ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ

Федеральное государственное
бюджетное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
им. проф. М. А. БОНЧ-БРУЕВИЧА»

Ф. В. Филиппов

МОДЕЛИРОВАНИЕ НЕЙРОННЫХ СЕТЕЙ НА R

УЧЕБНОЕ ПОСОБИЕ

СПб ГУТ)))

САНКТ-ПЕТЕРБУРГ
2016

УДК 004.42(075.8)
ББК 32.973-018.2я73
Ф53

Рецензенты:
кандидат технических наук, доцент кафедры робототехники
и автоматизации производственных систем СПбГЭТУ «ЛЭТИ»
А. В. Шевченко,
кандидат технических наук, доцент кафедры конструирования
и производства радиоэлектронных средств
Т. В. Матюхина

Утверждено редакционно-издательским советом СПбГУТ
в качестве учебного пособия

Филиппов, Ф. В.
Ф53 Моделирование нейронных сетей на R : учебное пособие / Ф. В. Филиппов ; СПбГУТ. – СПб., 2016. – 84 с.

Рассматриваются практические аспекты моделирования нейронных сетей на базе широкого спектра методов, предоставляемых открытыми библиотеками языка R.

Предназначено для студентов, обучающихся по направлению подготовки 09.03.02 «Информационные системы и технологии», и будет полезно при изучении дисциплин «Технологии обработки информации» и «Интеллектуализация управления инфокоммуникационными системами и сетями».

УДК 004.42(075.8)
ББК 32.973-018.2я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель настоящего пособия состоит в том, чтобы познакомить студентов с основами использования современных сред обработки информации. Язык R используется в учебном процессе в подавляющем большинстве университетов мира, готовящих ИТ-специалистов, а также в научных кругах для проведения исследований и разработки бизнес-приложений.

Язык R и среда RStudio являются свободно распространяемым программным обеспечением и широко используются ИТ-специалистами, занимающимися всесторонним анализом данных, в частности исследованием систем искусственного интеллекта.

Основная задача пособия заключается в развитии у студентов практических навыков использования эффективных программных пакетов моделирования нейронных систем.

Пособие включает необходимый теоретический материал для подготовки к практическим занятиям, а также примеры использования программных пакетов для закрепления изученного материала. Для успешного освоения материала пособия рекомендуется по мере знакомства с возможностями описываемых программных пакетов тщательно изучать примеры, запускать их на исполнение в среде RStudio и анализировать результаты их выполнения.

1. НЕЙРОННЫЕ СЕТИ

Интеллектуальные системы на основе искусственных нейронных сетей позволяют с успехом решать проблемы распознавания образов, выполнения прогнозов, оптимизации, организации ассоциативной памяти и управления. Известны и иные, более традиционные подходы к решению этих проблем, однако они не обладают необходимой гибкостью за пределами ограниченных условий. Именно нейронные сети дают многообещающие альтернативные решения, и многие приложения выигрывают от их использования.

Искусственные нейронные сети получили широкое распространение за последние 20 лет и позволили решать сложные задачи обработки данных, часто значительно превосходя точность других методов искусственного интеллекта, либо являясь единственно возможным методом решения отдельных задач. Нейронные сети (нейросети) успешно применяются в самых различных областях – бизнесе, медицине, технике, геологии, физике. Такой впечатляющий успех определяется несколькими причинами: нейросети – исключительно мощный метод моделирования, позволяющий воспроизводить чрезвычайно сложные зависимости; они *нелинейные* по своей природе, и, кроме того, нейронные сети справляются с проблемой *размерности*, которая не позволяет моделировать линейные зависимости в случае большого числа переменных.

Нейронные сети возникли из исследований в области искусственного интеллекта, а именно, из попыток воспроизвести способность биологических нервных систем обучаться и исправлять ошибки, моделируя низкоуровневую структуру мозга. Основной областью исследований по искусственному интеллекту в 60–80-е гг. были экспертные системы. Такие системы основывались на высокоуровневом моделировании процесса мышления (в частности, на представлении, что процесс нашего мышления построен на манипуляциях с символами). Скоро стало ясно, что подобные системы, хотя и могут принести пользу в некоторых областях, не охватывают некоторые ключевые аспекты человеческого интеллекта. Согласно одной из точек зрения, причина этого состоит в том, что они не в состоянии воспроизвести структуру мозга. Чтобы создать искусственный интеллект, необходимо построить систему с похожей архитектурой.

Мозг человека состоит приблизительно из 86 млрд нейронов, соединенных многочисленными связями (в среднем несколько тысяч связей на один нейрон, однако это число может сильно колебаться). Нейроны – это специальные клетки, способные распространять электрохимические сигналы. Нейрон имеет разветвленную структуру (рис. 1): дендриты (ввод информации), ядро и разветвляющийся выход (аксон).

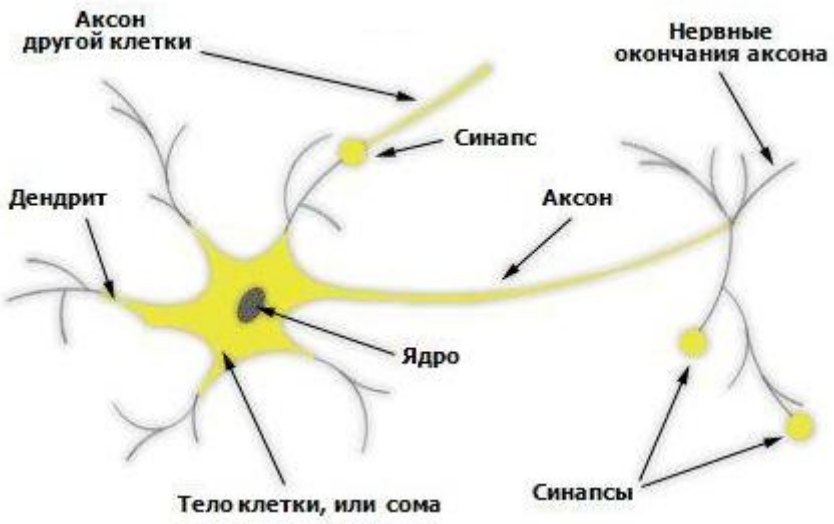


Рис. 1. Структура нейрона

Аксоны клетки соединяются с дендритами других клеток с помощью синапсов. При активации нейрон посылает электрохимический сигнал по своему аксону. Через синапсы этот сигнал достигает других нейронов, которые могут в свою очередь активироваться. Нейрон активируется тогда, когда суммарный уровень сигналов, пришедших в его ядро из дендритов, превысит определенный уровень (порог активации).

Интенсивность сигнала, получаемого нейроном (а следовательно, и возможность его активации), сильно зависит от активности синапсов. Каждый синапс имеет определенную протяженность, и специальные химические вещества передают сигнал вдоль него. Один из самых авторитетных исследователей нейросистем, Дональд Хебб, высказал постулат, что обучение заключается в первую очередь в изменениях «силы» синаптических связей.

Таким образом, будучи построен из очень большого числа совсем простых элементов (каждый из которых берет взвешенную сумму входных сигналов и, в случае если суммарный вход превышает определенный уровень, передает дальше двоичный сигнал), мозг способен решать чрезвычайно сложные задачи. Интересно то, что искусственные нейронные сети способны достичь замечательных результатов при использовании модели, которая ненамного сложнее, чем описанная выше.

1.1. Базовая искусственная модель

Чтобы отразить суть биологических нейронных систем, определение искусственного нейрона дается следующим образом.

1. Нейрон получает входные сигналы (исходные данные либо выходные сигналы других нейронов нейронной сети) через несколько входных

каналов. Каждый входной сигнал проходит через соединение, имеющее определенную интенсивность (или вес), и этот вес соответствует синаптической активности биологического нейрона. С каждым нейроном связано определенное пороговое значение. Вычисляется взвешенная сумма входов, из нее вычитается пороговое значение, и в результате получается величина активации нейрона (она также называется пост-синаптическим потенциалом нейрона – PSP).

2. Сигнал активации преобразуется с помощью функции активации (или передаточной функции), и в результате получается выходной сигнал нейрона.

3. Если при этом использовать ступенчатую функцию активации (т. е. выход нейрона равен нулю, если вход отрицательный, и единице, если вход нулевой или положительный), то такой нейрон будет работать точно так же, как описанный выше естественный нейрон (вычесть пороговое значение из взвешенной суммы и сравнить результат с нулем – это то же самое, что сравнить взвешенную сумму с пороговым значением). В действительности, как мы скоро увидим, пороговые функции редко используются в искусственных нейронных сетях. Необходимо учесть, что веса могут быть отрицательными, – это значит, что синапс оказывает на нейрон не возбуждающее, а тормозящее воздействие (в мозге присутствуют тормозящие нейроны).

Это было описание отдельного нейрона. Теперь возникает вопрос: как соединять нейроны друг с другом? Если сеть предполагается для чего-то использовать, то у нее должны быть входы (принимающие значения интересующих нас переменных из внешнего мира) и выходы (прогнозы или управляющие сигналы). Входы и выходы соответствуют сенсорным и двигательным нервам, например, соответственно, идущим от глаз и в руки. Кроме этого, однако, в сети может быть еще много промежуточных (скрытых) нейронов, выполняющих внутренние функции. Входные, скрытые и выходные нейроны должны быть связаны между собой.

Ключевой вопрос здесь – обратная связь. Простейшая сеть имеет структуру прямой передачи сигнала: сигналы проходят от входов через скрытые элементы и в конце концов приходят на выходные элементы. Такая структура имеет устойчивое поведение. Если же сеть рекуррентная (т. е. содержит обратные связи), то она может быть неустойчива и иметь очень сложную динамику поведения. Рекуррентные сети представляют большой интерес для исследователей в области нейронных сетей, однако при решении практических задач наиболее полезными оказались структуры прямой передачи.

Типичный пример сети с прямой передачей сигнала показан на рис. 2. Нейроны регулярным образом организованы в слои. Входной слой служит просто для ввода значений входных переменных. Каждый из скрытых и

выходных нейронов соединен со всеми элементами предыдущего слоя. Можно было бы рассматривать сети, в которых нейроны связаны только с некоторыми из нейронов предыдущего слоя, однако для большинства приложений сети с полной системой связей предпочтительнее.

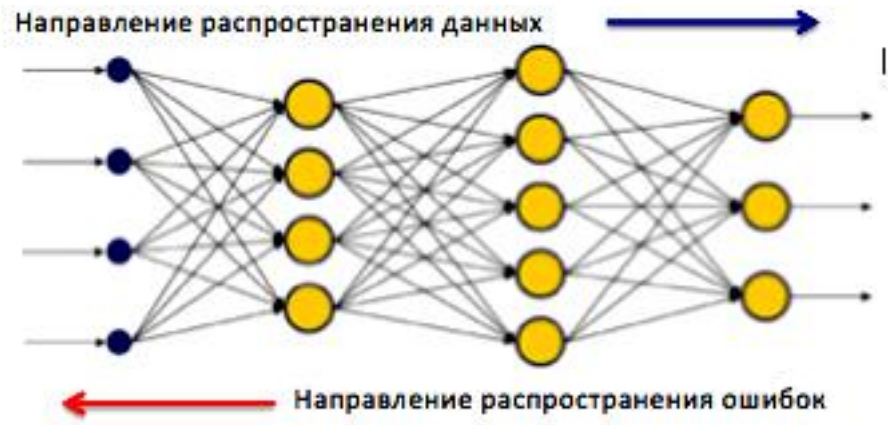


Рис. 2. Сеть с прямой передачей сигнала

При работе сети во входные элементы подаются значения входных переменных, затем последовательно отрабатывают нейроны промежуточных и выходного слоев. Каждый из них вычисляет свое значение активации, беря взвешенную сумму выходов элементов предыдущего слоя и вычитая из нее пороговое значение. Затем значение активации преобразуются с помощью функции активации, и в результате получается выход нейрона. После того как вся сеть отработает, выходные значения элементов выходного слоя принимаются за выход всей сети в целом.

1.2. Свойства нейронных сетей

Несомненно, что технические средства, построенные на тех же принципах, что и биологические нейронные сети, обладают рядом схожих свойств. К таким свойствам относятся:

- параллелизм вычислений;
- распределенное представление информации и вычислений;
- способность к обучению и к обобщению;
- толерантность к ошибкам.

Можно выделить следующие основные идеи, лежащие в основе нейронных сетей и нейромоделирования.

• Нейросеть воспроизводит структуру и свойства нервной системы живых организмов: нейронная сеть состоит из большого числа простых вычислительных элементов (нейронов) и обладает более сложным поведением по сравнению с возможностями каждого отдельного нейрона. Нейросеть

получает на входе набор входных сигналов и выдает соответствующий им ответ (выходные сигналы нейросети), являющийся решением задачи.

• Искусственная нейросеть, как и естественная биологическая нейронная сеть, может обучаться решению задач: нейросеть содержит внутренние адаптивные параметры нейронов и своей структуры и, меняя их, может менять свое поведение.

• Место программирования занимает обучение, тренировка нейронной сети: для решения задачи не нужно программировать алгоритм.

• Нейронная сеть обучается решению задачи на некоем «учебнике» − наборе ситуаций, каждая из которых описывает значения входных сигналов нейросети и требуемый для этих входных сигналов ответ. «Учебник» задает набор эталонных ситуаций с известными решениями, а нейронная сеть при обучении сама находит зависимости между входными сигналами и ответами.

Аппаратная реализация нейросети – нейрокомпьютер – имеет существенные отличия (как по структуре, так и по классу решаемых задач) от вычислительных машин, выполненных в соответствии с традиционной архитектурой фон Неймана. Сравнительные характеристики нейрокомпьютеров и традиционных ЭВМ, приведенные в табл. 1, показывают существенные различия таких основополагающих компонент, как процессор, память и организация вычислений.

Таблица 1

Сравнительные оценки традиционных ЭВМ и нейрокомпьютеров

Ресурс	Традиционная ЭВМ	Нейрокомпьютер
Процессор	Сложный Высокоскоростной Один или несколько	Простой Низкоскоростной Большое количество
Память	Отделена от процессора Локализована Адресация формальная	Интегрирована в процессор Распределенная Адресация по содержанию
Вычисления	Централизованные Последовательные Хранимые программы	Распределенные Параллельные Самообучение
Надежность	Высокая уязвимость	Живучесть
Специализация	Численные операции	Проблемы восприятия

1.3. Применение нейронных сетей

Класс задач, которые можно решить с помощью нейронной сети, определяется тем, как сеть работает, и тем, как она обучается. При работе нейронная сеть принимает значения входных переменных и выдает значения

выходных переменных. Таким образом, сеть можно применять в ситуации, когда имеется определенная известная информация, и мы хотим из нее получить некоторую пока не известную информацию.

Вот некоторые примеры таких задач.

1. **Классификация образов**. Задача состоит в указании принадлежности входного образа (например, речевого сигнала или рукописного символа), представленного вектором признаков, одному или нескольким предварительно определенным классам. К известным приложениям относятся распознавание букв, распознавание речи, классификация сигнала электрокардиограммы, классификация клеток крови.

2. **Кластеризация/категоризация**. При решении задачи кластеризации, которая известна также как классификация образов «без учителя», отсутствует обучающая выборка с метками классов. Алгоритм кластеризации основан на подобии образов и размещает близкие образы в один кластер. Известны случаи применения кластеризации для извлечения знаний, сжатия данных и исследования свойств данных.

3. **Аппроксимация функций**. Предположим, что имеется обучающая выборка ((x_1, y_1), (x_2, y_2), ..., (x_n, y_n) – пары данных вход-выход), которая генерируется неизвестной функцией $y(x)$, искаженной шумом. Задача аппроксимации состоит в нахождении оценки неизвестной функции $y_e(x)$. Аппроксимация функций необходима при решении многочисленных инженерных и научных задач моделирования.

4. **Предсказание/прогноз**. Пусть заданы n дискретных отсчетов $\{y(t_1), y(t_2), ..., y(t_n)\}$ в последовательные моменты времени $t_1, t_2, ..., t_n$. Задача состоит в предсказании значения $y(t_{n+1})$ в некоторый будущий момент времени t_{n+1}. Предсказание/прогноз имеют значительное влияние на принятие решений в бизнесе, науке и технике. Предсказание цен на фондовой бирже и прогноз погоды являются типичными приложениями техники предсказания/прогноза.

5. **Оптимизация**. Многочисленные проблемы в математике, статистике, технике, науке, медицине и экономике могут рассматриваться как проблемы оптимизации. Задачей алгоритма оптимизации является нахождение такого решения, которое удовлетворяет системе ограничений и максимизирует или минимизирует целевую функцию. Задача коммивояжера, относящаяся к классу NP-полных, является классическим примером задачи оптимизации.

6. **Память, адресуемая по содержанию**. В модели вычислений фон Неймана обращение к памяти доступно только посредством адреса, который не зависит от содержания памяти. Более того, если допущена ошибка в вычислении адреса, то может быть найдена совершенно иная информация.

Ассоциативная память, или память, адресуемая по содержанию, доступна по указанию заданного содержания. Содержимое памяти может быть вызвано даже по частичному входу или искаженному содержанию. Ассоциативная память чрезвычайно желательна при создании мультимедийных информационных баз данных.

7. **Управление**. Рассмотрим динамическую систему, заданную совокупностью $\{u(t), y(t)\}$, где $u(t)$ является входным управляющим воздействием, а $y(t)$ – выходом системы в момент времени t. В системах управления с эталонной моделью целью управления является расчет такого входного воздействия $u(t)$, при котором система следует по желаемой траектории, диктуемой эталонной моделью. Примером является оптимальное управление двигателем.

Необходимо отметить важное условие применения нейронных сетей: между известными входными значениями и неизвестными выходами должна существовать связь. Эта связь может быть искажена шумом, но она обязательно должна существовать.

Как правило, нейронная сеть используется тогда, когда неизвестен точный вид связей между входами и выходами. Если бы он был известен, то связь можно было бы моделировать непосредственно. Другая существенная особенность нейронных сетей состоит в том, что зависимость между входом и выходом отыскивается в процессе обучения сети. Для обучения нейронных сетей применяются алгоритмы двух типов (разные типы сетей используют разные типы обучения): управляемое («обучение с учителем») и не управляемое («без учителя»). Чаще всего применяется обучение с учителем.

Для управляемого обучения сети пользователь должен подготовить набор обучающих данных. Эти данные представляют собой примеры входных данных и соответствующих им выходов. Сеть учится устанавливать связь между первыми и вторыми. Обычно обучающие данные берут из исторических сведений.

Кроме того, нейронная сеть может обучаться с помощью того или иного алгоритма управляемого обучения (наиболее известным из них является метод обратного распространения), при котором имеющиеся данные используются для корректировки весов и пороговых значений сети таким образом, чтобы минимизировать ошибку прогноза на обучающем множестве. Если сеть обучена хорошо, она приобретает способность моделировать (неизвестную) функцию, связывающую значения входных и выходных переменных, и впоследствии такую сеть можно использовать для прогнозирования в ситуации, когда выходные значения неизвестны.

1.4. Классификация нейронных сетей

Существует широкий спектр вариантов структурной организации нейронных сетей (рис. 3) на различной программно-аппаратной базе. Всегда можно подобрать наиболее оптимальный вариант для конкретной задачи − все определяется свойствами задачи и требованиями к решению.

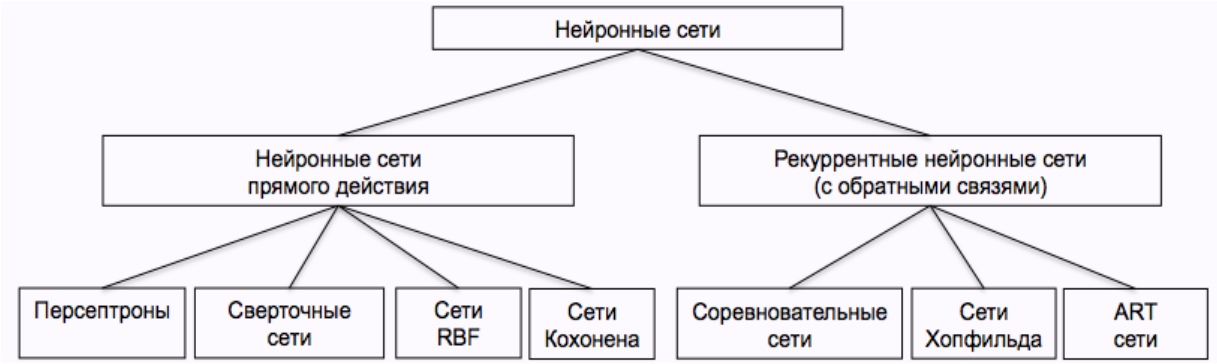


Рис. 3. Классификация искусственных нейронных сетей

Однако нельзя придумать некоторую универсальную сеть, которая бы подошла для различных типов задач, поэтому нейросети создаются с использованием одного из подходов, когда нейросеть строится для решения:

1) определенного класса задач;
2) конкретной задачи.

Наиболее распространенным семейством сетей прямого действия являются многослойные персептроны, в них нейроны расположены слоями и соединены однонаправленными связями, идущими от входа к выходу сети. Сети прямого действия являются статическими в том смысле, что на заданный вход они вырабатывают одну совокупность выходных значений, не зависящих от предыдущего состояния сети.

Рекуррентные сети являются динамическими, так как в силу обратных связей в них модифицируются входы нейронов, что приводит к изменению состояния сети. Поведение рекуррентных сетей описывается дифференциальными или разностными уравнениями, как правило, первого порядка. Это гораздо расширяет области применения нейросетей и способы их обучения. Сеть организована так, что каждый нейрон получает входную информацию от других нейронов, возможно, и от самого себя, и от окружающей среды.

Также можно выделить два основных подхода к реализации нейросетей: цифровой и аналоговый. Преимуществом аналоговых реализаций являются: высокое быстродействие, надежность и экономичность. Однако сфера возможного массового использования обучаемых аналоговых нейрочипов достаточно узка. Это обусловлено большой сложностью аппарат-

ной реализации высокоэффективных обучающих алгоритмов и необходимостью специальной подготовки потенциальных пользователей для оптимальной организации адаптивного процесса. В то же время широкое распространение могут получить обученные аналоговые нейрокомпьютеры (нейросети) с фиксированной или незначительно подстраиваемой структурой связей – нейропроцессоры.

Задача создания нейропроцессоров сводится к обучению цифровой нейросетевой модели нужному поведению на обычном цифровом компьютере.

Сети также можно классифицировать по числу слоев. В этом случае важную роль играет нелинейность активационной функции, так как, если бы она не обладала данным свойством или не входила в алгоритм работы каждого нейрона, результат функционирования любой *n*-слойной нейронной сети сводился бы к перемножению входного вектора сигналов φ на матрицу весовых коэффициентов. То есть фактически такая нейронная сеть эквивалентна однослойной нейросети с весовой матрицей единственного слоя *W*. Кроме того, нелинейность иногда вводится и в синаптические связи.

