения при обучении нейронной сети решению задачи классификации типа XOR

После завершения обучения нейронную сеть можно протестировать на специальном тестовом множестве xor_test_set.dat, данные из которого не входили ни в обучающее, ни в контрольное множество. Это делается следующей командой (см. разд. 7):

```
nnsys -mlp=mlp_for_xor.dat -in=xor_test_set.dat -task=class > testlog.txt
```

В результате оказывается, что среднеквадратичная ошибка на тестовом множестве составила 0,0505856, а ошибка классификации — 0 %. Это означает, что ни один пример из тестового множества не был классифицирован неправильно (способ вычисления ошибки классификации см. в подразд. 7.2).

РАЗДЕЛ 7
ПРИМЕНЕНИЕ ОБУЧЕННОЙ НЕЙРОННОЙ СЕТИ

7.1 Необходимые действия

Пользователь может применять нейронную сеть с помощью сочетания параметров командной строки **–mlp**, **–in**, необязательных параметров **–out** и **–task**, указываемых в произвольном порядке:

- **–mlp=**<*строка с названием файла*>;
- **–in=**<*строка с названием файла*>;
- **–out=**<*строка с названием файла*>;
- **–task=class** или **–task=reg** (параметр **–task**).

Значением параметра **–mlp** является строка с названием файла, содержащего применяемую нейронную сеть.

Параметр **–in** указывает на файл входных данных, содержащий входное множество, т. е. множество входных сигналов для нейронной сети и соответствующее множество желаемых выходных сигналов (последнее — опционально). Размер входного сигнала должен совпадать с количеством входов нейронной сети, а размер желаемого выходного сигнала должен быть либо равен размеру выходного слоя нейронной сети (в таком случае программа не только рассчитает множество выходных сигналов нейронной сети, но и вычислит среднеквадратичную ошибку), либо же должен быть равен нулю.

Параметр **–out** указывает на файл результатов (выходных данных), в который будет записано множество выходных сигналов нейронной сети, вычисленных как отклики на соответствующие входные сигналы заданного входного множества. Если параметр **–out** не указывать, то множество результатов (выходных сигналов) будет всё равно вычислено, но не будет никуда сохранено.

Параметр **–task** определяет задачу, решаемую нейронной сетью (**–task=class** — решается задача классификации, **–task=reg** — решается задача

регрессии). Если этот параметр указан, то, кроме среднеквадратичной ошибки, программа рассчитает ещё и ошибку классификации или ошибку регрессии (см. подразд. 7.2). Естественно, что использование этого параметра целесообразно только в том случае, если в файле с входным множеством, определяемом параметром **`-in`**, содержатся не только входные, но и желаемые выходные сигналы. В противном случае указание данного параметра не имеет силы.

Вне зависимости от допустимых комбинаций параметров командной строки, после завершения расчёта выходных сигналов нейронной сети программа выведет оценку времени, которое было затрачено на эти расчёты.

7.2 Вычисление ошибок классификации и регрессии

Если принять за N количество примеров во входном множестве, то ошибка классификации считается следующим образом:

$$Err_{class} = \frac{\sum_{i=1}^{N} f_{err}(net_i, trg_i)}{N} \cdot 100\,\% \quad ,$$

где $f_{err}(net, trg)$ — это функция, вычисляющая ошибку классификации путём сопоставления выходного сигнала нейронной сети *net* и желаемого выходного сигнала тестового множества *trg*. Если размер выходного сигнала $M = 1$, т. е. равен единице, то программа считает, что решается задача двухклассовой классификации. Тогда функция ошибки классификации считается так:

$$f_{class}(net, trg) = \begin{cases} 1, & \text{sign}(net) = \text{sign}(trg); \\ 0, & \text{sign}(net) \neq \text{sign}(trg). \end{cases}$$

Если же размер выходного сигнала $M > 1$, т. е. больше единицы, то программа считает, что решается задача многоклассовой классификации с M классами. В таком случае функция ошибки классификации считается по-другому:

$$f_{class}(net, trg) = \begin{cases} 1, & \text{argmax}(net_1, .. net_M) = \text{argmax}(trg_1, .. trg_M); \\ 0, & \text{argmax}(net_1, .. net_M) \neq \text{argmax}(trg_1, .. trg_M). \end{cases}$$

Ошибка регрессии считается, если мы пример за M размер выходного сигнала нейронной сети, следующим образом:

$$f_{reg}(net, trg) = \sum_{j=1}^{M} \frac{net_j - trg_j}{trg_j}.$$

7.3 Форматы файла входных данных и файла результатов

Форматы файлов входных и выходных данных соответствуют формату файлов с обучающими множествами (см. пункт 6.1.1), при этом:

1) файл входных данных может содержать желаемые выходные сигналы, а может и не содержать их;

2) файл результатов всегда будет содержать только входные сигналы, поскольку выходные сигналы нейронной сети записываются в файл результатов как входные сигналы.