1.5. Структура и принципы работы нейронной сети

В качестве модели нейрона был выбран бинарный пороговый элемент, вычисляющий взвешенную сумму входных сигналов и формирующий на выходе сигнал величины 1, если эта сумма превышает определенное пороговое значение, и 0 – в противном случае. К настоящему времени данная модель не претерпела серьезных изменений. Были введены новые виды активационных функций. Структурная модель технического нейрона представлена на рис. 4.

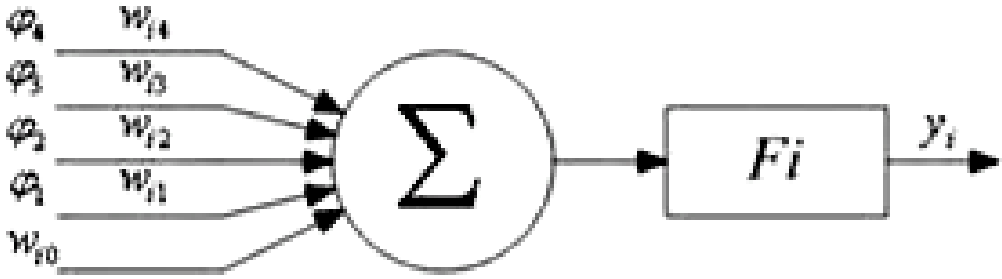


Рис. 4. Формальная модель искусственного нейрона

На вход искусственного нейрона поступает некоторое множество сигналов, каждый из которых является выходом другого нейрона, или входным сигналом нейросетевой модели. Каждый вход умножается на соответствующий вес, аналогичный синаптической силе биологического нейрона. Вес определяет, насколько соответствующий вход нейрона влияет на его

состояние. Все произведения суммируются, определяя уровень активации нейрона. Состояние нейрона определяется по формуле

$$S = \sum_{i=1}^{n} \varphi_i w_i ,$$

где φ_i – множество сигналов, поступающих на вход нейрона; w_i – весовые коэффициенты нейрона. Далее сигнал S преобразуется активационной (передаточной) функцией нейрона F в выходной сигнал y. Математически это можно выразить формулой

$$y = F\left(\sum_{i=1}^{n} \varphi_i w_i + w_0 \right),$$

где n – размерность вектора входов; w_0 – «нейронное смещение», вводимое для инициализации сети, подключается к неизменяемому входу +1; F – активационная функция нейрона.

Нейроны могут группироваться в сетевую структуру различным образом. Функциональные особенности нейронов и способ их объединения в сетевую структуру определяют особенности нейросети. Для решения задач идентификации и управления наиболее адекватными являются многослойные нейронные сети (МНС) прямого действия или многослойные персептроны. При проектировании МНС нейроны объединяют в слои, каждый из которых обрабатывает вектор сигналов от предыдущего слоя. Минимальной реализацией является двухслойная нейронная сеть, состоящая из входного (распределительного), промежуточного (скрытого) и выходного слоев.

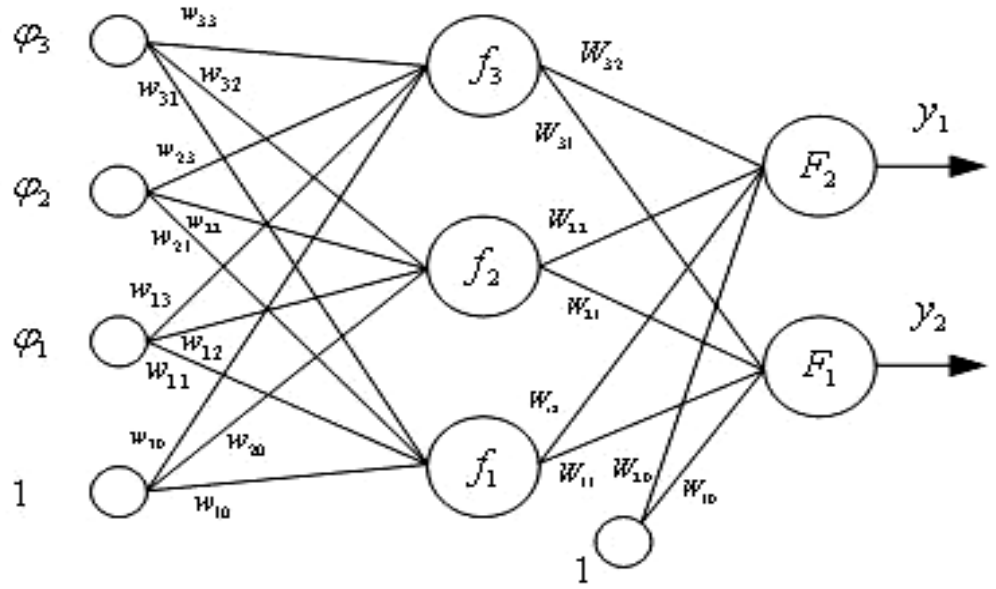


Рис. 5. Структурная схема двухслойной нейронной сети

Реализация модели двухслойной нейронной сети прямого действия имеет следующее математическое представление:

$$y(\theta) = F_i\left(\sum_{j=1}^{n_h} W_{ij} f_j\left(\sum_{j=1}^{n_h} w_{ij}\varphi_j + w_{j0}\right) + W_{j0}\right),$$

где n_φ – размерность вектора входов φ нейронной сети; n_h – число нейронов в скрытом слое; θ – вектор настраиваемых параметров нейронной сети, включающий весовые коэффициенты и нейронные смещения (w_{ji}, W_{ij}); $f_j(x)$ – активационная функция нейронов скрытого слоя, $F_i(x)$ – активационная функция нейронов выходного слоя.

Персептрон представляет собой сеть, состоящую из нескольких последовательно соединенных слоев формальных нейронов (рис. 5). На низшем уровне иерархии находится входной слой, состоящий из сенсорных элементов, задачей которого является только прием и распространение по сети входной информации. Далее имеются один или, реже, несколько скрытых слоев. Каждый нейрон на скрытом слое имеет несколько входов, соединенных с выходами нейронов предыдущего слоя или непосредственно со входными сенсорами φ_1, φ_2, ..., φ_n, и один выход. Нейрон характеризуется уникальным вектором настраиваемых параметров θ. Функция нейрона состоит в вычислении взвешенной суммы его входов с дальнейшим нелинейным преобразованием ее в выходной сигнал.

1.6. Обучение нейронной сети

Способность к обучению является фундаментальным свойством мозга. В контексте нейросети процесс обучения может рассматриваться как настройка архитектуры сети и весов связей для эффективного выполнения специальной задачи. Обычно нейронная сеть должна настроить веса связей по имеющейся обучающей выборке. Функционирование сети улучшается по мере итеративной настройки весовых коэффициентов. Свойство сети обучаться на примерах делает их более привлекательными по сравнению с системами, которые следуют определенной системе правил функционирования, сформулированной экспертами.

Для конструирования процесса обучения прежде всего необходимо иметь модель внешней среды, в которой функционирует нейронная сеть, знать доступную для сети информацию. Эта модель определяет парадигму обучения. Кроме того, необходимо понять, как модифицировать весовые параметры сети, какие правила обучения управляют процессом настройки. Алгоритм обучения означает процедуру, в которой используются правила обучения для настройки весов.

Существуют три парадигмы обучения: «с учителем», «без учителя» (самообучение) и смешанная. В первом случае нейронная сеть располагает правильными ответами (выходами сети) на каждый входной пример. Веса настраиваются так, чтобы сеть производила ответы, как можно более близкие к известным правильным ответам. Усиленный вариант обучения «с учителем» предполагает, что известна только критическая оценка правильности выхода нейронной сети, но не сами правильные значения выхода. Обучение «без учителя» не требует знания правильных ответов на каждый пример обучающей выборки. В этом случае раскрывается внутренняя структура данных или корреляции между образцами в системе данных, что позволяет распределить образцы по категориям. При смешанном обучении часть весов определяется посредством обучения «с учителем», в то время как другая часть получается с помощью самообучения.

Теория обучения рассматривает три фундаментальных свойства, связанных с обучением по примерам: емкость, сложность образцов и вычислительная сложность. Под емкостью понимается, сколько образцов может запомнить сеть и какие функции и границы принятия решений могут быть на ней сформированы. Сложность образцов определяет число обучающих примеров, необходимых для достижения способности сети к обобщению. Слишком малое число примеров может вызвать «переобученность» сети, когда она хорошо функционирует на примерах обучающей выборки, но плохо – на тестовых примерах, подчиненных тому же статистическому распределению. Известны 4 основных типа правил обучения: коррекция по ошибке, машина Больцмана, правило Хебба и обучение методом соревнования. В табл. 2 представлены некоторые алгоритмы обучения и связанные с ними архитектуры сетей. В последней колонке перечислены задачи, для которых может быть применен каждый алгоритм. Каждый алгоритм обучения ориентирован на сеть определенной архитектуры и предназначен для ограниченного класса задач.

Таблица 2

Алгоритмы обучения

Парадигма	Обучающее правило	Архитектура	Алгоритм обучения
С учителем	Коррекция по ошибке	Перцептроны	Алгоритмы обучения перцептрона Обратное распространение Adaline и Madaline
	Правило Больцмана	Рекуррентная	Алгоритм обучения Больцмана
	Правило Хебба	Многослойный персептрон	Линейный дискриминантный анализ
	Соревнование	Соревновательная	Векторное квантование
		Сеть ART	ARTMap

Окончание табл. 2

Парадигма	Обучающее правило	Архитектура	Алгоритм обучения
Без учителя	Коррекция по ошибке	Многослойный персептрон	Проекция Саммона
	Правило Хебба	Персептроны или соревновательная	Анализ главных компонентов
		Сеть Хопфилда	Обучение ассоциативной памяти
	Соревнование	Соревновательная	Векторное квантование
		Кохонена	SOM Кохонена
		Сети ART	ART1, ART2
Смешанная	Коррекция по ошибке и соревнование	Сеть RBF	Алгоритм обучения RBF

Правило коррекции по ошибке. При обучении «с учителем» для каждого входного примера задан желаемый выход d. Реальный выход сети y может не совпадать с желаемым. Принцип коррекции по ошибке при обучении состоит в использовании сигнала $(d - y)$ для модификации весов, обеспечивающей постепенное уменьшение ошибки. Обучение имеет место только в случае, когда перцептрон ошибается. Известны различные модификации этого алгоритма обучения.

Обучение Больцмана. Представляет собой стохастическое правило обучения, которое следует из информационных теоретических и термодинамических принципов. Целью обучения Больцмана является такая настройка весовых коэффициентов, при которой состояния видимых нейронов удовлетворяют желаемому распределению вероятностей. Обучение Больцмана может рассматриваться как специальный случай коррекции по ошибке, в котором под ошибкой понимается расхождение корреляций состояний в двух режимах.

Правило Хебба. Самым старым обучающим правилом является постулат обучения Хебба. Хебб опирался на следующие нейрофизиологические наблюдения: если нейроны с обеих сторон синапса активизируются одновременно и регулярно, то сила синаптической связи возрастает. Важной особенностью этого правила является то, что изменение синаптического веса зависит только от активности нейронов, которые связаны данным синапсом. Это существенно упрощает цепи обучения в реализации VLSI.

Обучение методом соревнования. В отличие от обучения Хебба, в котором множество выходных нейронов могут возбуждаться одновременно, при соревновательном обучении выходные нейроны соревнуются между

собой за активизацию. Это явление известно как правило «победитель берет все». Подобное обучение имеет место в биологических нейронных сетях. Обучение посредством соревнования позволяет кластеризовать входные данные: подобные примеры группируются сетью в соответствии с корреляциями и представляются одним элементом. При обучении модифицируются только веса «победившего» нейрона. Эффект этого правила достигается за счет такого изменения сохраненного в сети образца (вектора весов связей победившего нейрона), при котором он становится чуть ближе ко входному примеру.

Можно заметить, что сеть никогда не перестанет обучаться, если параметр скорости обучения не равен 0. Некоторый входной образец может активизировать другой выходной нейрон на последующих итерациях в процессе обучения. Это ставит вопрос об устойчивости обучающей системы. Система считается устойчивой, если ни один из примеров обучающей выборки не изменяет своей принадлежности к категории после конечного числа итераций обучающего процесса. Один из способов достижения стабильности состоит в постепенном уменьшении до 0 параметра скорости обучения. Однако это искусственное торможение обучения вызывает другую проблему, называемую пластичностью и связанную со способностью адаптироваться к новым данным. Эти особенности обучения методом соревнования известны под названием дилеммы стабильности-пластичности Гроссберга.

2. ПАКЕТ МОДЕЛИРОВАНИЯ NEURALNET

Пакет neuralnet [1] служит для построения и обучения нейронных сетей с использованием ряда алгоритмов. Наряду с основной функцией neuralnet() пакет содержит ряд вспомогательных методов: compute(), confidence.interval(), gwplot(), plot.nn() и prediction(). Их изучению посвящен настоящий раздел.

2.1. Функция neuralnet()

Функция neuralnet() служит для построения и обучения нейронных сетей с использованием алгоритмов обратного распространения, упругого обратного распространения (resilient backpropagation – RPROP), с весом или без веса, с возвратом или модифицированной глобально сходящейся версии (globally convergent version – GRPROP). Функция позволяет осуществлять гибкие настройки ошибок и функции активации. Кроме того, в ней реализуется расчет обобщенных весов.

Ниже приведен вариант вызова функции neuralnet() с указанием значений ее параметров, которые устанавливаются по умолчанию:

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
        stepmax = 1e+05, rep = 1, startweights = NULL,
        learningrate.limit = NULL,
        learningrate.factor = list(minus = 0.5, plus = 1.2),
        learningrate=NULL, lifesign = «none»,
        lifesign.step = 1000, algorithm = «rprop+»,
        err.fct = «sse», act.fct = «logistic»,
        linear.output = TRUE, exclude = NULL,
        constant.weights = NULL, likelihood = FALSE)
```

Здесь formula – символьное описание модели сети, например formula y1 + y2 ~ x1 + x2 + x3 – описывает сеть с тремя входами x1, x2, x3 и двумя выходами y1, y2;

data – фрейм данных, содержащий имена переменных, указанных в формуле, и набор значений этих переменных, используемый для обучения;

hidden – вектор целых чисел, определяющих количество скрытых нейронов (вершин) в каждом слое, например вектор hidden = c(6, 12, 8) определяет три скрытых слоя с 6, 12 и 8 нейронами на каждом слое соответственно;

threshold – числовое значение с указанием порога функции ошибки для частных производных как критерия остановки;

stepmax – максимальное число шагов для подготовки нейронной сети. Достижение этой максимальной величины приводит к остановке процесса подготовки нейронной сети;

rep – число повторений для тренировки нейронной сети;

startweights – вектор, содержащий начальные значения для весов. Веса не будут инициализироваться случайным образом;

learningrate.limit – вектор или список, содержащий самое низкое и самое высокое ограничения на скорость обучения. Используется только для RPROP и GRPROP;

learningrate.factor – вектор или список, содержащий коэффициенты усиления для верхней и нижней скоростей обучения. Используется только для RPROP и GRPROP;

learningrate – числовое значение с указанием скорости обучения, используемой только для традиционного обратного распространения;

lifesign – целое число, определяющее размер шага для печати минимального порога в режиме lifesign;

algorithm – строка, содержащая тип алгоритма для вычисления нейронной сети. Типы «backprop», «rprop +», «rprop –», «sag» или «slr», «backprop» относятся к обратному распространению, «rprop +» и «rprop –» относятся к упругому обратному распространению с весом и без веса с возвратом, в то время как «sag» и «slr» используют модифицированный глобально сходящийся алгоритм (grprop). Смотрите пояснения ниже для получения дополнительной информации;

err.fct – дифференцируемая функция ошибки, которая используется для расчета погрешности. Альтернативно могут быть использованы строки «SSE – sum of squared errors» и «CE – cross-entropy», которые определят суммарную квадратичную ошибку и кросс-энтропию;

act.fct – дифференцируемая функция активации, которая используется для сглаживания результата векторного произведения входов или нейронов и весов. Альтернативно возможны строки «logistic» и «tanh» для задания логистической функции и гиперболического тангенса;

linear.output – логический. Если функцию активации act.fct не применять к выходным нейронам, устанавливается TRUE, в противном случае – FALSE;

exclude – вектор или матрица с указанием весов, которые исключены из вычисления. Если задан вектор, то должны быть известны точные позиции весов. Матрица с n-строками и 3 столбцами будет исключать n веса следующим образом: первый столбец – веса перед первым слоем, второй столбец – веса входного нейрона и третий столбец – веса выходного нейрона;

constant.weights – вектор с указанием значения весов, которые исключены из процесса обучения и рассматриваются как фиксированные;

likelihood – логический. Если функция ошибки равна отрицательной логарифмической функции правдоподобия (likelihood), будут рассчиты-

ваться информационные критерии AIC и BIC. Кроме того, возможно использование confidence.interval – доверительного интервала.

Глобально алгоритм сходится на основе упругого обратного распространения без веса и дополнительно модифицирует одну скорость обучения, либо скорость обучения, связанную с наименьшим абсолютным градиентом (sag), либо саму наименьшую скорость обучения (slr). Темпы обучения в алгоритме grprop имеют границы, определенные в learningrate.limit.

Функция neuralnet() возвращает объекты класса nn, состоящие из следующих компонент:

- call – согласованный вызов;
- response – выходные переменные, извлеченные из данных;
- covariate – входные переменные, извлеченные из данных;
- model.list – список, содержащий входные и выходные переменные, извлеченные из формулы аргумента;
- err.fct – функция ошибки;
- act.fct – функция активации;
- data – данные;
- net.result – список, содержащий общий результат нейронной сети для каждого повторения;
- weights – список, содержащий встроенные веса нейронной сети для каждого повторения;
- generalized.weights – список, содержащий обобщенные веса нейронной сети для каждого повторения;
- result.matrix – матрица, содержащая достигнутый порог, необходимые шаги, ошибки, AIC и BIC (если вычислены) и веса для каждого повторения. Каждый столбец представляет одно повторение;
- startweights – список, содержащий начальные веса нейронной сети для каждого повторения.

Пример 1

```
AND <- c(rep(0,7),1)
OR <- c(0,rep(1,7))
binary.data <- data.frame(expand.grid(c(0,1), c(0,1), c(0,1)), AND, OR)
net <- neuralnet(AND+OR~Var1+Var2+Var3,  binary.data,
        hidden=0, linear.output=FALSE)
```

В качестве данных для обучения сети использовался фрейм binary.data:

```
  Var1 Var2 Var3 AND  OR
1    0    0    0   0   0
2    1    0    0   0   1
3    0    1    0   0   1
```

```
4   1   1   0   0   1
5   0   0   1   0   1
6   1   0   1   0   1
7   0   1   1   0   1
8   1   1   1   1   1
```

Данный фрейм представляет собой таблицу истинности логических функций AND и OR от трех входных переменных на всех возможных наборах значений.

Проанализируем работу функции neuralnet(), рассматривая возвращаемые компоненты. Отметим, что для извлечения соответствующей компоненты comp объекта net необходимо указывать ее наименование после символа $, например net$comp.

Компонента call показывает формулу сети, данные для обучения и все входные параметры, которые имеют значения, отличные от значений по умолчанию:

```
>net$call
neuralnet(formula = AND + OR ~ Var1 + Var2 + Var3, data = binary.data,
   hidden = 0, linear.output = FALSE)
```

Компонента model.list содержит списки входных и выходных переменных сети:

```
> net$model.list
$response
[1] «AND» «OR»
$variables
[1] «Var1» «Var2» «Var3»
```

Посмотрим, какие функции активации и ошибки используются по умолчанию «logistic» и «sse» соответственно:

```
>net$act.fct

function (x)
{
1/(1 + exp(-x))
}
```

```
>net$err.fct

function (x, y)
{
   1/2 * (y – x)^2
}
```

Эти функции можно заменить, указывая альтернативные значения «tanh» и «ce», соответственно:

```
function (x)
{
tanh(x)
}
```

```
function (x, y)
{
 -(y * log(x) + (1 – y) * log(1 – x))
}
```

Компонента net.result формирует список, содержащий общий результат работы нейронной сети:

```
> net$net.result
                [,1]            [,2]
1 0.0004257798874 0.01111658098
2 0.0082448164652 0.99365087982
3 0.0082416029384 0.99797617953
4 0.1395522164597 0.99999985433
5 0.0083789389427 0.99474869475
6 0.1415653424697 0.99999962081
7 0.1415175803706 0.99999987965
8 0.7628789015482 0.99999999999
```

В частности, этот результат показывает, что функция OR смоделирована сетью более точно: логическая единица соответствует значениям более 0,99, а логический нуль – значениям менее 0,012, в то время как для функции AND логическая единица не превышает 0,77, а логический нуль доходит до 0,14.

Компонента weights показывает список, содержащий встроенные веса нейронной сети:

```
> net$weights
[,1]       [,2]
[1,] -7.761162171 -4.488138671
[2,]  2.971270580  9.541208340
[3,]  2.970877501 10.688881034
[4,]  2.987542420  9.732152143
```

Используя функцию plot(), можно увидеть, как распределены эти веса в построенной сети (рис. 6).

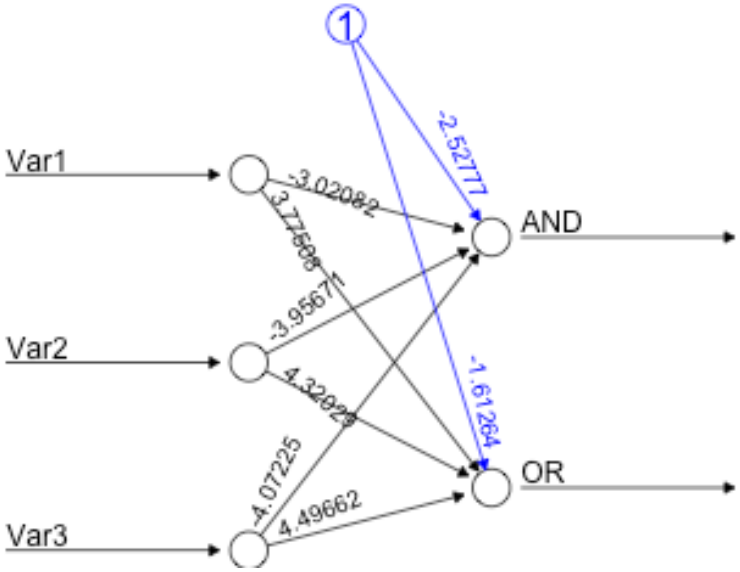


Рис. 6. Нейронная сеть примера 1

Рассмотрение остальных компонент оставим для самостоятельного изучения и анализа. Также будет весьма полезно, оперируя входными параметрами, оценить их влияние на качество моделирования.

Пример 2

```
data(infert, package=«datasets»)
net.infert <- neuralnet(case~parity+induced+spontaneous, infert,
        err.fct=«ce», linear.output=FALSE, likelihood=TRUE)
plot(net.infert)
```

В сети рис. 7 один скрытый слой и число повторений для тренировки сети равно 1 (по умолчанию).

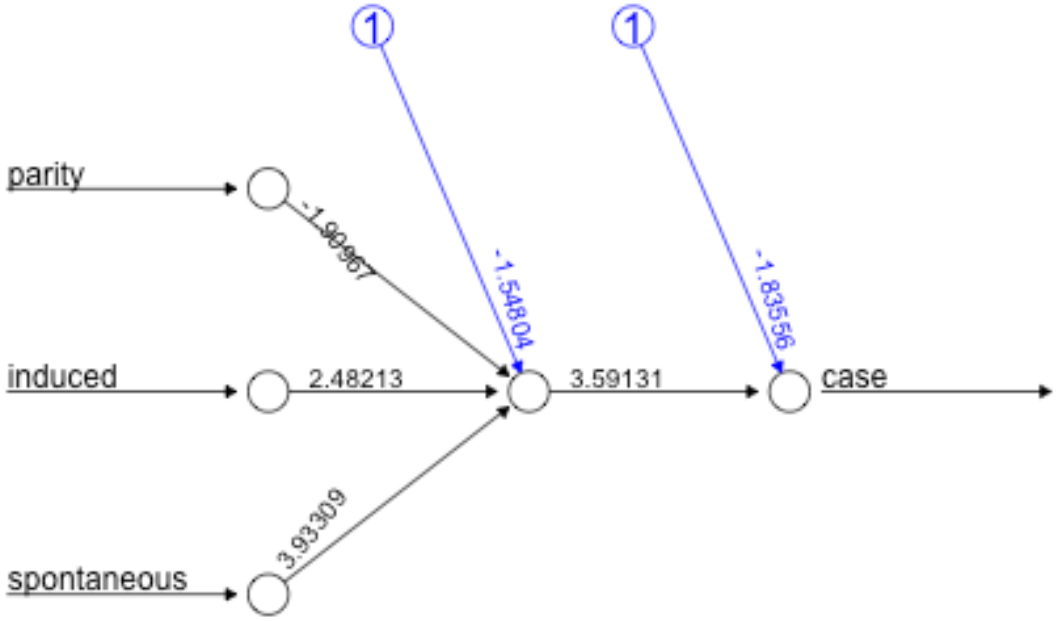


Рис. 7. Нейронная сеть примера 2

2.2. Метод compute()

Метод применим к объектам класса nn, обычно создаваемым neuralnet. Вычисляет выходы всех нейронов для векторов данных обученной нейронной сети. При его использовании необходимо убедиться, что порядок входных переменных в новой матрице или новом фрейме данных тот же самый, что и в оригинальной нейронной сети.

Приведем вариант вызова метода compute() с указанием значений его параметров, которые устанавливаются по умолчанию:

```
compute(x, covariate, rep = 1)
```

Здесь x – объект класса нейронных сетей;

covariate – фрейм данных или матрица, содержащая переменные, используемые для обучения нейронной сети;

rep – целое число, указывающее число повторений при обучении нейронной сети.

В результате метод compute() возвращает список, содержащий следующие компоненты:

- neurons – список выходов нейронов, для каждого слоя нейронной сети;
- net.result – матрицу, содержащую общий результат нейронной сети.

Пример 3

```
Var1 <- runif(50, 0, 100)
sqrt.data <- data.frame(Var1, Sqrt=sqrt(Var1))
print(net.sqrt <- neuralnet(Sqrt~Var1,  sqrt.data, hidden=10, threshold=0.01))
compute(net.sqrt, (1:10)^2)$net.result
```

В качестве данных для обучения сети использовался фрейм sqrt.data, в котором сформирована таблица из 50 случайных вещественных чисел и значений квадратного корня.

Метод compute(), используя обученную нейронную сеть, вычисляет квадратные корни для чисел 1^2–10^2:

```
          [,1]
 [1,] 1.053352055
 [2,] 1.985307241
 [3,] 3.002431207
 [4,] 3.998049506
 [5,] 4.999291631
 [6,] 6.003230166
 [7,] 6.997450496
 [8,] 7.995180965
 [9,] 9.006797008
[10,] 9.982220273
```

Для обеспечения достаточной точности вычислений выбрано значение порога функции ошибки threshold = 0,01 как критерия остановки и 10 нейронов в скрытом слое.

2.3. Метод confidence.interval()

Метод confidence.interval() применим к объектам класса nn, обычно создаваемым neuralnet. Рассчитывает доверительные интервалы весов и сетевых информационных критериев (network information criteria) NIC. Все

доверительные интервалы рассчитываются в предположении локальной идентификации данной нейронной сети. Если это предположение нарушено, то результаты не будут разумными. Также нужно убедиться, что выбранная функция ошибки равна негативной логарифмической функции правдоподобия, в противном случае результаты также не значимы.

Ниже приведен вариант вызова метода confidence.interval() с указанием значений его параметров, которые устанавливаются по умолчанию:

```
confidence.interval(x, alpha = 0.05)
```

Здесь x – нейронная сеть;

alpha – численное значение устанавливает уровень доверия к (1-альфа).

В результате метод confidence.interval() возвращает список, содержащий следующие компоненты:

- lower.ci – список, содержащий более низкие доверительные интервалы всех весов нейронной сети, различающиеся по повторениям;
- upper.ci – список, содержащий верхние доверительные интервалы всех весов нейронной сети, различающиеся по повторениям;
- NIC – вектор, включающий сетевой информационный критерий для каждого повторения.

Пример 4 (см. пример 2)

```
data(infert, package=«datasets»)
print(net.infert <- neuralnet(case~parity+induced+spontaneous,
                    infert, err.fct=«ce», linear.output=FALSE))
 confidence.interval(net.infert)

$lower.ci
$lower.ci[[1]]
$lower.ci[[1]][[1]]
          [,1]
[1,]  0.2314469832
[2,] -0.7460196908
[3,] -5.6691016442
[4,] -8.7442737295

$lower.ci[[1]][[2]]
          [,1]
[1,] -0.7116526129
[2,] -6.4162960007

$upper.ci
$upper.ci[[1]]
$upper.ci[[1]][[1]]
```

```
          [,1]
[1,] 2.8542829892
[2,] 4.5293065766
[3,] 0.7552012900
[4,] 0.9529224881

$upper.ci[[1]][[2]]
          [,1]
[1,]  4.2596071994
[2,] -0.8081559202

$nic
[1] 135.6753816
```

2.4. Метод gwplot()

Метод применим к объектам класса nn, обычно создаваемым neuralnet, и предназначен для построения графиков обобщенных весов для одной входной переменной и одной переменной отклика.

Приведем вариант вызова метода gwplot() с указанием значений его параметров, которые устанавливаются по умолчанию:

```
gwplot(x, rep = NULL, max = NULL, min = NULL, file = NULL,
        selected.covariate = 1, selected.response = 1,
        highlight = FALSE, type=«p», col = «black», ...)
```

Здесь x – объект класса nn;

rep – целое число, указывающее число повторений. Если rep = «best», то будет построено повторение с минимальной ошибкой. Если не указано, будут построены все повторы;

max – максимум оси y. По умолчанию максимум устанавливается как наибольшее значение y;

min – минимум оси y. По умолчанию минимум устанавливается как наименьшее значение y;

file – строка символов с названием графика. Если название не указано, график не будет сохранен;

selected.covariate – либо строка имен входных переменных, либо целое число, указывающее порядковый номер входной переменной в обобщенных весах графика, по умолчанию – с первой переменной;

selected.response – либо строка имен выходных переменных, либо целое число, указывающее порядковый номер выходной переменной в обобщенных весах графика, по умолчанию – с первой переменной;

highlight – логическое значение, указывающее, следует ли выделить (красный цвет) самую лучшую попытку (наименьшая ошибка). Действует, только если rep = NULL. По умолчанию имеет значение FALSE;

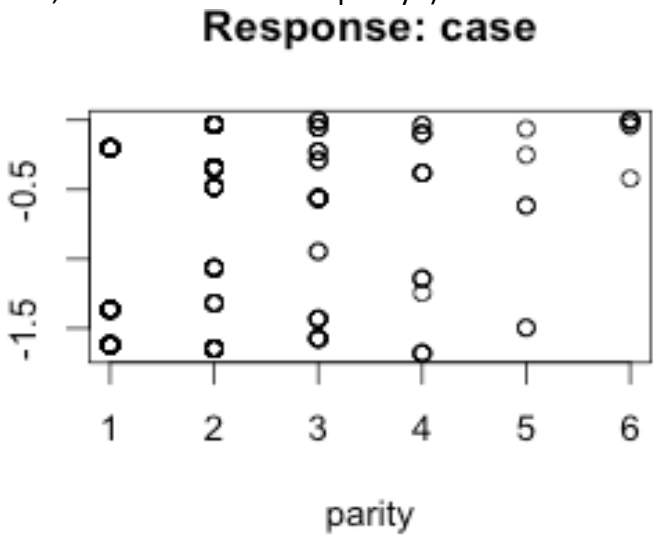
type – буква, указывающая тип графика; фактически может быть любого типа, как в plot.default;

col – цвет обобщенных весов;

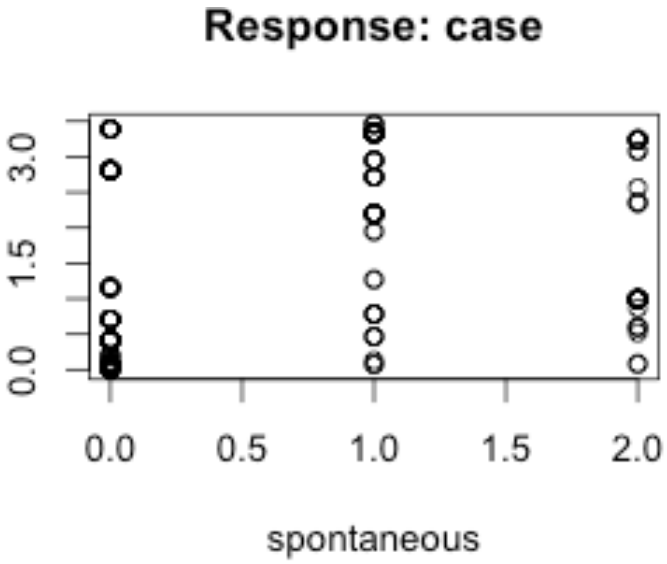
... – аргументы, которые передаются методам, такие как графические параметры (см. par).

Пример 5

```
data(infert, package=«datasets»)
print(net.infert <- neuralnet(case~parity+induced+spontaneous,
      nfert,err.fct=«ce», linear.output=FALSE, likelihood=TRUE))
gwplot(net.infert, selected.covariate=«parity»)
```




```
gwplot(net.infert, selected.covariate=«spontaneous»)
```



2.5. Метод plot.nn()

Метод служит для графического построения нейронной сети. Он предназначен для проверки весов объектов класса nn, как правило, созданных с помощью функции neuralnet(). Большое число входных параметров служит для задания необходимой цветовой гаммы графического представления сети.

Приведем вариант вызова метода plot.nn() с указанием значений его параметров, которые устанавливаются по умолчанию:

```
plot(x, rep = NULL, x.entry = NULL, x.out = NULL,
radius = 0.15, arrow.length = 0.2, intercept = TRUE,
intercept.factor = 0.4, information = TRUE,
information.pos = 0.1, col.entry.synapse = «black»,
col.entry = «black», col.hidden = «black»,
col.hidden.synapse = «black», col.out = «black»,
col.out.synapse = «black», col.intercept = «blue»,
fontsize = 12, dimension = 6, show.weights = TRUE,
file = NULL, ...)
```

Здесь x – объект класса nn;

rep – повторения нейронной сети. Если rep = «best», будет построено повторение с наименьшей ошибкой. Если rep не указано, будут нанесены все повторы, каждый в отдельном окне;

x.entry – x-координата входного слоя. Зависит от arrow.length по умолчанию;

x.out – x-координата выходного слоя;

radius – radius of the neurons;

arrow.length – длина входного и выходного массива;

intercept intercept – логическое значение, указывающее на участок перехвата;

intercept.factor – x-позиционный коэффициент перехвата. Чем ближе коэффициент к 0, тем точка пересечения ближе к левому нейрону;

information – логическое значение, указывает, следует ли добавлять ошибки и шаги к графику;

information.pos – y-позиция информации;

col.entry.synapse – цвет синапсов, ведущих к входным нейронам;

col.entry – цвет входных нейронов;

col.hidden – цвет нейронов в скрытом слое;

col.hidden.synapse – цвет взвешенных синапсов;

col.out – цвет выходных нейронов;

col.out.synapse – цвет синапсов, ведущих от выходных нейронов;

col.intercept – цвет перехвата;

fontsize – размер шрифта текста;

dimension – размер графика в дюймах;

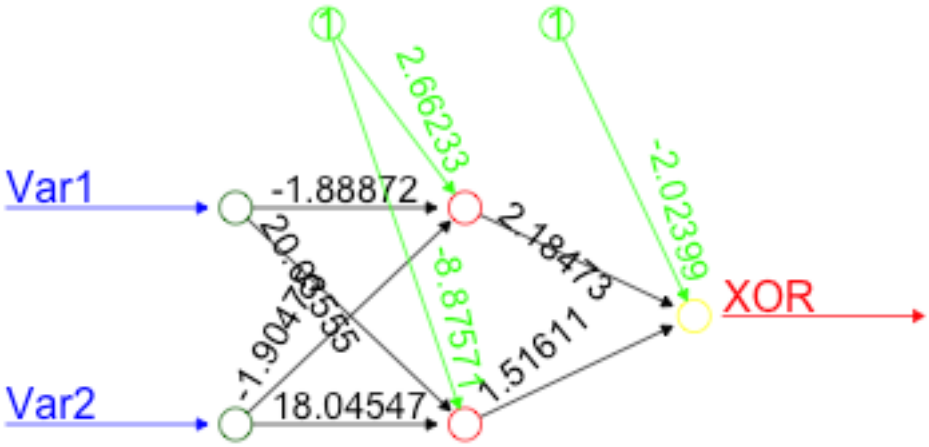
show.weights – логическое значение, указывает, нужно ли печатать расчетные веса выше синапсов;

file – строка символов с названием графика для записи. Если не указано, график не будет сохранен;

... – аргументы, которые передаются методам, такие как графические параметры (см. par).

Пример 6

```
XOR <- c(0,1,1,0)
xor.data <- data.frame(expand.grid(c(0,1), c(0,1)), XOR)
print(net.xor <- neuralnet( XOR~Var1+Var2, xor.data, hidden=2, rep=5))
plot(net.xor, col.hidden = «red», col.intercept = «green»,col.entry=«dark green»,
col.entry.synapse = «blue»,col.out.synapse=«red», col.out=«yellow», fontsize=16)
```



2.6. Метод prediction()

Метод служит для прогнозирования работы нейронной сети. Он предназначен для обобщенных линейных моделей (generalized linear models – glm) объектов класса nn, как правило, созданных с помощью функции neuralnet().

На первом этапе данные в dataframe будут изменены на усредненную реакцию, среднее значение всех откликов сети, соответствующих одному и тому же входному вектору. Будет рассчитана data.error как функция ошибки между исходной реакцией и новым средним откликом. На втором этапе все повторяющиеся строки будут удалены, чтобы получить быстрый обзор данных. Чтобы получить обзор результатов нейронной сети и объектов glm, матрица будет связана с выходом нейронной сети и соответствующими значениями объекта glm (при наличии), и будут сокращены все дублирующие строки.

Пример 7

```
Var1 <- rpois(100,0.5)
Var2 <- rbinom(100,2,0.6)
Var3 <- rbinom(100,1,0.5)
SUM <- as.integer(abs(Var1+Var2+Var3+(rnorm(100))))
sum.data <- data.frame(Var1+Var2+Var3, SUM)
print(net.sum <- neuralnet( SUM~Var1+Var2+Var3, sum.data, hidden=1, act.fct=«tanh»))
main <- glm(SUM~Var1+Var2+Var3, sum.data, family=poisson())
full <- glm(SUM~Var1*Var2*Var3, sum.data, family=poisson())
prediction(net.sum, list.glm=list(main=main, full=full))
```

				$rep1	$data	$glm.main	$glm.full
	Var1	Var2	Var3	SUM	SUM	SUM	SUM
1	0	0	0	0.3148064728	0.3000000000	0.6670445303	0.3058033025
2	2	0	0	1.7030487315	2.0000000000	1.3332115860	2.4475212570
3	0	1	0	1.0071624893	0.8461538462	1.0503629399	0.8807173210
4	1	1	0	1.7381728270	1.6000000000	1.4849489239	1.5335652374
5	2	1	0	2.5051515236	3.0000000000	2.0993441628	2.6703486818
6	0	2	0	1.7733762464	2.6000000000	1.6539560036	2.5364768569
7	1	2	0	2.5411198940	2.7500000000	2.3382776510	2.7184427365
8	2	2	0	3.3045710020	2.0000000000	3.3057362838	2.9134627788
9	3	2	0	4.0260015770	4.0000000000	4.6734793764	3.1224734842
10	0	0	1	0.9515413079	1.6666666667	1.0222341629	1.1838977807
11	1	0	1	1.6782006768	2.0000000000	1.4451819104	1.7482591053
12	3	0	1	3.2098422106	4.0000000000	2.8884627327	3.8123168054
13	0	1	1	1.7132663100	1.5294117647	1.6096629713	1.6935302204
14	1	1	1	2.4796514170	2.1428571429	2.2756584474	2.3751829226
15	2	1	1	3.2448868176	3.5000000000	3.2172084851	3.3312035699
16	3	1	1	3.9708939628	4.0000000000	4.5483233428	4.6720263599
17	0	2	1	2.5156240446	2.5000000000	2.5346588631	2.4225441201
18	1	2	1	3.2798449215	3.4285714286	3.5833699079	3.2269209403
19	2	2	1	4.0031974268	4.0000000000	5.0659834678	4.2983814696

2.7. Примеры к разделу

Все примеры к разделу содержат проверенный в RStudio код и комментарии к определенным строкам кода в виде результата, полученного в процессе вычислений. Там, где комментарий дается только к одной строке, соответствующая строка выделена в коде жирным шрифтом.

Пример 8

```
AND <- c(rep(0,3),1) #1 комментарий
binary.data <- data.frame(expand.grid(c(0,1),c(0,1)), AND) #2 комментарий
print(net <- neuralnet(AND~Var1+Var2,  binary.data, hidden=0,
                rep=10, err.fct=«ce», linear.output=FALSE)) #3 комментарий
plot(net) #4 комментарий
compute(net,expand.grid(c(0,1), c(0,1)))$net.result #5 комментарий
```

#1 комментарий

AND = 0001

#2 комментарий

binary.data =

Var1	Var2	AND
0	0	0
1	0	0
0	1	0
1	1	1

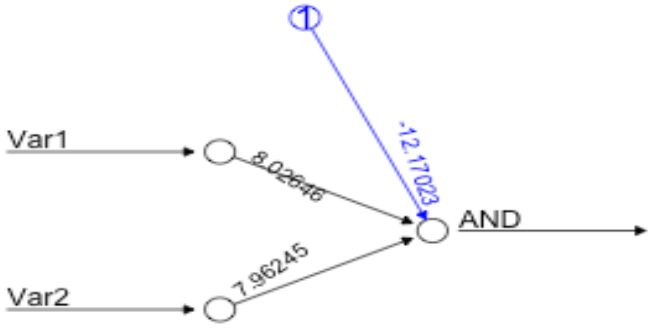
#3 комментарий

Call: neuralnet(formula = AND ~ Var1 + Var2, data = binary.data, hidden = 0, rep = 10, err.fct = «ce», linear.output = FALSE)

3 repetitions were calculated.

	Error	Reached Threshold	Steps
2	0.04495236104	0.009299918714	114
3	0.06309904979	0.009285865662	119
1	0.06332917116	0.009104723098	117

#4 комментарий



#5 комментарий

expand.grid(c(0,1), c(0,1)) = Var1 Var2

1 0 0

2 1 0

3 0 1

4 1 1

compute = [1,] 0.000006308766118

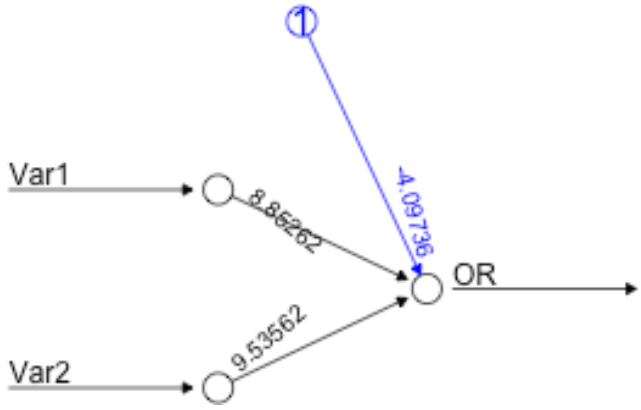
[2,] 0.016041426066345

[3,] 0.015241085372694

[4,] 0.975606741097143

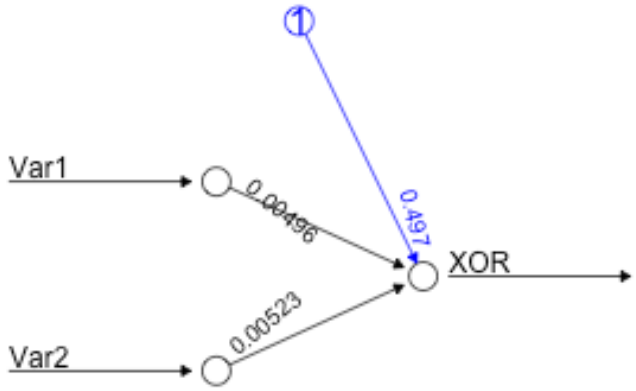
Пример 9

```
OR <- c(0,rep(1,3))
binary.data <- data.frame(expand.grid(c(0,1),c(0,1)), OR)
print(net <- neuralnet(OR~Var1+Var2,  binary.data, hidden=0,
                 rep=3, err.fct=«ce», linear.output=FALSE))
plot(net)
```



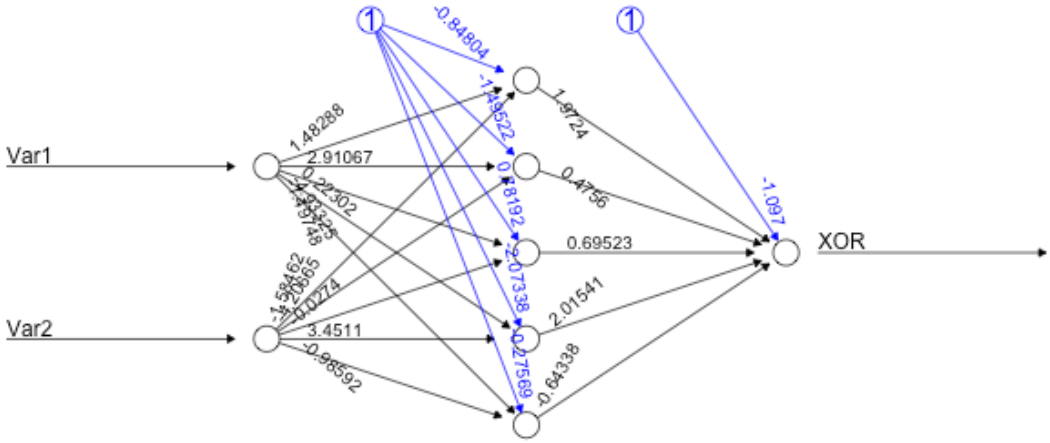
Пример 10

XOR <- c(0,1,1,0)
xor.data <- data.frame(expand.grid(c(0,1), c(0,1)), XOR)
print(net.xor <- neuralnet(XOR~Var1+Var2, xor.data, hidden=0, rep=5))
plot(net.xor, rep=«best»)



Пример 11

XOR <- c(0,1,1,0)
xor.data <- data.frame(expand.grid(c(0,1), c(0,1)), XOR)
print(net.xor <- neuralnet(XOR~Var1+Var2, xor.data, hidden=5, **rep=5**))
plot(net.xor)



prediction(net.xor)
Data Error: 0;