7.4 Примеры

Предположим, что в файле tested_mlp.dat у нас содержится нейронная сеть, а в файле test_set.dat — множество входных данных (входное множество) для этой нейронной сети, причём размер входного сигнала в этом множестве совпадает с размером входного сигнала нейронной сети, а размер желаемого выходного сигнала является ненулевым и совпадает с размером выходного сигнала нейронной сети.

В таком случае, если мы хотим вычислить выходные сигналы нейронной сети как отклики на входные сигналы заданного входного множества, а также

сохранить вычисленные выходные сигналы в файле results.dat, то мы должны выполнить следующую команду:

```
nnsys –mlp=tested_mlp.dat –in=test_set.dat –out=results.dat
```

При этом, кроме расчёта выходных сигналов нейронной сети и сохранения их в файл results.dat, программа вычислит среднеквадратичную ошибку (поскольку во входном множестве test_set.dat, кроме входных, присутствуют и желаемые выходные сигналы), а также оценит время, затраченное на работу нейронной сети.

Если же мы хотим рассчитать ошибку классификации, то нам следует выполнить следующую команду:

```
nnsys –mlp=tested_mlp.dat –in=test_set.dat –task=class
```

При этом будет вычислена и выведена на экран как среднеквадратичная ошибка, так и ошибка классификации.

В случае же, если нас интересуют только временны́е характеристики работы данной нейронной сети, достаточно выполнить команду:

```
nnsys –mlp=tested_mlp.dat –in=test_set.dat
```

На экран будет выведена среднеквадратичная ошибка нейронной сети, а также общее время, затраченное на обработку всех примеров входного множества, и среднее время, затраченное нейронной сетью на вычисление выходного сигнала для одного примера входного множества. Сами же результаты вычислений — рассчитанные выходные сигналы нейронной сети — никуда не будут сохранены. При этом, если входное множество tested_mlp.dat не содержало бы желаемых выходных сигналов, то на экран не была бы выведена даже среднеквадратичная ошибка, а только временны́е характеристики работы нейронной сети.

РАЗДЕЛ 8
ПОЛУЧЕНИЕ ИНФОРМАЦИИ ОБ ОБУЧАЮЩЕМ МНОЖЕСТВЕ

Пользователь может получить информацию о структуре обучающего множества (количество примеров, размеры входного и желаемого выходного сигналов), указав в командной строке параметр **–trainset** следующим образом:

- **–trainset=**<*строка с названием файла*>.

Значением параметра **–trainset** является строка с названием файла, содержащего проверяемое обучающее множество. Формат такого файла с обучающим множеством описан в пункте 6.1.1.

РАЗДЕЛ 9
РАЗДЕЛЕНИЕ ОБУЧАЮЩЕГО МНОЖЕСТВА НА СОБСТВЕННО ОБУЧАЮЩЕЕ И КОНТРОЛЬНОЕ

Пользователь может выполнить разделение исходного обучающего множества на собственно обучающее и контрольное с помощью сочетания параметров командной строки **–trainset**, **–controlset**, и **–r** указываемых в произвольном порядке:

- **–trainset=**<*названием файла с обучающим множеством*>;
- **–controlset=**<*названием файла с контрольным множеством*>;
- **–r=**<*вещественное положительное число меньше 0 и больше 1*>.

Значением параметра **–trainset** является строка с названием файла, содержащего исходное обучающее множество.

Значением параметра **–controlset** является строка с названием файла, содержащего контрольное множество, которое будет выделено из обучающего в результате разделения.

Параметр **–r** определяет коэффициент разделения, т. е. какую долю примеров исходного обучающего множества надо перенести в создаваемое контрольное множество, если размер исходного обучающего множества принять за единицу. Таким образом, очевидно, что значение данного параметра должно быть меньше единицы (нельзя перенести в контрольное множество все примеры из исходного обучающего множества, ведь надо же что-то оставить и в обучающем), но больше нуля (нельзя создать пустое контрольное множество — надо, чтобы туда было перенесено хотя бы немного примеров из обучающего множества).