```
$rep1
  Var1 Var2          XOR
1    0    0  0.008457622835
2    1    0  0.993978873239
3    0    1  0.996403720703
4    1    1  0.008863814003

$rep2
  Var1 Var2          XOR
1    0    0  0.006802657909
2    1    0  0.993743345608
3    0    1  0.992469487115
4    1    1  0.011629989857

$rep3
  Var1 Var2              XOR
1    0    0 -0.00105837081
2    1    0  0.99305253912
3    0    1  0.99313343280
4    1    1  0.00870173949

$rep4
  Var1 Var2            XOR
1    0    0  0.009862892925
2    1    0  0.995748076370
3    0    1  0.994980783218
4    1    1 -0.005142245999

$rep5
  Var1 Var2          XOR
1    0    0  0.00745609358
2    1    0  0.99619528536
3    0    1  0.99467627379
4    1    1  0.01033379656

$data
  Var1 Var2 XOR
1    0    0   0
2    1    0   1
3    0    1   1
4    1    1   0
```

Пример 12

```
Var1 <- runif(50, 0, 100)
sqrt.data <- data.frame(Var1, Sqrt=sqrt(Var1))
print(net.sqrt <- neuralnet(Sqrt~Var1, sqrt.data, hidden=10,
                            threshold=0.01))
compute(net.sqrt, (1:10)^2)$net.result
```

```
 [1,] 0.8477740258
 [2,] 1.9038178924
 [3,] 3.0016156935
 [4,] 3.9969460979
 [5,] 5.0021900595
 [6,] 5.9962817933
 [7,] 7.0009828046
 [8,] 7.9986902299
 [9,] 9.0017703356
[10,] 9.9810129183
```

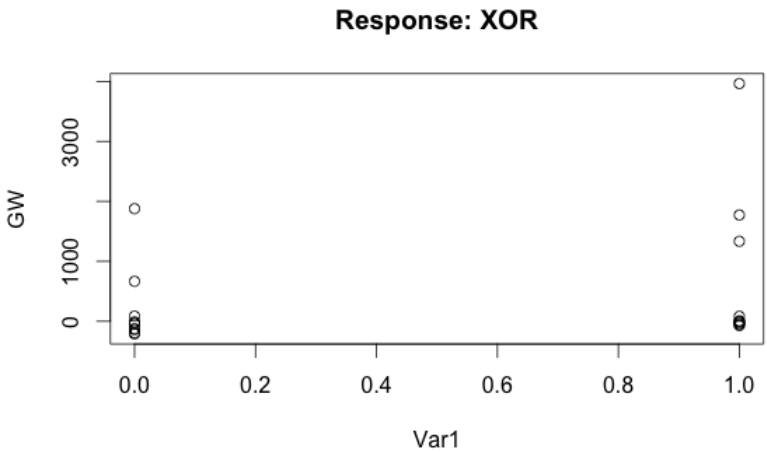
prediction(net.sqrt)

```
Data Error:          0;
$rep1
                 Var1              Sqrt
1   0.3289304907 0.5735064559
2   8.9676589472 2.9961918853
3  10.2785215247 3.2067588460
…............
49 95.4132530140 9.7647036231
50 95.4222697765 9.7651441351

$data
                 Var1              Sqrt
1   0.3289304907 0.5735246208
2   8.9676589472 2.9946049735
3  10.2785215247 3.2060133382
……................
49 95.4132530140 9.7679707726
50 95.4222697765 9.7684323090
```

Пример 13

```
XOR <- c(0,1,1,0)
xor.data <- data.frame(expand.grid(c(0,1), c(0,1)), XOR)
print(net.xor <- neuralnet(XOR~Var1+Var2, xor.data, hidden=5, rep=5))
```

gwplot(net.xor, selected.covariate=«Var1»)



3. ПАКЕТ МОДЕЛИРОВАНИЯ NNET

Пакет nnet служит для построения и обучения нейронных сетей прямого действия с единственным скрытым слоем, и для полиномиальных логарифмо-линейных моделей.

3.1. Функция nnet()

Функция nnet() предоставляет нейронную сеть с одним скрытым слоем, возможно, со связями с пропуском нейронов скрытого слоя. Способ вызова функции для класса formula:

```
nnet(formula, data, weights, ...,
        subset, na.action, contrasts = NULL),
```

по умолчанию:

```
nnet(x, y, weights, size, Wts, mask,
      linout = FALSE, entropy = FALSE, softmax = FALSE,
      censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
      maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
      abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

Здесь formula – формула класса формы ~ x1 + x2 + ...;

x – матрица или фрейм данных входных переменных для примеров;

y – матрица или фрейм данных целевых переменных для примеров;

weights – (тематические) веса для каждого примера. Если не указаны, то по умолчанию 1;

size – количество нейронов в скрытом слое. Может быть ноль, если сеть с пропуском уровня;

data – фрейм данных, из которого берутся значения переменных, указанных в формуле;

subset – индекс-вектор с указанием примеров (тем), которые будут использоваться в обучающей выборке (*примечание*: если индекс-вектор есть, то должен быть указан);

na.action – функция, определяющая действие, которое будет выполнено, если обнаружены NAs (отсутствующие значения). Действием по умолчанию является выполнение процедуры сбоя. Альтернативой является na.omit, что приводит к отказу от случаев с отсутствующими значениями требуемых переменных (*примечание*: если функция есть, то должна быть указана);

contrasts – список контрастов, которые будут использоваться для некоторых или всех факторов, возникающих в качестве переменных в модели формулы;

Wts – начальный вектор параметров. Если отсутствует, то параметры выбираются случайно;

mask – логический вектор, указывающий, какие параметры должны быть оптимизированы (по умолчанию все);

linout – переключатель для линейных выходных нейронов, по умолчанию – для логарифмических выходных нейронов;

entropy – переключатель для энтропии (максимальное правдоподобие), по умолчанию – для энтропии наименьших квадратов;

softmax – переключатель для лог-линейной модели и модели максимального правдоподобия. Параметры linout, entropy, softmax и censored являются взаимоисключающими;

censored – вариант по softmax, в котором ненулевые цели означают возможные классы. Таким образом, ряд softmax (0, 1, 1) означает один пример каждого из классов 2 и 3, но для censored это означает один пример, его известный класс 2 или 3;

skip – переключатель для добавления связи входа с выходом с пропуском скрытого слоя;

rang – первоначальные случайные веса. Значение около 0,5, если входов много, и в этом случае они должны быть выбраны таким образом, чтобы произведене rang * max(|x|) составляло около 1;

decay – параметр веса распада. По умолчанию 0;

maxit – максимальное число итераций. По умолчанию 100;

Hess – если этот флаг установлен, то возвращается лучший вариант подходящих весов в виде компонентов гессиана;

trace – переключатель для отслеживания оптимизации. По умолчанию TRUE;

MaxNWts – максимально допустимое число весов. В коде нет внутреннего предела, но увеличение MaxNWts, вероятно, позволит найти более точные веса, но это займет много времени;

abstol – прекратить, если подходящий критерий падает ниже abstol, указывая, по существу, идеальное соответствие;

reltol – остановка, если оптимизатор не может уменьшить подходящий критерий с коэффициентом по крайней мере (1 – reltol);

... – аргументы, передающиеся в другие методы или получаемые из других методов.

Если ответ в формуле является фактором, будет построен соответствующий сетевой классификатор: он будет иметь один выход, если число уровней два, либо число выходов, равное числу классов, а также SOFTmax – выходной каскад для необходимого числа уровней. Если ответ не является фактором, он передается неизмененным на nnet.default.

Объект класса nnet или nnet.formula – небольшая внутренняя структура, имеющая следующие компоненты:

- wts – лучший найденный набор весов;
- value – значение критерия совпадения плюс срок распада веса;
- fitted.value – подходящие значения для тренировочных данных;
- residuals – остатки для обучающих данных;
- convergence – 1, если было достигнуто максимальное число итераций, 0 – в противном случае.

Пример 14

```
ir <- rbind(iris3[,,1],iris3[,,2],iris3[,,3]) #1 комментарий
targets <- class.ind( c(rep(«s», 50), rep(«c», 50), rep(«v», 50)) ) #2 комментарий
samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
ir1 <- nnet(ir[samp,], targets[samp,], size = 2, rang = 0.1,
        decay = 5e-4, maxit = 200) #3 комментарий
test.cl <- function(true, pred) {
  true <- max.col(true)
  cres <- max.col(pred)
  table(true, cres)
}
test.cl(targets[-samp,], predict(ir1, ir[-samp,])) #4 комментарий
#import the function from Github
library(devtools)
source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a
505ff044412703516c34f1a4684a5/nnet_plot_update.r') #5 комментарий
plot.nnet(ir1) #6 комментарий
```

#1 комментарий

	Sepal L.	**Sepal W.**	**Petal L.**	**Petal W.**
1	5.1	3.5	1.4	0.2
2	4.9	3.0	1.4	0.2
…	…	…	…	…
3	4.7	3.2	1.3	0.2
150	5.9	3.0	5.1	1.8

#2 комментарий

	c	**s**	**v**
1	0	1	0
2	0	1	0
3	0	1	0
…			
150	0	0	1

#3 комментарий

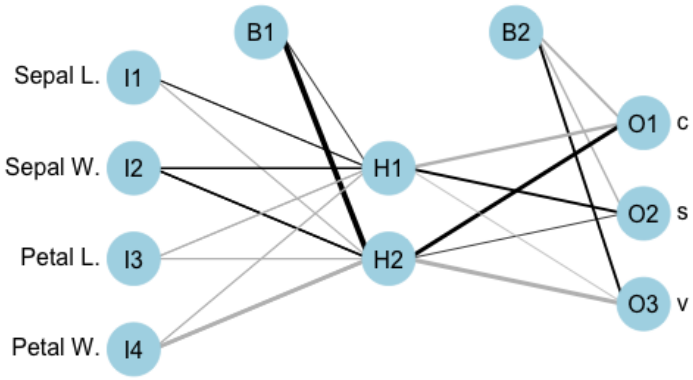
```
# weights:  19
initial  value 57.031148
```

```
iter  10 value 36.195891
iter  20 value 19.581396
…
iter 200 value 1.790721
final  value 1.790721
stopped after 200 iterations
```

```
#4 комментарий
   cres
true   1   2   3
  1   25   0   0
  2    0  25   0
  3    2   0  23
```

```
#5 комментарий
SHA-1 hash of file is 74c80bd5ddbc17ab3ae5ece9c0ed9beb612e87ef
```

```
#6 комментарий
```



3.2. Функция class.ind()

Создает функцию индикатора класса из данного фактора. Вызов функции:

```
class.ind(cl)
```

Здесь cl – фактор или вектор классов для различных случаев.

В результате формируется матрица, содержащая нули, за исключением столбца, соответствующего классу.

Функция class.ind() определена следующим образом:

```
class.ind <- function(cl)
  {
    n <- length(cl)
    cl <- as.factor(cl)
    x <- matrix(0, n, length(levels(cl)) )
```

```
    x[(1:n) + n*(unclass(cl)-1)] <- 1
    dimnames(x) <- list(names(cl), levels(cl))
    x
}
```

Пример 15

```
set.seed(1)
x <- factor(sample(1:3,10,TRUE)) #1 комментарий
class.ind(x) #2 комментарий

#1 комментарий
> x
 [1] 1 2 2 3 1 3 3 2 2 1
Levels: 1 2 3

#2 комментарий
       1 2 3
 [1,]  1 0 0
 [2,]  0 1 0
 [3,]  0 1 0
 [4,]  0 0 1
 [5,]  1 0 0
 [6,]  0 0 1
 [7,]  0 0 1
 [8,]  0 1 0
 [9,]  0 1 0
[10,] 1 0 0
```

3.3. Метод multinom()

Метод multinom() строит полиномиальные логарифмо-линейные модели с помощью нейронных сетей.

Вызов метода:

```
multinom(formula, data, weights, subset, na.action,
         contrasts = NULL, Hess = FALSE, summ = 0, censored = FALSE,
         model = FALSE, ...)
```

Здесь formula – выражение в форме ответ (выход) ~ предикторы (входы). Ответ должен быть фактором или матрицей с K столбцами, которые будут интерпретироваться как значения для каждого из классов K. Может быть включено смещение: оно должно быть представлено числовой матрицей с K столбцами, если ответом являются либо матрица с K столбцами, либо фактор с $K \geq 2$ классами, либо числовой вектор для фактора отклика с 2 уровнями;

data – дополнительный (необязательный) фрейм данных, в котором интерпретированы переменные, входящие в формулу;

weights – дополнительные подходящие веса;

subset – выражение, показывающее, какое подмножество строк данных следует использовать. По умолчанию включены все наблюдения;

na.action – функция фильтрации недостающих данных;

contrasts – список контрастов, которые будут использоваться для некоторых или всех факторов, возникающих в качестве переменных в формуле;

Hess – ключ, указывающий, должен ли возвращаться гессиан (информационная матрица наблюдаемые/ожидаемые значения);

summ – целое число, отличное от нуля, указывающее количество строк и настройки весов;

censored = TRUE – если Y представляет собой матрицу с K колонками, можно интерпретировать данные, как один за возможных классов;

model – флаг. Если он установлен, фрейм сохраняется в качестве компонента модели возвращенного объекта;

... – дополнительные аргументы для nnet.

Метод multinom() вызывает nnet. Переменные в правой части формулы должны быть масштабированы примерно до [0,1], либо схождение будет медленным или не произойдет.

В результате метод multinom () возвращает следующие компоненты:

- deviance – остаточное отклонение по сравнению с полностью насыщенной моделью (что точно объясняет отдельные наблюдения);
- edf – (эффективное) число степеней свободы используемой модели;
- AIC – подходящий информационный критерий;
- Hessian – если ключ Hess включен;
- model – если флаг установлен.

Пример 16

```
options(contrasts = c(«contr.treatment», «contr.poly»))
library(MASS)
example(birthwt) #1 комментарий
bwt.mu <- multinom(low ~ ., bwt) #2 комментарий

#1 комментарий
brthwt> bwt <- with(birthwt, {
brthwt+ race <- factor(race, labels = c(«white», «black», «other»))
brthwt+ ptd <- factor(ptl > 0)
brthwt+ ftv <- factor(ftv)
brthwt+ levels(ftv)[-(1:2)] <- «2+»
brthwt+ data.frame(low = factor(low), age, lwt, race, smoke = (smoke > 0),
```

```
brthwt+          ptd, ht = (ht > 0), ui = (ui > 0), ftv)
brthwt+ })

brthwt> options(contrasts = c(«contr.treatment», «contr.poly»))

brthwt> glm(low ~ ., binomial, bwt)

Call:  glm(formula = low ~ ., family = binomial, data = bwt)

Coefficients:
(Intercept)        age        lwt    raceblack    raceother
   0.82302    -0.03723    -0.01565      1.19241      0.74068
  smokeTRUE     ptdTRUE      htTRUE      uiTRUE        ftv1
   0.75553      1.34376      1.91317      0.68020     -0.43638
     ftv2+
   0.17901

Degrees of Freedom: 188 Total (i.e. Null);  178 Residual
Null Deviance:      234.7
Residual Deviance: 195.5         AIC: 217.5

#2 комментарий
# weights:  12 (11 variable)
initial  value 131.004817
iter  10 value 98.029803
final  value 97.737759
converged
```

3.4. Метод nnetHess()

Метод nnetHess() оценивает гессиан (матрица вторых производных) указанной нейронной сети. Он обычно вызывается с помощью аргумента Hess = TRUE в nnet() или в multinom().

Вызов метода:

```
nnetHess(net, x, y, weights)
```

Здесь net – объект класса nnet, возвращаемый nnet;

x – обучающие данные;

y – классы для обучающих данных;

weights – (тематические) веса, используемые в подходящей сети nnet.

Пример 17

```
ir <- rbind(iris3[,,1], iris3[,,2], iris3[,,3])
targets <- matrix(c(rep(c(1,0,0),50), rep(c(0,1,0),50), rep(c(0,0,1),50)),150, 3, byrow=TRUE)
samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
ir1 <- nnet(ir[samp,], targets[samp,], size=2, rang=0.1, decay=5e-4,
```

```
            maxit=200) #1 комментарий
eigen(nnetHess(ir1, ir[samp,], targets[samp,]), TRUE)$values #2 комментарий

#1 комментарий
# weights:  19
initial  value 56.363335
iter  10 value 48.476340
iter  20 value 29.676611
. . .
iter 200 value 26.434747
final  value 26.434747
stopped after 200 iterations

#2 комментарий
[1] 64.6297832648 19.7638412778  0.3236392305  0.1720687451  0.1595663085
[6]  0.0868587467  0.0495710962  0.0180492853  0.0111170950  0.0059058616
[11]  0.0044094835  0.0034012449  0.0030334369  0.0018706829  0.0016419805
[16]  0.0013232198  0.0010353038  0.0005585442 -0.0011342777
```

3.5. Метод predict()

Метод predict() предсказывает новые примеры от обученной нейронной сети. Вызов метода:

```
predict(object, newdata, type = c(«raw»,»class»), ...)
```

Здесь object – объект класса nnet, возвращаемый nnet;

newdata – матрица или фрейм данных тестовых примеров. Вектором считается вектор-строка, содержащая один пример;

type – тип выхода;

... – аргументы, передающиеся в другие методы или из других методов.

Этот метод реализован с помощью функции прогнозирования predict() для класса nnet. Он может быть вызван как predict(x) для объекта x соответствующего класса, так и непосредственно путем вызова predict.nnet (x), независимо от класса объекта.

Если type = «raw», то возвращается матрица значений тренированной сети; если type = «class», то возвращается соответствующий класс (который, вероятно, полезен только в случае, если сеть была порождена nnet.formula).

Пример 18

```
ird <- data.frame(rbind(iris3[,,1], iris3[,,2], iris3[,,3]),
            species = factor(c(rep(«s»,50), rep(«c», 50), rep(«v», 50))))
ir.nn2 <- nnet(species ~ ., data = ird, subset = samp, size = 2, rang = 0.1,
          decay = 5e-4, maxit = 200) #1 комментарий
```

```
table(ird$species[-samp], predict(ir.nn2, ird[-samp,], type = «class»)) #2 комментарий

#1
# weights:  19
initial  value 82.354675
iter  10 value 34.798991
iter  20 value 22.428795
iter  30 value 6.944567
iter  40 value 5.351012
iter  50 value 5.321461
iter  60 value 5.310114
iter  70 value 5.199628
iter  80 value 4.452037
iter  90 value 4.314860
iter 100 value 3.930743
iter 110 value 3.793143
iter 120 value 3.755575
iter 130 value 3.729552
iter 140 value 3.711989
final  value 3.710825
converged

#2
   c  s  v
 c 24  0  1
 s  0 25  0
 v  1  0 24
```

3.6. Метод which.is.max()

Метод which.is.max() возвращает индекс элемента, случайно выбранного из максимальных элементов вектора.

Вызов метода:

```
which.is.max(x)
```

Здесь x – вектор.

Пример 19

```
 x = c(32,7,90,2,0,90,-5,33,90)
 which.is.max(x)
[1] 9
which.is.max(x)
[1] 3
```

4. ПАКЕТ МОДЕЛИРОВАНИЯ RSNNS

Пакет Stuttgart Neural Network Simulator (SNNS) – это библиотека, содержащая множество стандартных реализаций нейронных сетей. При использовании SNNS низкоуровневого интерфейса доступны все алгоритмические возможности пакета. Кроме того, пакет содержит удобный интерфейс высокого уровня, так что наиболее распространенные топологии и алгоритмы обучения нейронных сетей интегрируются в R.

4.1. Функция analyzeClassification()

Функция analyzeClassification() преобразует непрерывные значения в бинарные, что может быть использовано для классификации.

Вызов функции:

```
analyzeClassification(y, method = «WTA», l = 0, h = 0)
```

Здесь y – входные значения;

method – WTA или 402040;

l – нижняя граница, для метода 402040 l = 0,4;

h – верхняя граница, для метода 402040 h = 0,6.

Метод 402040

Образец классифицируется правильно, если:

1) выход только одного выходного блока имеет значение >=h;

2) обучаемый выход данного устройства имеет максимальное выходное значение > 0;

3) выход всех других выходных блоков <=l.

Образец классифицируется некорректно, если пп. 1 и 3 имеют значения, указаные выше, а п. 2 утверждает, что обучаемый выход данного устройства не имеет максимального выходного значения или нет обучаемого выхода со значением > 0.

Образец признается неклассифицированным во всех остальных случаях.

Метод получил свое название от широко используемого значения по умолчанию l = 0,4 и h = 0,6.

Метод WTA

Образец признается классифицированным правильно, если:

1) есть выходной блок со значением большим, чем выходное значение всех других выходных блоков (это выходное значение должно быть равно a);

2) a > h;

3) обучаемый выход данного устройства имеет максимальное выходное значение > 0;

4) выход всех других выходных блоков < a – l.

Образец классифицируется некорректно, если указанные в пп. 1, 2, 4 величины имеют значения, определенные выше, а п. 3 утверждает, что обучаемый выход данного устройства не имеет максимального выходного значения или нет обучаемого выхода со значением > 0.

Образец признается неклассифицированным во всех остальных случаях. Обычно используемыми значениями по умолчанию для этого метода являются: l = 0 и h = 0.

Пример 20

```
> (y=rnorm(7))
[1]  0.2274959  0.5953962  0.1491366 -1.7319829  2.0395894  0.4019920  1.0919739
> analyzeClassification(y, method = «WTA»)
[1] 5
```

4.2. Методы art1() и art2()

Сети ART (Adaptive Resonance Theory) выполняют кластеризацию, находя прототипы. Они в основном предназначены для решения дилеммы стабильности/пластичности, которая является одной из центральных проблем в нейронных сетях. Суть проблемы (или дилеммы) стабильности/пластичности памяти в том, что нейросеть при восприятии новой информации должна не просто добавить ее в память, но соотнести новую информацию с уже запомненной и, если есть какое-то сходство между новой и старой информацией, скорректировать запомненную информацию, т. е. «принять во внимание» новую информацию.

По сути, сеть ART реализует принципы ассоциативной памяти с возможностью восстановления по входной информации множества ассоциативных друг другу и входным данным изображений. Она обладает свойствами дообучения, стабильного и компактного хранения запомненной ранее информации, что позволяет эту сеть использовать для разработки специализированных баз знаний, использующих ассоциативную информацию.

Сеть ART – попытка приблизить механизм запоминания образов в ИНС к биологическому. Результатом работы АРТ является устойчивый набор запомненных образов и возможность выборки «похожего» вектора по произвольному предъявленному на входе вектору. Важные качества АРТ – динамическое запоминание новых образов без полного переобучения и отсутствие потерь уже запомненных образов при предъявлении новых.

Входной вектор сети $X = X_1, X_2, \ldots, X_n$ имеет размерность n (рис. 8). В слое распознавания F_R запоминается m классов образов, по одному классу

на каждый нейрон. Основную работу по классификации производят слои сравнения F_C и распознавания F_R, состоящие соответственно из n и m нейронов. Работа блоков управления G_1, G_2 и G_3 описывается следующими формулами:

$G_1 = OR(X_n)\ AND\ NOT(OR(R_m))$;

$G_2 = OR(X_n)$;

$G_3 = SUM(C_n)/SUM(X_n) < h$.

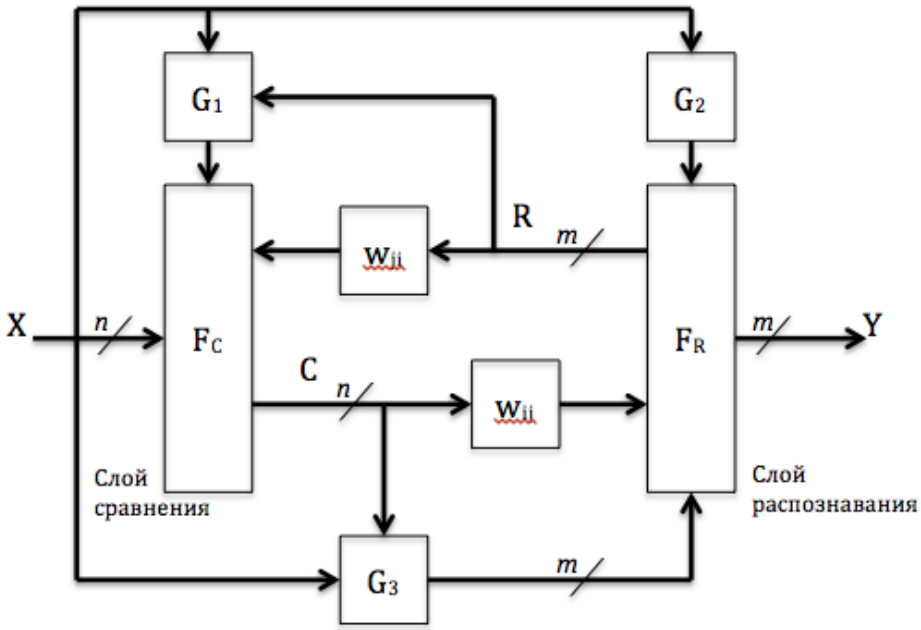


Рис. 8. Структурная схема нейронной сети ART

***Блок* G_1** обеспечивает единичный сигнал для слоя сравнения F_C, если на вход сети подан вектор X (нулевой вектор на входе недопустим) и если выход слоя распознавания равен нулю.

***Блок* G_2** формирует на выходе единичный сигнал и тем самым разрешает работу слоя распознавания F_R, если на вход подан вектор X.

***Блок* G_3** проверяет критерий сходства для векторов X и C. Критерий состоит в сравнении количества единиц в векторах X, C. Количество единиц сравнивается в виде отношения с некоторым пороговым уровнем сходства h. Если порог не превышен, то сходство считается плохим, и схема сброса вырабатывает сигнал торможения для нейрона в слое распознавания F_R. Выход схемы сброса – двоичный вектор с m компонентами. Схема сброса является динамической и «помнит» свое состояние в течение одной классификации. Порог h является внешним параметром по отношению к сети и задается пользователем в интервале от 0 до 1. Чем меньше h, тем менее похожие векторы будут отнесены сетью к одному классу.

В слое сравнения каждый нейрон имеет порог, равный двум. На вход одного нейрона в слое сравнения подаются: сигнал G1 с единичным весом, одна компонента X_i с единичным весом и все выходы слоя распознавания, m компонент с вектором весов w_{ji}, где i – номер нейрона в слое сравнения. Весовые коэффициенты w_{ji} – двоичные. Таким образом, i-й нейрон активируется, т. е. $C_i = 1$, если $SUM(w_{ji}R_j) + X_i + G_1$ больше или равно 2. Это «правило 2 из 3»: для активации нейрона достаточно двух сигналов из трех.

В слое распознавания каждый нейрон в слое имеет следующие входы: один сигнал G2 с единичным весом, одна компонента $G3_j$ с большим отрицательным весом (j – номер нейрона) и n сигналов со слоя сравнения с вектором весов w_{ij} (у вектора w_{ij} всего n компонент, w_{1j}, w_{2j}, …, w_{nj}).

Нейроны слоя распознавания не содержат нелинейных элементов, но обладают следующей особенностью. Каждый нейрон в слое связан со всеми остальными нейронами этого же слоя обратными тормозящими связями, а положительной обратной связью – с самим собой. Такой способ связности называется ***латеральным торможением***. Это приводит к тому, что только один нейрон в слое распознавания может быть активирован. Между нейронами существует конкуренция, и нейрон с максимальным выходом «подавляет» все остальные нейроны в слое, выигрывая «состязание». Его выход становится равным единице, а выходы остальных нейронов – нулю.

В отличие от слоя сравнения, веса wij – вещественные, а j-й нейрон слоя распознавания принимает значение в соответствии формулой

$$R_j = (w_{ij}C) \text{ AND } G_2 \text{ AND NOT}(G3_j),$$

т. е. сигнал G2 «разрешает» работу слоя распознавания, а сигнал G3 позволяет выборочно затормозить любые нейроны в слое.

Метод art1() реализует модель нейросети ART, работающую с бинарными входными векторами, а для вещественных входов используется метод art2(). Вызов метода:

```
art1(x, dimX, dimY, f2Units = nrow(x), maxit = 100,
initFunc = «ART1_Weights», initFuncParams = c(1, 1), learnFunc = «ART1»,
  learnFuncParams = c(0.9, 0, 0), updateFunc = «ART1_Stable»,
  updateFuncParams = c(0), shufflePatterns = TRUE, ...)
```

Здесь x – матрица с тренировочными входами для нейросети;

dimX – размерность входов по x;

dimY – размерность входов по y;

f2Units – контролирует количество предполагаемых кластеров;

maxit – максимум итераций для обучения;

initFunc – используемая функция инициализации;
initFuncParams – параметры функции инициализации;
learnFunc – используемая функция обучения;
learnFuncParams – параметры функции обучения;
updateFunc – используемая функция обновления;
updateFuncParams – параметры функции обновления;
shufflePatterns – указывает, следует ли перемешивать образцы.

Обучение в сети ART протекает следующим образом: осуществляется попытка классифицировать новый вход (входной образец) в соответствии с прототипами, уже присутствующими в сети. Рассчитывают сходство между входом и всеми образцами. Наиболее близкий прототип является победителем. Если сходство между входом и победителем достаточно высоко (определяется параметром бдительности), победитель адаптируется, чтобы сделать его более похожим на входной образец. При адаптации победителя все другие прототипы остаются неизменными. Если подобие не достаточно высоко, то создается новый прототип.

Архитектура сети ART основана на более общей концепции конкурентного обучения. В сети есть два полностью подключенных слоя (в обоих направлениях), слой вход/сравнения и слой распознавания. Они распространяют активацию назад и вперед. Блоки в слое распознавания имеют боковое торможение, так что они определяют победителя, т. е. устройство, которое имеет самую высокую активацию, и подавляют активацию других блоков, так что после нескольких циклов активации победитель будет сходиться к 1, в то время как другие блоки активации будут сходиться к 0. ART стабилизирует общий механизм обучения с помощью некоторых специальных блоков.

Функция инициализации ART1_Weights по умолчанию является единственной подходящей для ART1 сетей. Она имеет два параметра, которые по умолчанию равны 1,0 и, как правило, хорошо подходят. Единственной функцией обучения, подходящей для ART1, является ART1. Разница между функциями обновления ART1_Stable и ART1_Synchronous состоит в том, что первая осуществляет обновление, пока сеть не находится в стабильном состоянии, а вторая выполняет только один шаг обновления. И функция обучения и функция обновления имеют один параметр – бдительность [3].

В текущей реализации сеть имеет двумерный вход. Матрица x содержит все (одномерные) входные шаблоны. Каждый из этих шаблонов преобразуется в двумерный шаблон с использованием параметров dimX и dimY.

Параметр f2Units задает количество единиц в слое распознавания, и тем самым задает максимальное количество кластеров, которые, как предполагается, будут присутствовать во входных образцах.

Пример 21

```
data(snnsData)
patterns <- snnsData$art1_letters.pat #1 комментарий
inputMaps <- matrixToActMapList(patterns, nrow=7) #2 комментарий
n <-c(1,2,26)
for (i in n) plotActMap(inputMaps[[i]]) #3 комментарий
model <- art1(patterns, dimX=7, dimY=5) #4 комментарий
encodeClassLabels(model$fitted.values) #5 комментарий
```

#1 комментарий

patterns – представляет собой матрицу с тренировочными входами из 26 строк (по числу букв латинского алфавита) и 35 столбцов

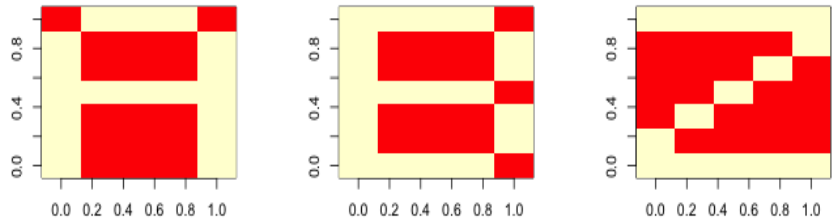
#2 комментарий

inputMaps – входные образцы, представленные в виде dimY x dimX (5x7), как показано ниже для букв A(pattern1), B(pattern2 и Z(pattern26).

```
$pattern1
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    1    1    0
[2,]    1    0    0    0    1
[3,]    1    0    0    0    1
[4,]    1    1    1    1    1
[5,]    1    0    0    0    1
[6,]    1    0    0    0    1
[7,]    1    0    0    0    1

$pattern2
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    1    1    0
[2,]    1    0    0    0    1
[3,]    1    0    0    0    1
[4,]    1    1    1    1    0
[5,]    1    0    0    0    1
[6,]    1    0    0    0    1
[7,]    1    1    1    1    0

. . .

$pattern26
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    1    1    1
[2,]    0    0    0    0    1
[3,]    0    0    0    1    0
[4,]    0    0    1    0    0
[5,]    0    1    0    0    0
[6,]    1    0    0    0    0
[7,]    1    1    1    1    1
```

#3 комментарий

Входные образцы букв A,B и Z из комментария #2, представленные графически.



#4 комментарий

```
Class: art2->rsnns
Number of inputs: 35
Number of outputs: 5
Maximal iterations: 100
Initialization function: ART2_Weights
Initialization function parameters: 0.9 2
Learning function: ART2
Learning function parameters: 0.98 10 10 0.1 0
Update function:ART2_Stable
Update function parameters: 0.98 10 10 0.1 0
Patterns are shuffled internally: TRUE
Compute error in every iteration: FALSE
```

```
Architecture Parameters:
$f2Units
[1] 5

All members of model:
 [1] «nInputs»          «maxit»
 [3] «initFunc»         «initFuncParams»
 [5] «learnFunc»        «learnFuncParams»
 [7] «updateFunc»       «updateFuncParams»
 [9] «shufflePatterns»  «computeIterativeError»
[11] «snnsObject»       «archParams»
[13] «fitted.values»    «nOutputs»

#5 комментарий
[1] 1 2 3 2 4 4 3 5 0 0 0 0 0 5 3 0 3 0 0 0 0 0 0 0 0
```

Метод art2() реализует модель нейросети ART, работающую с вещественными входными векторами. В отличие от реализации ART1 реализация ART2 не предполагает двумерный вход. Вызов метода:

```
art2(x, f2Units = 5, maxit = 100,
 initFunc = «ART2_Weights», initFuncParams = c(0.9, 2),
 learnFunc = «ART2», learnFuncParams = c(0.98, 10, 10, 0.1, 0),
 updateFunc = «ART2_Stable», updateFuncParams = c(0.98, 10, 10, 0.1, 0),
 shufflePatterns = TRUE, ...)
```

Здесь x – матрица с тренировочными входами для нейросети;

f2Units – контролирует количество предполагаемых кластеров;

maxit – максимум итераций для обучения;

initFunc – используемая функция инициализации;

initFuncParams – параметры функции инициализации;

learnFunc – используемая функция обучения;

learnFuncParams – параметры функции обучения;

updateFunc – используемая функция обновления;

updateFuncParams – параметры функции обновления;

shufflePatterns – указывает, следует ли перемешивать образцы.

Как сравнение вещественных векторов сложнее, чем сравнение бинарных векторов, так и сопоставление слоев является более сложной процедурой в сети ART2, состоящей на самом деле из трех слоев. В SNNS эта сложность отражается наличием нескольких параметров в initialization-, learning- и update-функциях.

По аналогии с осуществлением ART1 есть одна функция инициализации, одна функция обучения и две функции обновления, подходящие для ART2. Функции обучения и обновления имеют пять параметров, функция инициализации имеет два параметра [3].

Пример 22

```
data(snnsData)
patterns <- snnsData$art2_tetra_med.pat #1 комментарий
model <- art2(patterns, f2Units=5, learnFuncParams=c(0.99, 20, 20, 0.1, 0),
        updateFuncParams=c(0.99, 20, 20, 0.1, 0)) #2 комментарий
testPatterns <- snnsData$art2_tetra_high.pat #3 комментарий
predictions <- predict(model, testPatterns) #4 комментарий
library(scatterplot3d)
par(mfrow=c(2,2))
scatterplot3d(patterns, pch=encodeClassLabels(model$fitted.values)) #5 комментарий
scatterplot3d(testPatterns, pch=encodeClassLabels(predictions)) #6 комментарий
```

Содержимое комментариев #1, #3 и #4 представлено в следующей таблице.

patterns #1 комментарий				testPatterns #3 комментарий				Predictions #4 комментарий					
in1	in2	in3		in1	in2	in3		[,1]	[,2]	[,3]	[,4]	[,5]	
pattern1	0.863768	0.938372	0.968274	pattern1	0.777482	1.257200	1.158920	pattern1	0.0	0.0	0.0	0.9	0
pattern2	1.018000	2.943250	1.096770	pattern2	0.987845	3.033830	0.707253	pattern2	0.0	0.9	0.0	0.0	0
pattern3	0.992644	1.877340	2.660770	pattern3	0.930242	1.926600	2.869180	pattern3	0.9	0.0	0.0	0.0	0
pattern4	0.125113	2.143530	1.526980	pattern4	0.026284	2.106530	1.631210	pattern4	0.0	0.0	0.9	0.0	0
pattern5	0.984390	0.860563	0.930223	pattern5	0.647141	1.021900	1.038160	pattern5	0.9	0.0	0.0	0.0	0
pattern6	1.093690	2.930590	1.216960	pattern6	1.005110	3.276830	0.669638	pattern6	0.0	0.9	0.0	0.0	0
pattern7	0.969513	1.897120	2.801910	pattern7	1.200950	1.766010	2.881180	pattern7	0.9	0.0	0.0	0.0	0
pattern8	0.081274	2.122980	1.430710	pattern8	0.096206	1.919500	1.918760	pattern8	0.0	0.0	0.9	0.0	0
pattern9	0.898985	0.811716	0.916534	pattern9	0.649059	0.817931	1.061880	pattern9	0.9	0.0	0.0	0.0	0
pattern10	0.957104	2.985760	1.085610	pattern10	0.820644	3.126270	0.567751	pattern10	0.0	0.9	0.0	0.0	0
pattern11	0.984925	1.841970	2.732370	pattern11	0.991474	1.839560	2.708210	pattern11	0.9	0.0	0.0	0.0	0
pattern12	0.081504	1.991740	1.570150	pattern12	0.362500	1.992710	1.746970	pattern12	0.0	0.0	0.9	0.0	0
pattern13	0.967858	0.903610	0.812126	pattern13	0.633264	1.023570	1.026590	pattern13	0.9	0.0	0.0	0.0	0
pattern14	1.018610	3.054750	1.188380	pattern14	0.592057	3.378520	0.672310	pattern14	0.0	0.9	0.0	0.0	0
pattern15	1.020300	1.978610	2.726190	pattern15	0.790306	2.007570	2.833930	pattern15	0.9	0.0	0.0	0.0	0
pattern16	0.089630	1.850560	1.602440	pattern16	0.173392	2.158750	1.576080	pattern16	0.0	0.0	0.9	0.0	0
pattern17	0.832129	0.931441	0.782920	pattern17	0.485967	1.121280	1.302960	pattern17	0.9	0.0	0.0	0.0	0
pattern18	1.139930	3.003790	1.131630	pattern18	0.857307	3.563330	0.850719	pattern18	0.0	0.9	0.0	0.0	0
pattern19	1.126790	2.048850	2.605780	pattern19	1.082180	2.127500	2.932640	pattern19	0.9	0.0	0.0	0.0	0
pattern20	0.041768	1.821080	1.747990	pattern20	-0.055888	2.272560	1.369320	pattern20	0.0	0.0	0.9	0.0	0
pattern21	0.871007	1.031030	0.764056	pattern21	0.382580	1.092010	1.249900	pattern21	0.9	0.0	0.0	0.0	0
pattern22	1.227300	2.871620	1.022170	pattern22	0.861787	3.771090	1.045660	pattern22	0.0	0.9	0.0	0.0	0
pattern23	1.139940	2.177990	2.721250	pattern23	1.123450	2.242690	2.716910	pattern23	0.9	0.0	0.0	0.0	0
pattern24	0.008736	1.795450	1.855050	pattern24	0.152508	2.393320	1.193850	pattern24	0.0	0.0	0.9	0.0	0
pattern25	0.807035	0.913357	0.787059	pattern25	0.122752	1.055550	1.456880	pattern25	0.0	0.0	0.9	0.0	0
pattern26	1.310580	2.764300	1.114770	pattern26	0.742395	3.674530	1.068110	pattern26	0.0	0.9	0.0	0.0	0
pattern27	1.214360	2.150990	2.713780	pattern27	0.843208	2.469900	2.525770	pattern27	0.9	0.0	0.0	0.0	0
pattern28	-0.037926	1.770490	1.712140	pattern28	0.431367	2.529830	1.160860	pattern28	0.0	0.9	0.0	0.0	0
pattern29	0.814778	0.978724	0.707866	pattern29	0.302317	1.052780	1.355530	pattern29	0.9	0.0	0.0	0.0	0
pattern30	1.369040	2.848280	1.193600	pattern30	0.534551	3.816860	1.037460	pattern30	0.0	0.9	0.0	0.0	0
pattern31	1.100830	2.112150	2.842260	pattern31	0.753381	2.540900	2.790670	pattern31	0.9	0.0	0.0	0.0	0
pattern32	0.041876	1.707360	1.857240	pattern32	0.214097	2.685030	0.887740	pattern32	0.0	0.9	0.0	0.0	0
pattern33	0.793210	0.840068	0.737363	pattern33	0.354300	1.147290	1.061470	pattern33	0.9	0.0	0.0	0.0	0
pattern34	1.223710	2.925300	1.082800	pattern34	0.814633	3.718620	1.244220	pattern34	0.0	0.9	0.0	0.0	0
pattern35	1.063200	2.027400	2.865600	pattern35	0.820450	2.248320	3.002050	pattern35	0.9	0.0	0.0	0.0	0
pattern36	-0.046425	1.793470	1.931750	pattern36	0.220818	2.525710	0.990007	pattern36	0.0	0.9	0.0	0.0	0
pattern37	0.779811	0.775649	0.842429	pattern37	0.535583	0.883455	1.217730	pattern37	0.9	0.0	0.0	0.0	0
pattern38	1.208720	3.025810	1.061870	pattern38	0.885774	3.973030	1.161280	pattern38	0.0	0.9	0.0	0.0	0
pattern39	1.012500	2.162660	2.790260	pattern39	0.642553	2.143680	3.035740	pattern39	0.9	0.0	0.0	0.0	0
pattern40	-0.089387	1.768830	1.930870	pattern40	0.445469	2.794590	0.708227	pattern40	0.0	0.9	0.0	0.0	0

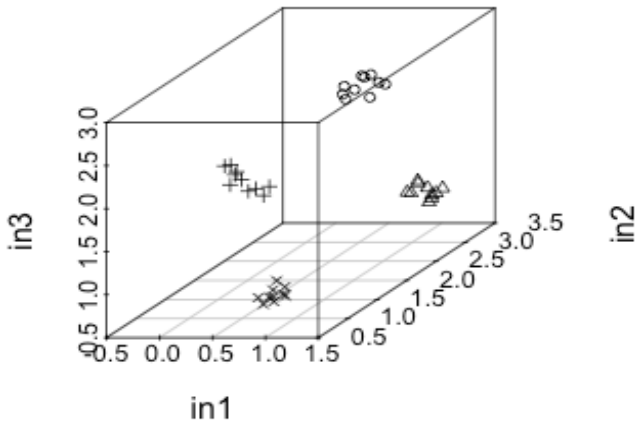
#2 комментарий

Class: art2->rsnns

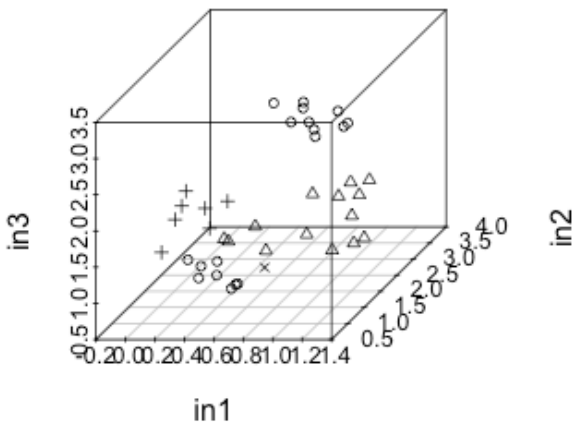
Number of inputs: 3
Number of outputs: 5
Maximal iterations: 100
Initialization function: ART2_Weights
Initialization function parameters: 0.9 2
Learning function: ART2
Learning function parameters: 0.99 20 20 0.1 0
Update function:ART2_Stable
Update function parameters: 0.99 20 20 0.1 0
Patterns are shuffled internally: TRUE
Compute error in every iteration: FALSE
Architecture Parameters:
$f2Units
[1] 5

All members of model:
[1] «nInputs» «maxit» «initFunc» «initFuncParams»
[5] «learnFunc» «learnFuncParams» «updateFunc» «updateFuncParams»
[9] «shufflePatterns» «computeIterativeError» «snnsObject» «archParams»
[13] «fitted.values» «nOutputs»

patterns #5 комментарий



testPatterns #6 комментарий



4.3. Метод artmap()

Архитектуру ARTMAP породила необходимость обеспечения у сети свойства прогнозирования. Эта архитектура состоит из двух ART-модулей, с помощью которых обеспечивается прогнозирование m-мерного выходного вектора на основе заданного n-мерного вектора входов.