Алгоритм разделения таков:

1) на основе коэффициента разделения определяется N_{contr} — число примеров из исходного обучающего множества, которые надо перенести в создаваемое контрольное множество;

2) из исходного обучающего множества случайным образом отбирается N_{contr} примеров, которые заносятся в создаваемое контрольное множество;

3) в соответствующих файлах сохраняются оба множества — и вновь созданное контрольное, и изменённое обучающее, из которого изъяли N_{contr} примеров.

Например, предположим, что у нас есть файл с обучающим множеством train_set.dat, а мы хотим разбить его на файлы с собственно обучающим и контрольным множествами, выделив для контрольного множества 15% примеров исходного обучающего множества и сохранив полученное контрольное множество в файле control_set.dat. Для этого мы должны выполнить следующую команду:

```
nnsys –trainset=train_set.dat –controlset=control_set.dat –r=0.15
```

В результате выполнения этой команды в файле control_set.dat будет создано новое контрольное множество, в котором все примеры имеют те же размеры входного и желаемого выходного сигналов, как и в обучающем множестве train_set.dat. Исходное обучающее множество так же будет модифицировано — количество примеров в нём уменьшится на столько, сколько примеров было добавлено в сформированное контрольное множество.

РАЗДЕЛ 10
ПРЕОБРАЗОВАНИЕ ДАННЫХ ИЗ ФОРМАТА CSV В ФОРМАТ ОБУЧАЮЩЕГО МНОЖЕСТВА

10.1 Необходимые действия

Пользователь может выполнить преобразование данных из формата CSV [3] в формат обучающего множества с помощью сочетания параметров командной строки **–trainset**, **–csv**, **–i**, **–o** и **–to_ts** указываемых в произвольном порядке:

- **–trainset=**<*названием файла с обучающим множеством*>;
- **–csv=**<*названием текстового файла с данными в CSV-формате*>;
- **–i=**<*целое положительное число*>;
- **–o=**<*целое неотрицательное число*>;
- **–to_ts**.

Значением параметра **–csv** является строка с названием текстового файла, содержащего данные в формате CSV. Требования к этому файлу следующие:

1) каждая строка CSV-файла представляет собой текст, описывающий последовательность чисел, разделённых запятыми;

2) каждое число является произвольным, целым или вещественным (в последнем случае десятичным разделителем является точка независимо от текущих настроек локализации);

3) количество чисел во всех строках CSV-файла должно быть одинаковым.

Значением параметра **–trainset** является строка с названием файла, в который будет записано обучающее множество, созданное на основе данных заданного CSV-файла. Формат создаваемого обучающего множества соответствует описанному в пункте 6.1.1.

Значениями параметров **–i** и **–o** являются, соответственно, размеры входного и желаемого выходного сигналов создаваемого обучающего множества. Сумма этих размеров должна совпадать с количеством чисел в строке исходного

CSV-файла, при этом размер входного сигнала должен быть строго положительной величиной, а размер желаемого выходного сигнала может быть и нулевым (в таком случае в обучающем множестве будут присутствовать только входные сигналы, и оно становится пригодным лишь для обучения без учителя).

Параметр **–to_ts** не имеет значения и указывает программе на направление преобразования данных (из CSV-формата в формат обучающего множества).

10.2 Примеры

Пусть у нас в некотором текстовом файле csv_data.txt есть множество данных в CSV-формате. Это множество данных приведено на рис. 10.1.

```
0.25, -0.3, 1
0.01, 0.2, 1
-0.29, 1.1, 0
0, 0.89, 0
0.7, 0.99, 1
1.24, 0.96, 1
1.02, 0.32, 0
1.11, -0.28, 0
```

Рис. 10.1. Множество данных в CSV-формате, которое будет преобразовано в заданное обучающее множество

Мы хотим преобразовать эти данные в обучающее множество, при этом первые два числа каждой строки CSV-файла должны образовать входной сигнал соответствующего примера, а третье, последнее, число — желаемый выходной сигнал. Сформированное обучающее множество мы хотим сохранить в файле train_set.dat. Для выполнения всех этих действий мы должны выполнить команду:

```
nnsys -to_ts -csv=csv_data.txt –trainset=train_set.dat -i=2 -o=1
```