На рис. 9 представлена ARTMAP система, которая включает два модуля ART, соединенных промежуточной ассоциативной памятью АП. Работа ART-модулей и ассоциативной памяти синхронизируется и управляется с помощью блока управления БУ.

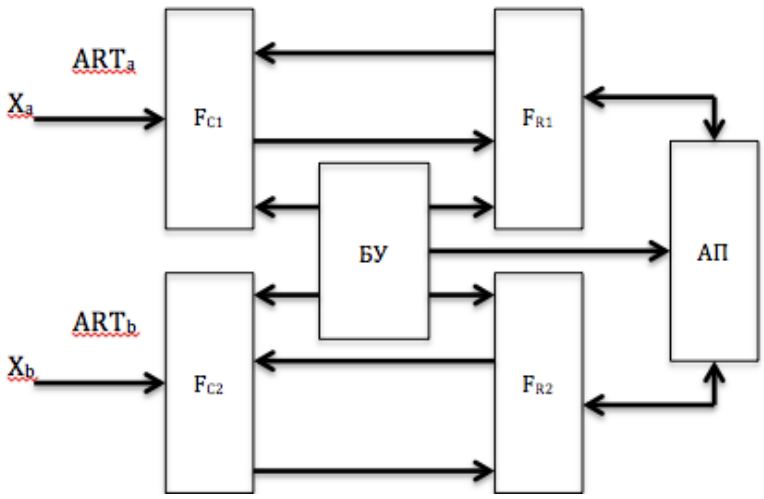


Рис. 9. Архитектура нейронной сети ARTMAP

По сравнению с традиционным многослойным перцептроном, использующим алгоритм обратного распространения ошибки, сеть ARTMAP может обучаться как с учителем, так и в режиме кластеризации, обеспечивая достаточно быстрый и точный результат.

Прогнозирующую нейронную сеть ART называют ARTMAP, так как ее преобразование из векторов C в векторы R определяется отображением, обученным на примере коррелированных пар $\{X_a(p), X_b(p)\}$, последовательно представленных векторами p = 1, 2,... Векторы $X_a(p)$ могут кодировать визуальные представления объектов, а векторы $X_b(p)$ могут кодировать их предсказываемые последствия. Степень сжатия кода в памяти является показателем возможности системы обучаться на примерах.

На некоторых этапах работы ARTMAP обучается с учителем. При таком обучении вектор входов $X_a(p)$ ассоциируется с другим вектором входов $X_b(p)$ на каждом учебном испытании. На проверочном испытании представляется новый вектор входов X_a, который ранее не был испытан. Этот вход предсказывает выходной вектор X_b. Точность работы системы оценивается путем сравнения X_b с правильным ответом. Это свойство представляет собой способность системы предсказывать правильные ответы на набор неопределенных входов X_a.

Система АРТМАР спроектирована так, чтобы совместно максимизировать обобщение и минимизировать ошибку предсказания в условиях стремительного обучения в реальном времени в ответ на произвольное упорядочивание входных образов. Она может достигать достаточно высокой точности кластеризации на небольших объемах обучающей выборки. Каждая АРТМАР-система обучается делать быстрые точные предсказания, используя сравнительно небольшие компьютерное время и число обучающих примеров, позволяя продолжать новое обучение на одной или более

базах данных, без потери прежних знаний, до тех пор пока полная емкость памяти сети не будет использована. В сети АРТМАР емкость памяти может быть выбрана произвольно большой без потери устойчивости стремительного обучения или точного обобщения.

Существенной особенностью модели АРТМАР является ее возможность совместно максимизировать обобщение и минимизировать ошибку предсказания на основе использования только локальных операций.

Сеть ARTMAP состоит из двух связанных ART-сетей и теоретически это могут быть ART1, ART2 или другие сети. Тем не менее в пакете RSNNS сеть ARTMAP реализуется только на базе ART1. Таким образом, функция artmap() будет использоваться с бинарным входом. Как пояснялось в разд. 4.2, сеть ART1 направлена на решение дилеммы стабильности/пластичности. Таким образом, преимуществом сети ARTMAP является то, что она реализует контролируемый механизм обучения, который гарантирует стабильность.

Вызов метода:

```
artmap(x, nInputsTrain, nInputsTargets, nUnitsRecLayerTrain,
 nUnitsRecLayerTargets, maxit = 1, nRowInputsTrain = 1,
 nRowInputsTargets = 1, nRowUnitsRecLayerTrain = 1,
 nRowUnitsRecLayerTargets = 1, initFunc = «ARTMAP_Weights»,
 initFuncParams = c(1, 1, 1, 1, 0), learnFunc = «ARTMAP»,
 learnFuncParams = c(0.8, 1, 1, 0, 0), updateFunc = «ARTMAP_Stable»,
 updateFuncParams = c(0.8, 1, 1, 0, 0), shufflePatterns = TRUE, ...)
```

Здесь x – матрица с тренировочными входами и целями для нейросети;

nInputsTrain – число столбцов входной обучающей матрицы;

nInputsTargets – число столбцов целевой матрицы;

nUnitsRecLayerTrain – число нейронов в слое распознавания обучающих данных сети ART;

nUnitsRecLayerTargets – число нейронов в слое распознавания целевых данных сети ART;

maxit – максимум выполняемых итераций;

nRowInputsTrain – число строк входных обучающих блоков, которые должны быть созданы (только для визуализации сети в оригинальном программном обеспечении SNNS);

nRowInputsTargets – то же, но только для входных целевых блоков;

nRowUnitsRecLayerTrain – то же, но только для слоя распознавания обучающих данных сети ART;

nRowUnitsRecLayerTargets – то же, но только для слоя распознавания целевых данных сети ART;

initFunc – используемая функция инициализации;

initFuncParams – параметры функции инициализации;

learnFunc – используемая функция обучения;

learnFuncParams – параметры функции обучения;
updateFunc – используемая функция обновления;
updateFuncParams – параметры функции обновления;
shufflePatterns – указывает, следует ли перемешивать образцы.

На вход первой ART_a сети подаются обучающие входные данные, а на вход второй сети ART_b – целевые значения, сигналы учителя. Эти две сети часто называют сетью передачи обучающих данных и сетью передачи целевых данных.

По аналогии с реализациями ART1 и ART2 для ARTMAP есть одна функция инициализации, одна функция обучения и две функции обновления. Параметры в основном такие же, как и для ART1, но только для двух сетей. Функция обучения и функция обновления имеют по три параметра: параметры бдительности для двух ART1-сетей и дополнительный параметр для контроля бдительности сброса между сетями. Функция инициализации имеет четыре параметра, по два для каждой ART1-сети.

Подробное описание теории и параметров можно получить в документации по SNNS и другой литературе.

Пример 23

```
data(snnsData)
trainData <- snnsData$artmap_train.pat #1 комментарий
testData <- snnsData$artmap_test.pat #2 комментарий
model <- artmap(trainData, nInputsTrain=70, nInputsTargets=5,
          nUnitsRecLayerTrain=50, nUnitsRecLayerTargets=26)
model$fitted.values
predict(model, testData) #3 комментарий

#1 комментарий
trainData – матрица[26x75] двоичных векторов patternI(in1,in2,…,in75)
#2 комментарий
testData – матрица[26x75] двоичных векторов patternI(in1,in2,…,in75)
#3 комментарий
```

Сравнение результатов обучения model$fitted.values и результатов предсказания показывает точность отработки ARTMAP сети model.

4.4. Функция som()

Сети Кохонена относятся к самоорганизующимся нейронным сетям SOM (self-organizing map). Самоорганизующаяся сеть позволяет выявлять кластеры входных векторов, обладающих некоторыми общими свойствами. С помощью сетей SOM производится кластеризация объектов, описываемых количественными характеристиками.

Сеть (слой) Кохонена (рис. 10) – это однослойная сеть, построенная из нейронов типа WTA (Winner Takes All – победитель получает все).

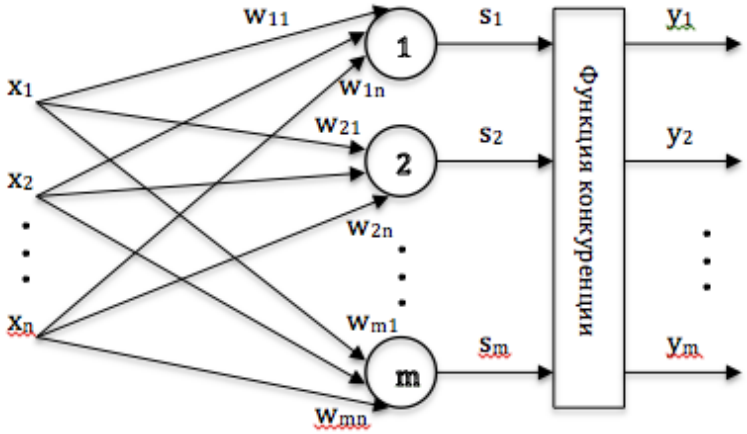


Рис. 10. Сеть Кохонена (SOM)

Каждый нейрон сети соединен со всеми компонентами n-мерного входного вектора $X = (x_1, x_2, \ldots, x_n)$. Входной вектор – это описание одного из объектов, подлежащих кластеризации. Количество нейронов совпадает с количеством кластеров, которое должна выделить сеть. В качестве нейронов сети Кохонена применяются линейные взвешенные сумматоры $s_j = b_j + SUM(w_{ij}x_i)$, где j – номер нейрона, i – номер входа, s_j – выход адаптивного сумматора, w_{ij} – вес i-го входа j-го нейрона, b_j – порог. С выходов сумматоров сигнал поступает на функцию конкуренции, работающую по правилу «победитель получает все». Функция конкуренции находит выход сумматора с максимальным значением выхода. Пусть k – номер такого сумматора. Тогда на выходе сети формируется выходной сигнал $y_k = 1$, остальные выходные сигналы равны нулю. Если максимум достигается одновременно на выходах нескольких сумматоров, то выходной сигнал, равный единице, соответствует одному из них, например первому.

Обучение сети Кохонена представляет собой подбор значений весов, минимизирующих ошибки от замены близких в смысле используемой метрики входных векторов вектором весов. Такой подход называется ***векторным квантованием*** и состоит в компактном представлении многомерных входных векторов с помощью ограниченного набора опорных векторов меньшей размерности, образующих кодовую таблицу. В случае сети Кохонена входные векторы кодируются номерами нейронов победителей (номерами кластеров). Таким образом, все векторы из некоторой области входного пространства заменяются одним и тем же опорным вектором, являющимся их ближайшим соседом. При использовании евклидова расстояния входное пространство разбивается на многогранники.

Функция som() создает и обучает самоорганизующуюся нейронную сеть SOM. Вызов функции:

```
som(x, mapX = 16, mapY = 16, maxit = 100,
    initFuncParams = c(1, -1), learnFuncParams = c(0.5, mapX/2, 0.8, 0.8,
    mapX), updateFuncParams = c(0, 0, 1), shufflePatterns = TRUE,
    calculateMap = TRUE, calculateActMaps = FALSE,
    calculateSpanningTree = FALSE, saveWinnersPerPattern = FALSE,
    targets = NULL)
```

Здесь x – матрица с обучающими входами для сети;

mapX – x-размерность SOM;

mapY – y-размерность SOM;

maxit – максимальное число итераций при обучении;

initFuncParams – параметры функции инициализации;

learnFuncParams – параметры функции обучения;

updateFuncParams – параметры функции обновления;

shufflePatterns – указывает, должны ли образцы перемешиваться;

calculateMap – указывает, должен ли вычисляться SOM;

calculateActMaps – указывает, должна ли расчитываться карта активации;

calculateSpanningTree – указывает, нужно ли применять алгоритм ядра SNNS для генерации покрывающего дерева;

saveWinnersPerPattern – указывает, должен ли сохраняться список с победителями для каждого образца;

targets – целевые классы для образцов.

В зависимости от включенных флагов SOM порождает некоторые специальные элементы:

- map – карта SOM, где для каждого нейрона указано количество образцов, для которых этот нейрон становится победителем;
- componentMaps – карта, показывающая для каждого входа, где на ней этот компонент приводит к высокой активации;
- actMaps – список, содержащий для каждого образца его карту активации, т. е. все единицы активации. Этот список может быть очень длинным, так что обычно он не сохраняется;
- winnersPerPattern – вектор, в котором для каждого образца дается номер блока победителя. Он является промежуточным результатом и обычно не сохраняется;
- labeledUnits – матрица. Если параметр targets задается, данная матрица содержит для каждого блока (строки) и каждого целевого класса (столбца) количество образцов класса, где блок выиграл. Из labeledUnits может быть вычислена labeledMap, например путем голосования классовых меток для выбора окончательной метки;
- labeledMap – размеченный SOM, который вычисляется из labeledUnits с использованием функции decodeClassLabels();

• spanningTree – результат оригинальной SNNS-функции по вычислению карты. Для каждой единицы присутствует последний образец, где этот блок выиграл. Поскольку другие результаты более информативны, покрывающее дерево интересно, если только другие функции вычисляются слишком медленно или если требуется оригинал реализации SNNS.

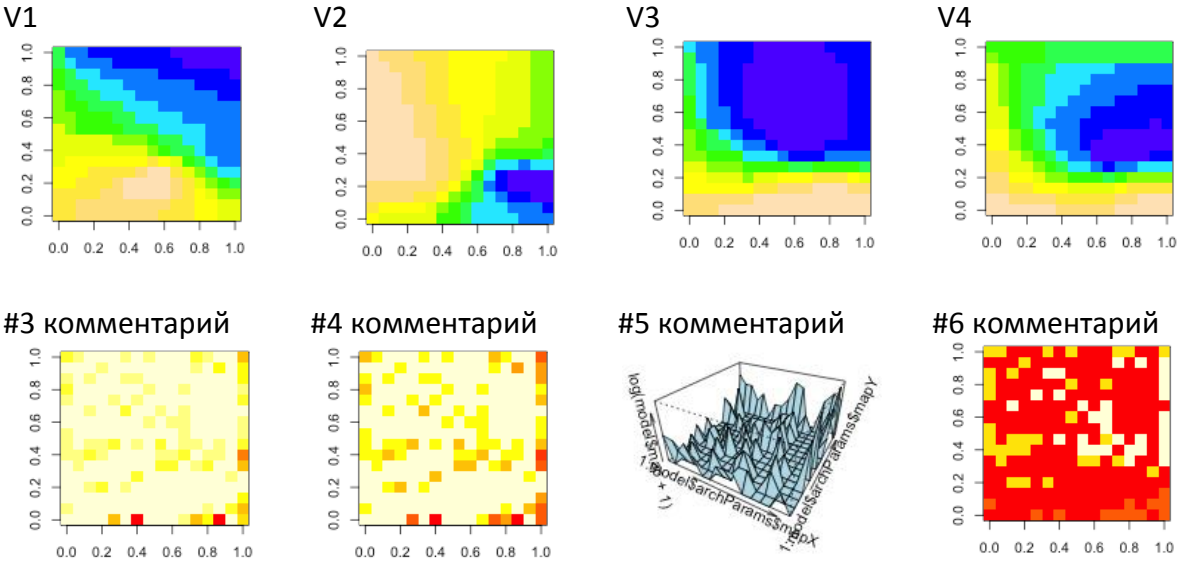
Пример 24

```
data(iris)
inputs <- normalizeData(iris[,1:4], «norm») #1 комментарий
model <- som(inputs, mapX=16, mapY=16, maxit=500,
        calculateActMaps=TRUE, targets=iris[,5])
par(mfrow=c(3,3))
for(i in 1:ncol(inputs))
plotActMap(model$componentMaps[[i]],col=rev(topo.colors(12))) #2 комментарий
plotActMap(model$map, col=rev(heat.colors(12))) #3 комментарий
plotActMap(log(model$map+1), col=rev(heat.colors(12))) #4 комментарий
persp(1:model$archParams$mapX, 1:model$archParams$mapY,
        log(model$map+1), theta = 30, phi = 30, expand = 0.5,
        col = «lightblue») #5 комментарий
plotActMap(model$labeledMap) #6 комментарий
model$componentMaps #7 комментарий
model$labeledUnits #8 комментарий
model$map #9 комментарий
names(model) #10 комментарий
```

#1 комментарий

inputs – матрица [150x4] нормализованных вещественных векторов V1,V2,V3,V4 – это знаменитый набор данных, который дает измерения в сантиметрах длины и ширины чашелистника и длины и ширины лепестков ирисов на 50 цветов от каждого из 3 видов ириса

#2 комментарий



#7 комментарий

Четыре матрицы 16x16 со значениями, соответствующими картам V1–V4 из комментария #2

#8 комментарий

```
       setosa versicolor virginica
  [1,]      0          2         0
  [2,]      0          1         0
. . .
[256,]      4          0         0
```

#9 комментарий

```
model$map
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16]
 [1,]    2    1    0    0    0    1    0    1    0     0     0     2     1     0     0     5
 [2,]    0    0    0    1    0    0    0    0    0     0     0     1     0     3     0     3
 [3,]    1    0    1    0    0    1    1    0    0     0     0     0     0     0     0     3
 [4,]    2    0    0    0    0    0    0    1    0     0     1     0     0     0     0     1
 [5,]    1    0    0    0    1    0    1    0    0     0     0     0     0     0     0     2
 [6,]    0    0    1    0    0    2    0    0    1     0     1     0     0     0     0     0
 [7,]    0    0    0    0    0    0    0    1    0     1     1     0     0     0     0     1
 [8,]    1    0    0    0    0    0    0    0    0     0     1     0     0     0     0     1
 [9,]    1    0    1    1    2    0    1    0    2     0     1     1     0     1     0     2
[10,]    1    1    2    1    0    0    0    0    1     1     2     0     1     1     0     6
[11,]    0    1    0    0    0    0    0    0    2     2     0     1     2     0     0     5
[12,]    0    0    0    0    0    0    2    0    0     0     0     0     0     0     1     0
[13,]    0    0    1    2    0    1    0    0    0     0     0     0     0     0     0     0
[14,]    0    0    0    0    0    0    0    0    0     0     0     0     0     0     0     3
[15,]    1    0    0    0    0    0    0    0    0     0     0     0     1     0     2     5
[16,]    0    0    0    0    5    0   10    0    0     0     1     5     3    10     0     4
```

#10 комментарий

```
> names(model)
 [1] «nInputs»          «maxit»          «initFunc»          «initFuncParams»
 [5] «learnFunc»        «learnFuncParams»    «updateFunc»
«updateFuncParams»
 [9] «shufflePatterns»    «computeIterativeError» «snnsObject»        «archParams»
[13] «nOutputs»         «map»            «labeledUnits»      «labeledMap»
[17] «actMaps»          «componentMaps»
```

4.5. Метод decodeClassLabels()

```
> decodeClassLabels(c(«r»,»b»,»b»,»r», «g», «g»))
     b g r
[1,] 0 0 1
[2,] 1 0 0
[3,] 1 0 0
[4,] 0 0 1
```

```
[5,] 0 1 0
[6,] 0 1 0
> decodeClassLabels(c(1,3,2,3))
   1 2 3
[1,] 1 0 0
[2,] 0 0 1
[3,] 0 1 0
[4,] 0 0 1
```

4.6. Функция MLP()

Эта функция создает многослойный персептрон (multilayer perceptron) MLP и обучает его. MLP является полносвязной нейронной сетью прямого распространения и, вероятно, наиболее распространенной в использовании сетевой архитектурой. Обучение, как правило, осуществляется на базе ошибки обратного распространения.

Вызов функции:

```
mlp(x, y, size = c(5), maxit = 100,
    initFunc = «Randomize_Weights», initFuncParams = c(-0.3, 0.3),
    learnFunc = «Std_Backpropagation», learnFuncParams = c(0.2, 0),
    updateFunc = «Topological_Order», updateFuncParams = c(0),
    hiddenActFunc = «Act_Logistic», shufflePatterns = TRUE,
    linOut = FALSE, inputsTest = NULL, targetsTest = NULL,
    pruneFunc = NULL, pruneFuncParams = NULL)
```

Здесь x – матрица с обучающими входами для сети;

y – соответствующие целевые значения;

size – число нейронов в скрытом слое (скрытых слоях);

initFunc – функция инициализации;

initFuncParams – параметры функции инициализации;

learnFunc – функция обучения;

learnFuncParams – параметры функции обучения;

updateFunc – функция обновления;

updateFuncParams – параметры функции обновления;

hiddenActFunc – функция активации для всех скрытых слоев;

shufflePatterns – указывает, должны ли образцы перемешиваться;

linOut – устанавливает функцию активации выходных нейронов линейной или логической;

inputsTest – матрица с входами для тестирования сети;

targetsTest – соответствующие цели для тестируемого входа;

pruneFunc – функция обрезки (сокращения);

pruneFuncParams – параметры функции обрезки. В отличие от других функций они должны быть приведены в поименованном списке.

Есть много различных функций обучения, присутствующих в SNNS, которые могут быть использованы вместе с этой функцией, например Std_Backpropagation, BackpropBatch, BackpropChunk, BackpropMomentum, BackpropWeightDecay, Rprop, QuickProp, SCG (scaled conjugate gradient) и др.

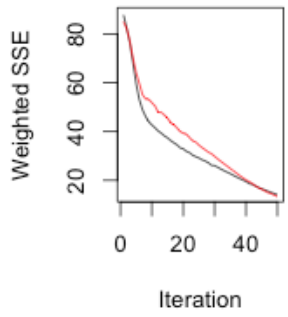
Например, у функций обучения Std_Backpropagation и BackpropBatch есть два параметра – скорость обучения и максимальное выходное различие. Скорость обучения, как правило, значение между 0,1 и 1. Она определяет градиент уменьшения ширины шага. Максимальная разница определяет, какое различие между выходным и целевым значением рассматривается как нулевая ошибка. Этот параметр используется для предотвращения перетренированности. Для получения полного списка параметров всех функций обучения следует воспользоваться руководством [5].

Значения по умолчанию, установленные для инициализации и обновлений функций, как правило, не должны быть изменены.

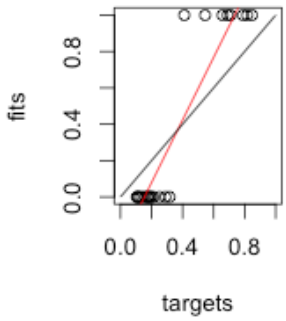
Пример 24

```
data(iris)
iris <- iris[sample(1:nrow(iris),length(1:nrow(iris))),1:ncol(iris)]
irisValues <- iris[,1:4]
irisTargets <- decodeClassLabels(iris[,5])
iris <- splitForTrainingAndTest(irisValues, irisTargets, ratio=0.15)
iris <- normTrainingAndTestSet(iris)
model <- mlp(iris$inputsTrain, iris$targetsTrain, size=5, learnFuncParams=c(0.1),
        maxit=50, inputsTest=iris$inputsTest, targetsTest=iris$targetsTest)
summary(model)
model
weightMatrix(model)
extractNetInfo(model)
par(mfrow=c(2,2))
plotIterativeError(model) #1 комментарий
predictions <- predict(model,iris$inputsTest)
plotRegressionError(predictions[,2], iris$targetsTest[,2]) #2 комментарий
confusionMatrix(iris$targetsTrain,fitted.values(model))
confusionMatrix(iris$targetsTest,predictions)
plotROC(fitted.values(model)[,2], iris$targetsTrain[,2]) #3 комментарий
plotROC(predictions[,2], iris$targetsTest[,2]) #4 комментарий
```

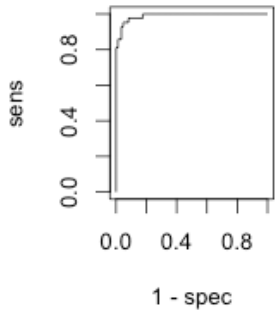
#1 комментарий



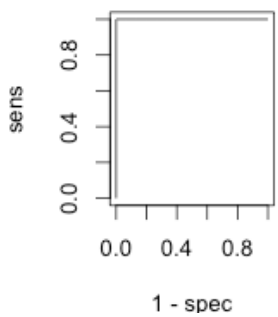
#2 комментарий



#3 комментарий



#4 комментарий



4.7. Функция rbf()

Применение нейронной сети RBF (radial basis function) аналогично сети MLP. Идея сети радиальной базисной функции возникла из теории интерполяции функций. RBF выполняет линейную комбинацию n базисных функций, которые радиально симметричны центральной точке. Радиально-симметричные функции – специальный класс функций. Их характерное свойство заключается в том, что отклик функции монотонно убывает (возрастает) с удалением от центральной точки. Типичный пример такой функции – функция Гаусса: $h(X) = \exp(-\|X - C\|^2/r^2)$.

Традиционно термин RBF-сети ассоциируется с радиально-симметричными функциями в однослойных сетях, имеющих структуру, представленную на рис. 11.

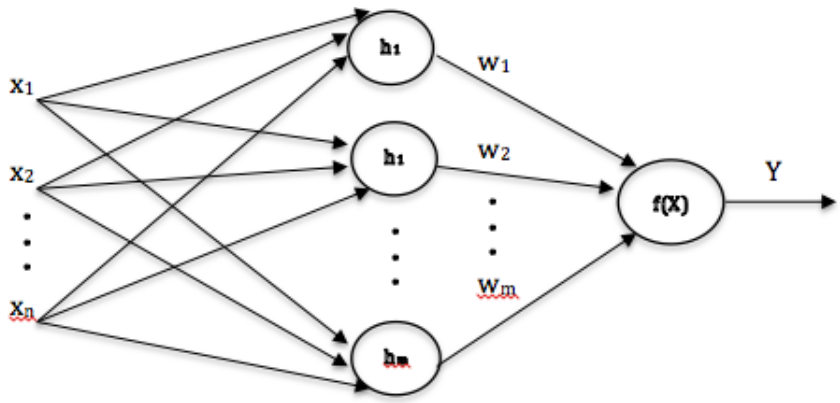


Рис. 11. Структура RBF-сети

Каждый из n компонентов входного вектора $X = (x_1, x_2, \ldots, x_n)$ подается на вход m базисных функций $h_i(X)$, и их выходы линейно суммируются с весами w_j, таким образом, выход RBF-сети является линейной комбинацией некоторого набора базисных функций: $f(X) = SUM(w_j h_j(X))$.

Вызов функции:

```
rbf(x, y, size = c(5), maxit = 100,
    initFunc = «RBF_Weights», initFuncParams = c(0, 1, 0, 0.02, 0.04),
    learnFunc = «RadialBasisLearning», learnFuncParams = c(1e-05, 0, 1e-05,
    0.1, 0.8), updateFunc = «Topological_Order», updateFuncParams = c(0),
    shufflePatterns = TRUE, linOut = TRUE, inputsTest = NULL,
    targetsTest = NULL)
```

Здесь x – матрица с обучающими входами для сети;

y – соответствующие целевые значения;

size – число нейронов в скрытом слое (скрытых слоях);

maxit – максимальное число итераций при обучении;

initFunc – функция инициализации;

initFuncParams – параметры функции инициализации;

learnFunc – функция обучения;

learnFuncParams – параметры функции обучения;

updateFunc – функция обновления;

updateFuncParams – параметры функции обновления;

shufflePatterns – указывает, должны ли образцы перемешиваться;

linOut – устанавливает функцию активации выходных нейронов линейной или логической;

inputsTest – матрица с входами для тестирования сети;

targetsTest – соответствующие цели для тестируемого входа.

Сети RBF имеют ряд преимуществ перед сетями MLP. Во-первых, как уже сказано, они моделируют произвольную нелинейную функцию с помощью всего одного промежуточного слоя и тем самым избавляют нас от необходимости решать вопрос о числе слоев. Во-вторых, параметры линейной комбинации в выходном слое можно полностью оптимизировать с помощью хорошо известных методов линейного моделирования, которые работают быстро и не испытывают трудностей с локальными минимумами, так мешающими при обучении MLP. Поэтому сеть-RBF обучается очень быстро (на порядок быстрее MLP).

С другой стороны, до того как применять линейную оптимизацию в выходном слое сети RBF, необходимо определить число радиальных элементов, положение их центров и величины отклонений. Соответствующие алгоритмы, хотя и работают быстрее алгоритмов обучения MLP, в меньшей степени пригодны для отыскания субоптимальных решений.

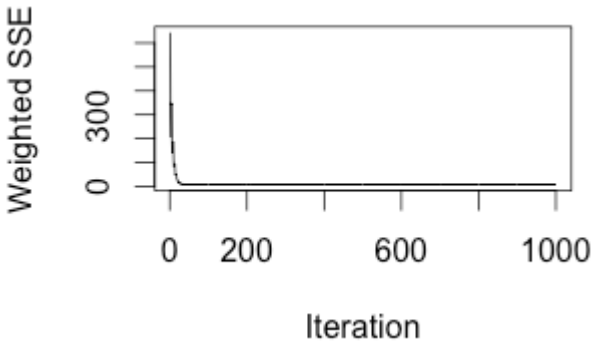
Пример 25

```
inputs <- as.matrix(seq(0,10,0.1)) #1
outputs <- as.matrix(sin(inputs) + runif(inputs*0.2))
outputs <- normalizeData(outputs, «0_1») #2
model <- rbf(inputs, outputs, size=40, maxit=1000,
                initFuncParams=c(0, 1, 0, 0.01, 0.01),
                learnFuncParams=c(1e-8, 0, 1e-8, 0.1, 0.8), linOut=TRUE)
par(mfrow=c(2,1))
plotIterativeError(model) #3
plot(inputs, outputs)
lines(inputs, fitted(model), col=«green») #4
```

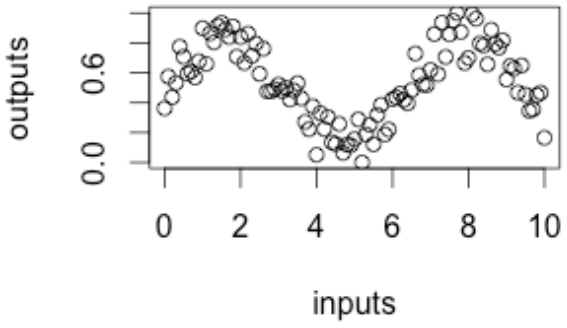
#1 inputs – вектор V1(1:101) значений от 0.0 до 10.0 с интервалом 0.1

#2 outputs – вектор V1(1:101) нормализованных значений от 0.0 до 1.0

#3



#4



4.8. Функция dlvq()

Сети DLVQ (dynamic learning vector quantization) основаны на динамически обучающемся векторном квантовании и представляют собой слой Кохонена, обучающийся с учителем. Для построения сети задается количество кластеров (нейронов) n, количество классов m (n>m) и принадлежность каждого кластера определенному классу. В процессе обучения сети веса нейронов настраиваются с учетом принадлежности обучающих примеров и кластеров к одному классу. Обученная DLVQ сеть производит кластеризацию входных векторов с учетом классов. Простейший алгоритм обучения на k-цикле работает следующим образом.

Шаг 1. Для очередного вектора X обучающей выборки находится нейрон с номером i, для которого евклидово расстояние между X и вектором его весов $w^{(k)}_i$ минимально.

Шаг 2. Корректируется вектор весов нейрона-победителя. Если $w^{(k)}_i$ и X принадлежат одному классу, то $w^{(k+1)}_i = w^{(k)}_i + r^{(k)}[X - w^{(k)}_i]$, а если различным классам, то $w^{(k+1)}_i = w^{(k)}_i - r^{(k)}[X - w^{(k)}_i]$, где $r^{(k)}$ – коэффициент скорости обучения. Веса других нейронов не изменяются.

Шаг 3. Выбирается следующий обучающий вектор и производится переход на шаг 1.

Шаги 1–3 повторяются до тех пор, количество правильно классифицированных векторов перестанет увеличиваться.

Если входной вектор классифицируется сетью правильно, то соответствующий вектор весов сдвигается в сторону входного вектора. Если входной вектор классифицируется неправильно, то соответствующий вектор весов сдвигается в противоположную сторону от входного вектора. Коэффициент скорости обучения $0 < r^{(k)} < 1$ должен монотонно уменьшаться с ростом номера цикла обучения. Но даже начальное значение $r^{(k)}$ должно быть достаточно малым, например меньше 0,1.

Возможна реализация DLVQ-сетей с двумя слоями: конкурирующим и линейным. Конкурирующий слой выполняет кластеризацию векторов, а линейный слой соотносит кластеры с целевыми классами, заданными пользователем. Как в конкурирующем, так и в линейном слое на кластер или целевой класс приходится один нейрон. Таким образом, если конкурирующий слой способен поддержать до n кластеров, то эти кластеры, в свою очередь, могут быть соотнесены с m целевыми классами, причем m не превышает n.

Например, предположим, что группа k из трех нейронов конкурирующего слоя определяют три кластера, которые принадлежат к одному целевому классу p линейного слоя. Тогда выходы конкурирующих нейронов группы k будут передаваться в линейный слой на нейрон p с весами, равными 1, а на остальные нейроны – выходы с весами, равными 0. Таким образом, нейрон p возвращает 1, если любой из трех нейронов группы k конкурирующего слоя выигрывает конкуренцию.

Для исследования сетей DLVQ в пакете RSNNS реализована функция dlvq() со следующими параметрами:

```
dlvq(x, y, initFunc = «DLVQ_Weights»,
    initFuncParams = c(1, -1), learnFunc = «Dynamic_LVQ»,
    learnFuncParams = c(0.03, 0.03, 10), updateFunc = «Dynamic_LVQ»,
    updateFuncParams = c(0), shufflePatterns = TRUE).
```

Здесь x – матрица с обучающими входами для сети;
y – соответствующие целевые значения;
initFunc – функция инициализации;
initFuncParams – параметры функции инициализации;
learnFunc – функция обучения;
learnFuncParams – параметры функции обучения;
updateFunc – функция обновления;
updateFuncParams – параметры функции обновления;
shufflePatterns – указывает, должны ли образцы перемешиваться.

Пример 26

```
data(snnsData)
dataset <- snnsData$dlvq_ziff_100.pat
inputs <- dataset[,inputColumns(dataset)]
outputs <- dataset[,outputColumns(dataset)]
model <- dlvq(inputs, outputs)
fitted(model) == outputs
mean(fitted(model) – outputs)
```

4.9. Функция jordan()

Сеть Джордана – вид рекуррентных нейронных сетей, которые получаются из многослойного перцептрона, если на его вход наряду с входным вектором подать выходной вектор с задержкой на один или несколько тактов. Ее можно рассматривать как сеть прямого распространения с дополнительными нейронами контекста во входном слое.

Эти контекстные нейроны принимают вход от самих себя и из выходных нейронов. Контекстные нейроны сохраняют текущее состояние сети. В сети Джордана количество контекстных и выходных нейронов должно быть одинаковым.

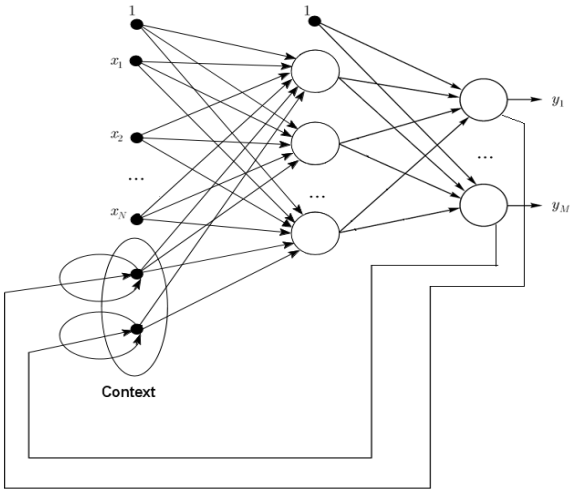


Рис. 12. Нейронная сеть Джордана

Вызов функции:

```
jordan(x, y, size = c(5), maxit = 100,
    initFunc = «JE_Weights», initFuncParams = c(1, -1, 0.3, 1, 0.5),
    learnFunc = «JE_BP», learnFuncParams = c(0.2), updateFunc = «JE_Order»,
    updateFuncParams = c(0), shufflePatterns = FALSE, linOut = TRUE,
    inputsTest = NULL, targetsTest = NULL).
```

Здесь x – матрица с обучающими входами для сети;
y – соответствующие целевые значения;
size – число нейронов в скрытом слое;
maxit – максимальное число итераций при обучении;
initFunc – функция инициализации;
initFuncParams – параметры функции инициализации;
learnFunc – функция обучения;
learnFuncParams – параметры функции обучения;
updateFunc – функция обновления;

updateFuncParams – параметры функции обновления;

shufflePatterns – указывает, должны ли образцы перемешиваться;

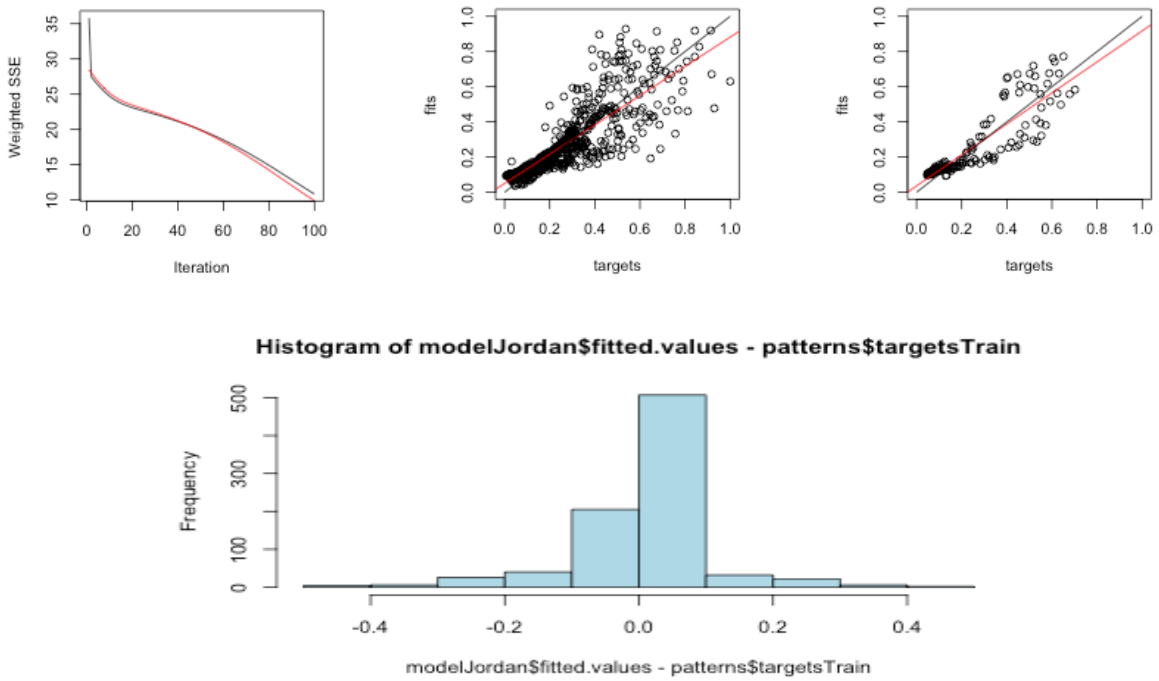
linOut – устанавливает функцию активации выходных нейронов линейной или логической;

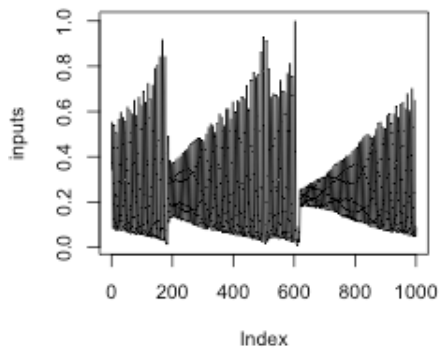
inputsTest – матрица с входами для тестирования сети;

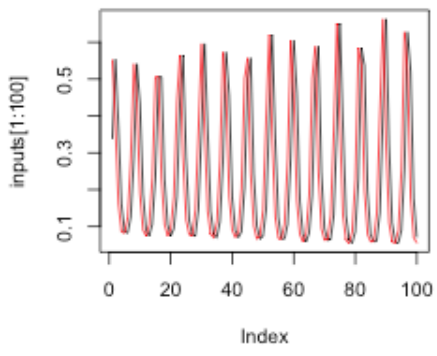
targetsTest – соответствующие цели для тестируемого входа.

Пример 27

```
data(snnsData)
inputs <- snnsData$laser_1000.pat[,inputColumns(snnsData$laser_1000.pat)]
outputs <- snnsData$laser_1000.pat[,outputColumns(snnsData$laser_1000.pat)]
patterns <- splitForTrainingAndTest(inputs, outputs, ratio=0.15)
modelJordan <- jordan(patterns$inputsTrain, patterns$targetsTrain,
                size=c(8), learnFuncParams=c(0.1), maxit=100,
                inputsTest=patterns$inputsTest,
                targetsTest=patterns$targetsTest, linOut=FALSE)
names(modelJordan)
par(mfrow=c(3,3))
plotIterativeError(modelJordan)
plotRegressionError(patterns$targetsTrain, modelJordan$fitted.values)
plotRegressionError(patterns$targetsTest, modelJordan$fittedTestValues)
hist(modelJordan$fitted.values – patterns$targetsTrain, col=«lightblue»)
plot(inputs, type=«l»)
plot(inputs[1:100], type=«l»)
lines(outputs[1:100], col=«red»)
lines(modelJordan$fitted.values[1:100], col=«green»)
```







4.10. Функция elman()

Нейронная сеть Элмана – один из видов рекуррентной сети, которая, так же как и сеть Джордана, получается из многослойного перцептрона введением обратных связей. Разница между сетью Элмана и Джордана в том, что в сети Элмана на вход контекстных нейронов подаются выходные значения не от выходных нейронов, а от скрытых. Кроме того, в контекстных нейронах нет никакой прямой обратной связи.

В сети Элмана число контекстных и скрытых нейронов должно быть одинаковым. Главное преимущество сетей Элмана состоит в том, что число контекстных нейронов определяется не размерностью выхода (как в сети Джордана), а количеством скрытых нейронов, что делает ее более гибкой. Можно легко добавить или убрать скрытые нейроны, в отличие от количества выходов.

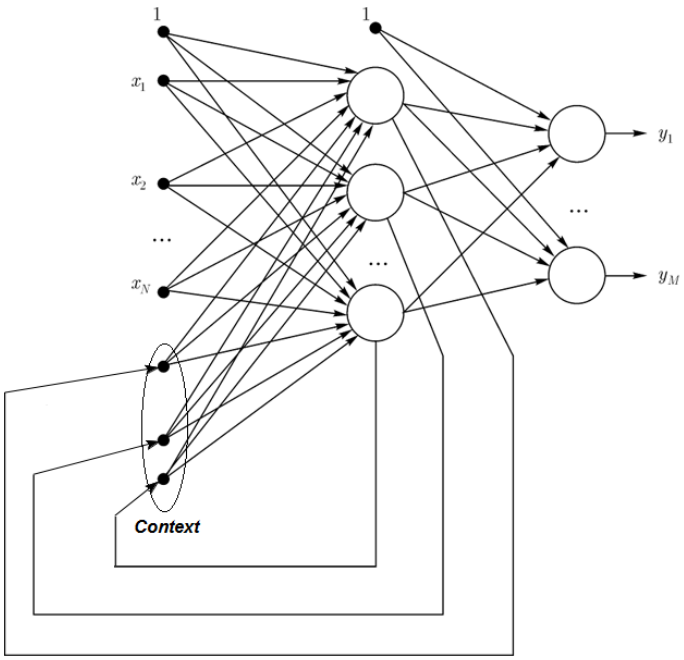


Рис. 13. Нейронная сеть Элмана

Это позволяет учесть предысторию наблюдаемых процессов и накопить информацию для выработки правильной стратегии управления. Эти сети могут применяться в системах управления движущимися объектами, так как их главной особенностью является запоминание последовательностей. Вызов функции Элмана:

```
elman(x, y, size = c(5), maxit = 100,
    initFunc = «JE_Weights», initFuncParams = c(1, -1, 0.3, 1, 0.5),
    learnFunc = «JE_BP», learnFuncParams = c(0.2), updateFunc = «JE_Order»,
    updateFuncParams = c(0), shufflePatterns = FALSE, linOut = TRUE,
    outContext = FALSE, inputsTest = NULL, targetsTest = NULL).
```

Здесь x – матрица с обучающими входами для сети;

y – соответствующие целевые значения;

size – число нейронов в скрытом слое;

maxit – максимальное число итераций при обучении;

initFunc – функция инициализации;

initFuncParams – параметры функции инициализации;

learnFunc – функция обучения;

learnFuncParams – параметры функции обучения;

updateFunc – функция обновления;

updateFuncParams – параметры функции обновления;

shufflePatterns – указывает, должны ли образцы перемешиваться;

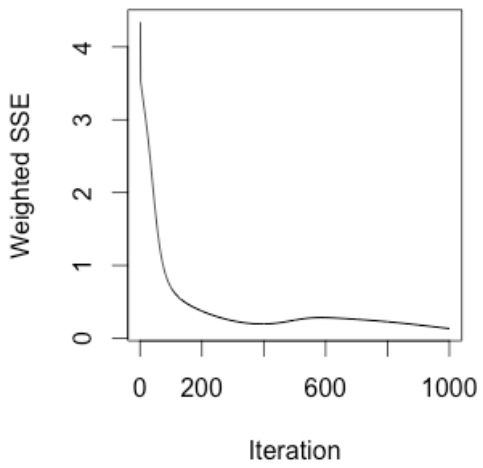
linOut – устанавливает функцию активации выходных нейронов линейной или логической;

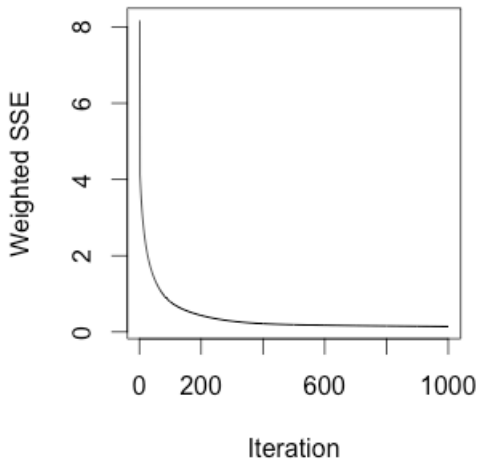
inputsTest – матрица с входами для тестирования сети;

targetsTest – соответствующие цели для тестируемого входа.