РАЗДЕЛ 11
ПРЕОБРАЗОВАНИЕ ДАННЫХ ИЗ ФОРМАТА ОБУЧАЮЩЕГО МНОЖЕСТВА В ФОРМАТ CSV

11.1 Необходимые действия

Пользователь может выполнить преобразование данных из формата обучающего множества в формат CSV [3] с помощью сочетания параметров командной строки **`-trainset`**, **`-csv`** и **`-to_csv`** указываемых в произвольном порядке:

- **`-trainset=`**<*названием файла с обучающим множеством*>;
- **`-csv=`**<*названием текстового файла с данными в CSV-формате*>;
- **`-to_csv`**.

Значением параметра **`-trainset`** является строка с названием файла с обучающим множеством (его формат см. в пункте 6.1.1), данные из которого будут преобразованы в CSV-формат.

Значением параметра **`-csv`** является строка с названием текстового файла, в который будут записаны данные в формате CSV. В каждой строке этого файла будет текст, описывающий последовательность чисел — компонентов входного и желаемого выходного сигналов соответствующего примера в исходном обучающем множестве. В качестве десятичного разделителя в этих числах будет использована точка. Сами числа в строке будут отделены друг от друга запятыми.

Параметр **`-to_csv`** не имеет значения и указывает программе на направление преобразования данных (из формата обучающего множества в формат CSV).

11.2 Примеры

Пусть у нас есть некоторое обучающее множество в файле train_set.dat, полученное так, как это описано в примере из подразд. 10.2. Мы хотим

преобразовать данные этого обучающего множества в формат CSV и сохранить в текстовом файле another_csv_data.txt. Для выполнения всех этих действий мы должны выполнить команду:

```
nnsys -to_csv –trainset=train_set.dat -csv=another_csv_data.txt
```

В результате выполнения команды будет создан текстовый файл another_csv_data.txt, данные в котором будут соответствовать данным из CSV-файла csv_data.txt, описанного в подразд. 10.2. Открыв файл another_csv_data.txt в любом текстовом редакторе, мы увидим набор текстовых данных, аналогичный тому, что мы видим на рис. 10.1 из предыдущего раздела.

ПЕРЕЧЕНЬ ССЫЛОК

1. Осовский С. Нейронные сети для обработки информации: Пер. с польск. — М.: Финансы и статистика, 2002. — 344 с.

2. Хайкин С. Нейронные сети: полный курс, 2-е изд., испр.: Пер. с англ. — М.: ООО «И.Д.Вильямс», 2006. — 1104 с.

3. CSV File Format Specification // http://mastpoint.curzonnassau.com/csv-1203 [Проверено 10.02.2014]

4. Erhan D.,Bengio Y., Courville A., Manzagol P. A., Vincent P. and Bengio S. Why does unsupervised pre-training help deep learning? // Journal of Machine Learning Research. — Vol. 11. — 2010. — P. 625-660.

5. GNU General Public License // http://www.gnu.org/licenses/gpl.html [Проверено 10.02.2014]

6. LeCun Y., Bottou L., Orr G. and Muller K. Efficient Backprop // Neural Networks: Tricks of the Trade. — New York: Springer, 1998. — P.5-50.

7. OpenMP Architecture Review Board (официальный сайт) // http://www.openmp.org [Проверено 10.02.2014]

8. Press W.H., Teukolsky S.A., Vetterling W.T., Flannery B.P. Numerical Recipes: The Art of Scientific Computing (2rd ed.). — New York: Cambridge University Press, 1992. — 1018 p.

9. Qt Project (официальный сайт проекта) // http://qt-project.org [Проверено 10.02.2014]

10. Sutton R.S. Adapting Bias by Gradient Descent: An Incremental Version of Delta-Bar-Delta // Proceedings of the 10th National Conf. On Artificial Intelligence, San Jose, CA, July 1992. — P.171-176.

11. The open standard for parallel programming of heterogeneous systems (официальная страница стандарта OpenCL) // http://www.khronos.org/opencl [Проверено 10.02.2014]

12. Wilson D.R., Martinez T.R. The inefficiency of batch training for large training sets // Proc. Int. Joint Conf. Neural Networks (IJCNN'2000), Como, Italy, 2000. Vol.2. — P.113-117.