Пример 28

```
data(snnsData)
inputs <- snnsData$eight_016.pat[,inputColumns(snnsData$eight_016.pat)]
outputs <- snnsData$eight_016.pat[,outputColumns(snnsData$eight_016.pat)]
par(mfrow=c(1,2))
modelElman <- elman(inputs, outputs, size=8, learnFuncParams=c(0.1), maxit=1000)
modelElman
modelJordan <- jordan(inputs, outputs, size=8, learnFuncParams=c(0.1), maxit=1000)
modelJordan
plotIterativeError(modelElman)
plotIterativeError(modelJordan)
summary(modelElman)
summary(modelJordan)
```





На основе сети Элмана строится нейронная сеть класса RAAM (рекурсивная автоассоциативная память), которая по своей структуре повторяет элмановскую. RAAM – это двойная сеть Элмана вида 2N-N-2N, которую используют для сжатия и шифрования информации. На вход сети подается битовый сигнал из 2N битов.

Обычно сеть имеет размер 20-10-20, первые 10 бит называются «левыми», следующие 10 бит – «правыми». В самом начале на левую матрицу подается нулевой вектор бит (0000000000), а на правую – код символа или предложения (например, 0010000000 = “A”). То же самое подается на выходные матрицы. Методом обратного распространения ошибки сеть обучается. Затем 10 бит из скрытого слоя передаются на левую входную матрицу, а на правую поступает очередной символ. В процессе такого рекурсивного обучения информация сжимается и шифруется.

5. ПАКЕТ NeuralNetTools

Пакет предоставляет средства визуализации и анализа, чтобы помочь в интерпретации нейросетевых моделей. Функции пакета позволяют строить графики, давать количественную оценку важности входных переменных, проводить анализ чувствительности, а также получать простой список весов модели.

5.1. Функция garson()

Функция garson() дает количественную оценку относительной важности входных переменных нейросети, используя алгоритм Гарсона. Вызов функции:

```
garson(mod_in, struct, bar_plot = TRUE, x_lab = NULL,
    y_lab = NULL, wts_only = FALSE, ...)
```

Здесь mod_in – входной объект, для которого формируются оценки. Входной объект может относиться к классам numeric, nnet, mpl или nn;

struct – (только для класса numeric!) числовой вектор, равный по длине числу слоев в сети. Каждое число указывает количество узлов в каждом слое, начиная с входного и заканчивая выходом. Может быть включено произвольное число скрытых слоев;

bar_plot – указывает, что возвращается объект ggplot, в противном случае возвращаются числовые значения;

x_lab – строка альтернативных имен, которые будут использоваться для независимых переменных на рисунке, по умолчанию берется из mod_in;

y_lab – строка альтернативных имен, которые будут использоваться для выходных переменных на рисунке, по умолчанию берется из входной модели (объекта);

wts_only – указывает на вызов функции neuralweights(), по умолчанию FALSE;

... – аргументы (параметры), передаваемые другим методам (функциям).

Веса, которые соединяют переменные в нейронной сети, частично аналогичны коэффициентам стандартной регрессионной модели и могут быть использованы для того, чтобы описать взаимосвязь между переменными. Веса диктуют относительное влияние информации, которая обрабатывается в сети так, что входные переменные, которые слабо коррелируют с переменной отклика, подавляются весами. Противоположный эффект наблюдается для весов, назначенных для переменных, которые имеют сильные положительные или отрицательные ассоциации с переменной отклика. Очевидным различием между нейронной сетью и регрессионной моделью яв-

ляется то, что число весов является чрезмерным в первом случае. Эта характеристика является выгодной в том смысле, что она делает нейронные сети очень гибкими для моделирования нелинейных функций, хотя интерпретация влияния конкретных переменных, конечно, не является простой.

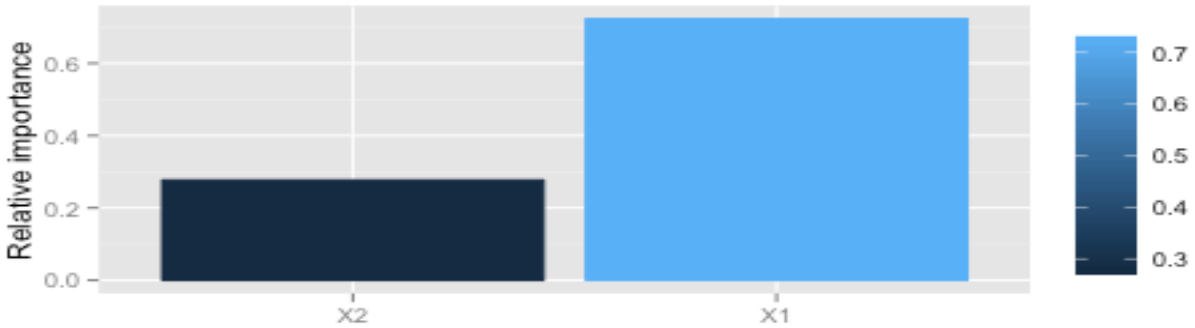
Способ, описанный Гарсоном, определяет относительную важность входных переменных для отдельных переменных отклика в контролируемой нейронной сети с помощью деконструкции модели весов. Относительная важность (или сила связи) конкретной входной переменной для переменной отклика может быть определена путем выявления всех взвешенных соединений, включенных между входным и выходным узлами. То есть определяются все веса, соединяющие конкретный входной узел и проходящие через скрытый слой к переменной отклика. Это повторяется для всех других независимых переменных, пока не будет составлен список всех весов, которые являются специфическими для каждой входной переменной. Соединения подсчитываются для каждого входного узла и масштабируются по отношению ко всем другим входам. Для каждой входной переменной получается единственное значение, описывающее ее отношение с переменной отклика в модели. Алгоритм оценивает относительную важность абсолютной величиной от нуля до единицы.

Алгоритм работает в настоящее время только для нейронных сетей с одним скрытым слоем и одной переменной отклика.

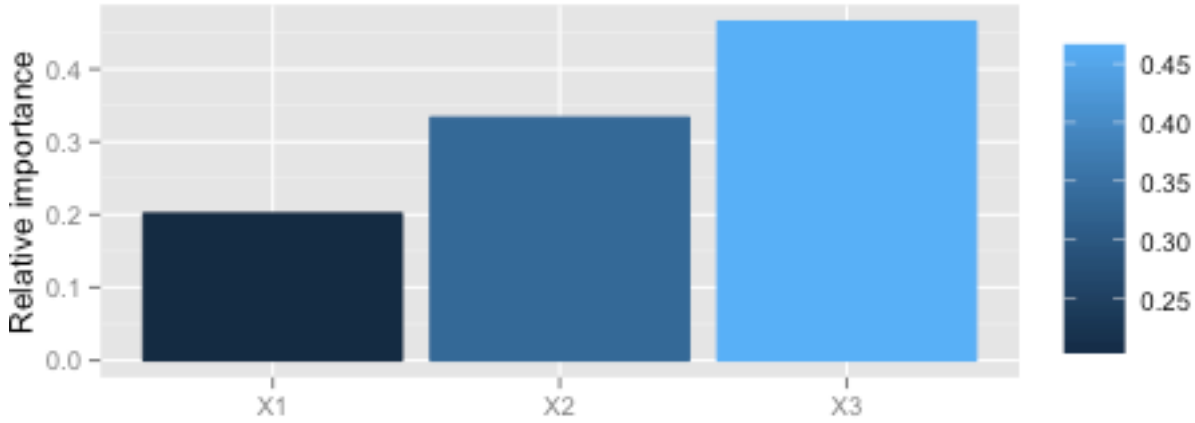
В последующих примерах используется фрейм данных neuraldat, включающий 300 наблюдений (строк) с тремя входными переменными и двумя выходными переменными. Входные переменные отобраны из стандартного нормального распределения. Выходные переменные образуют линейные комбинации независимых переменных. Каждая переменная во фрейме данных нормализована до значений от 0 до 1.

Пример 29

```
## использование входного объекта класса numeric
wts_in <- c(13.12, 1.49, 0.16, -0.11, -0.19, -0.16, 0.56, -0.52, 0.81)
struct <- c(2, 2, 1) #two inputs, two hidden, one output
garson(wts_in, struct)
```



```
## использование входного объекта класса neuralnet
library(neuralnet)
mod <- neuralnet(Y1 ~ X1 + X2 + X3, data = neuraldat, hidden = 5)
garson(mod)
```



5.2. Функция lekprofile()

Функция анализирует чувствительность выходов модели нейронной сети по отношению к входным переменным с использованием метода профиля Лека. Вызов функции:

```
lekprofile(mod_in, exp_in, steps = 100, split_vals = seq(0, 1,
    by = 0.2), val_out = FALSE, ...)
```

mod_in in – входной объект, для которого формируются оценки; Входной объект может относиться к классу nnet или mpl;

exp_in – (только для класса mlp!) матрица или фрейм данных, используемые для создания модели;

steps – числовое значение указывает количество наблюдений для оценки каждой входной переменной, по умолчанию – 100;

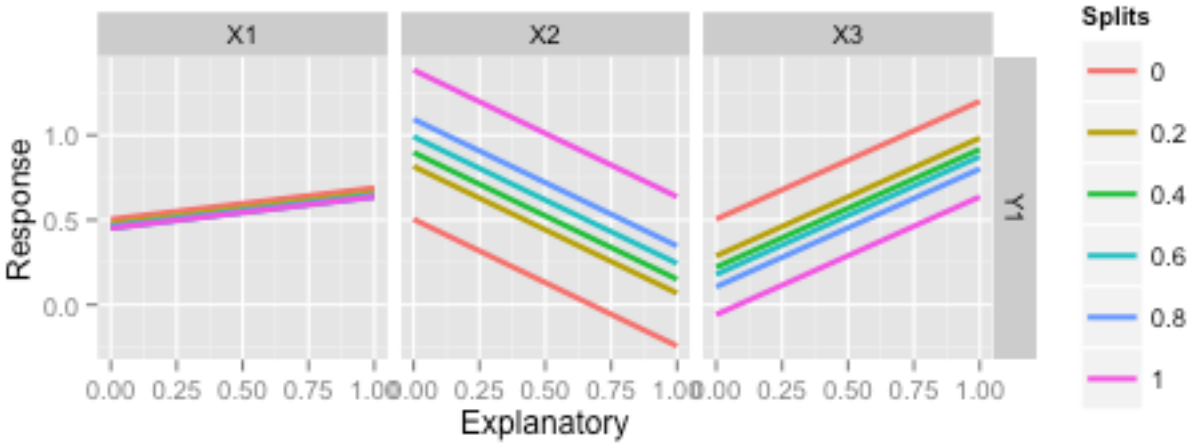
split_vals – числовой вектор, указывающий постоянные значения других переменных при анализе одной;

val_out – логическое значение, указывает, что возвращаются фактические значения чувствительности, а не график. По умолчанию – FALSE;

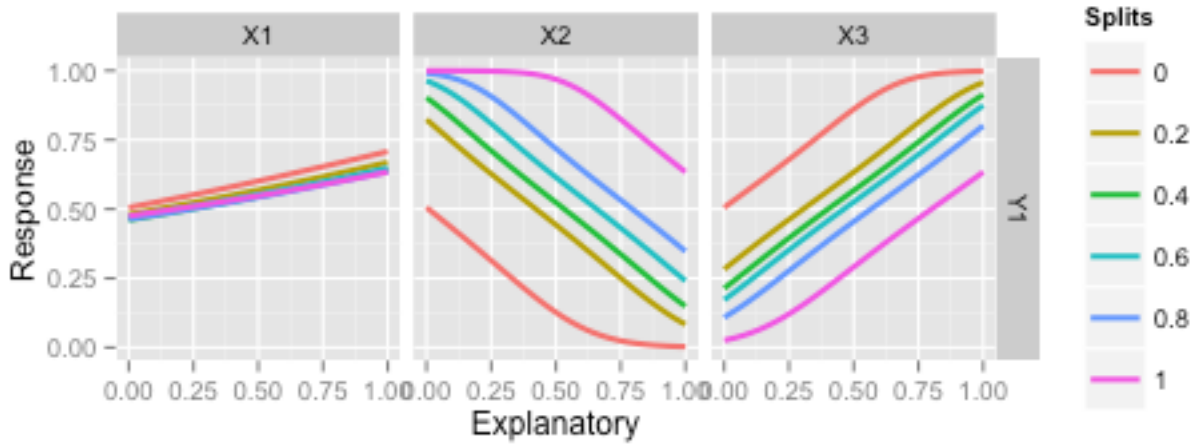
... – аргументы (параметры), передаваемые другим методам (функциям).

Пример 30

```
## a simple lm
data(neuraldat)
mod <- lm(Y1 ~ X1 + X2 + X3, data = neuraldat)
lekprofile(mod)
```




```
## using nnet
library(nnet)
set.seed(123)
mod <- nnet(Y1 ~ X1 + X2 + X3, data = neuraldat, size = 5)
lekprofile(mod)
```



5.3. Функция neuralweights()

Функция позволяет получить веса в виде списка значений, извлеченных из нейронной сети. Эта функция обычно не вызывается самостоятельно (см. параметр wts_only wts_only функции garson). Вызов функции:

```
neuralweights(mod_in, rel_rsc = NULL, struct, ...)
```

Здесь mod_in – нейронная сеть (объект), для которой формируется список весов; Нейронная сеть (объект) может относиться к классу numeric, nnet, mpl или nn;

rel_src – числовое значение, указывает максимальное масштабирование веса для построения интерпретационной диаграммы. По умолчанию значение NULL – без перемасштабирования;

struct – (только для объекта класса numeric) числовой вектор, равный по длине числу слоев в сети; Каждое число указывает количество узлов в каждом слое, начиная с входного и заканчивая выходом. Может быть указано произвольное число скрытых слоев.

Каждый элемент возвращаемого списка обозначается как «слой-узел», например, «out 1» является первым узлом в выходном слое. Скрытые слои нобозначаются с использованием трех значений, например: «hidden 1 1» является первым узлом в первом скрытом слое, «hidden 2 1» является первым узлом во втором скрытом слое и т. д. Значения в каждом элементе списка представляют веса входов в конкретный узел из предыдущего слоя в последовательном порядке, начиная со смещения, если оно есть. Например, элементы в списке для узла «hidden 1 1» нейронной сети со структурой 3 5 1 (3 входа, 5 скрытых узлов, 1 выход) будет иметь четыре значения, указывающие вес в последовательности: смещение, первый узел входного слоя, второй узел входного слоя и третий узел входного слоя к первому узлу скрытого слоя.

Пример 31

```
data(neuraldat)
set.seed(123)
## для класса numeric input
wts_in <- c(13.12, 1.49, 0.16, -0.11, -0.19, -0.16, 0.56, -0.52, 0.81)
struct <- c(2, 2, 1) # два узла входного слоя, два узла скрытого слоя, один выход
neuralweights(wts_in, struct = struct)
$wts$`hidden 1 1`
[1] 13.12  1.49  0.16
$wts$`hidden 1 2`
[1] -0.11 -0.19 -0.16
$wts$`out 1`
[1]  0.56 -0.52  0.81

## для класса nnet
library(nnet)
mod <- nnet(Y1 ~ X1 + X2 + X3, data = neuraldat, size = 5, linout = TRUE)
neuralweights(mod)
$wts$`hidden 1 1`
[1]  0.4143261  0.2487959 -0.2908262  1.0108349
$wts$`hidden 1 2`
[1] 2.9797245 0.4477592 1.1048278 1.3746734
$wts$`hidden 1 3`
[1]  0.59936349 -0.06161125  0.81214729 -0.70024032
$wts$`hidden 1 4`
[1]  1.34321424  0.07839821 -0.97108484  0.53072623
```

```
$wts$`hidden 1 5`
[1] -0.5192076  0.2641261 -0.5627597  0.4939219
$wts$`out 1`
[1]  0.1785571  0.8249573 -0.7924542 -1.4281850  1.0706880  1.7755640
```

5.4. Функция olden()

Функция позволяет оценить относительную важность входных переменных в нейронных сетях как сумму произведения весов соединения слоев входного-скрытого и скрытого-выходного, предложенную Олденом.

Вызов функции:

```
olden(mod_in, out_var, struct, bar_plot = TRUE,
    x_lab = NULL, y_lab = NULL, wts_only = FALSE, ...)
```

Здесь mod_in – входной объект, для которого формируются оценки. Входной объект может относиться к классам numeric, nnet, mpl или nn;

out_var – строка, указывающая переменную отклика в нейронной сети, по отношению к которой оценивается относительная важность входных переменных;

struct – (только для класса numeric!) числовой вектор, равный по длине числу слоев в сети. Каждое число указывает количество узлов в каждом слое, начиная с входного и заканчивая выходом. Может быть включено произвольное число скрытых слоев;

bar_plot – указывает, что возвращается объект ggplot, в противном случае возвращаются числовые значения;

x_lab – строка альтернативных имен, которые будут использоваться для независимых переменных на рисунке, по умолчанию берется из mod_in;

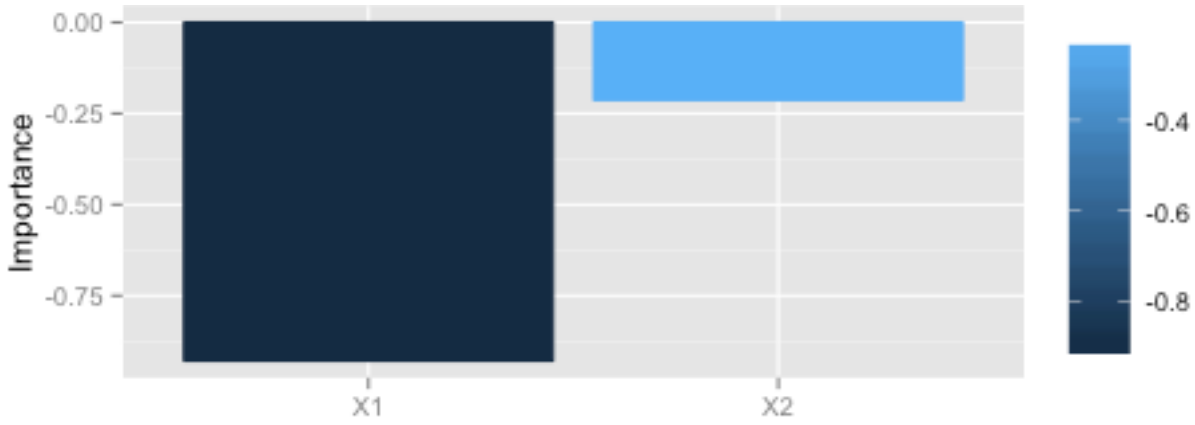
y_lab – строка альтернативных имен, которые будут использоваться для выходных переменных на рисунке, по умолчанию берется из out_var;

wts_only – указывает на вызов функции neuralweights(), по умолчанию FALSE;

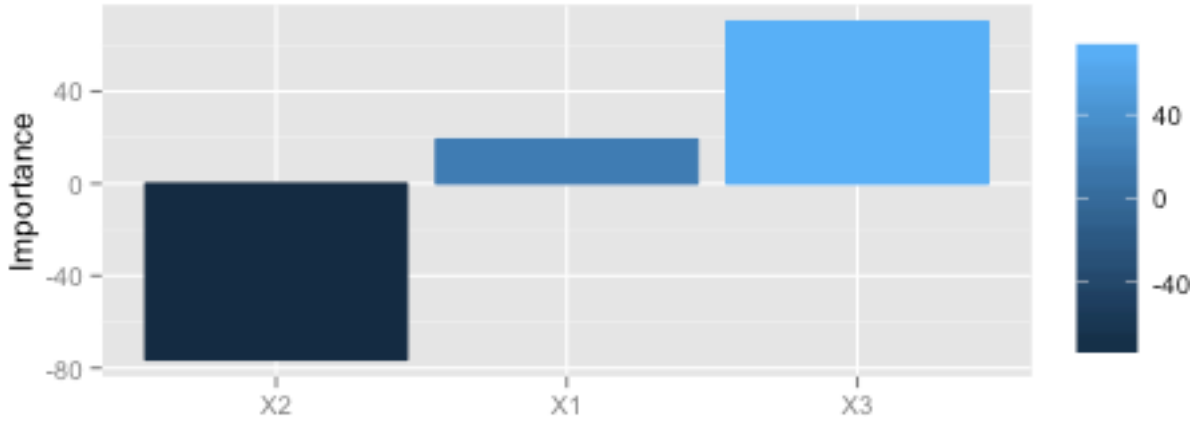
... – аргументы (параметры), передаваемые другим методам (функциям).

Пример 32

```
## для класса numeric input
wts_in <- c(13.12, 1.49, 0.16, -0.11, -0.19, -0.16, 0.56, -0.52, 0.81)
struct <- c(2, 2, 1) #two inputs, two hidden, one output
olden(wts_in,'Y1',struct)
```




```
## для класса nnet
library(nnet)
data(neuraldat)
set.seed(123)
mod <- nnet(Y1 ~ X1 + X2 + X3, data = neuraldat, size = 5)
olden(mod,'Y1')
```



5.5. Функция plotnet()

Функция осуществляет графическое построение структуры нейронной сети. Она предоставляет широкие возможности по заданию цвета, размера и других параметров изображения.

Ниже приведен вариант вызова функции plotnet() с указанием значений ее параметров, которые устанавливаются по умолчанию:

```
plotnet(mod_in, struct, nid = TRUE, all_out = TRUE, all_in = TRUE,
    bias = TRUE, wts_only = FALSE, rel_rsc = 5, circle_cex = 5,
    node_labs = TRUE, var_labs = TRUE, x_lab = NULL, y_lab = NULL,
    line_stag = NULL, cex_val = 1, alpha_val = 1,
    circle_col = «lightblue», pos_col = «black», neg_col = «grey»,
    bord_col = «lightblue», max_sp = FALSE)
```

Здесь mod_in – нейронная сеть (объект) или числовой вектор весов; Входной объект может относиться к классам numeric, nnet, mpl или nn;

struct – (только для класса numeric!) числовой вектор, равный по длине числу слоев в сети. Каждое число указывает количество узлов в каждом слое, начиная с входного и заканчивая выходом. Может быть включено произвольное число скрытых слоев;

nid – флаг, определяющий, что строится графическое изображение;

all_out – строковые значения имен выходных переменных, для которых строятся связи, по умолчанию – для всех;

all_in – строковые значения имен входных переменных, для которых строятся связи, по умолчанию – для всех;

bias – флаг, указывающий, что связи для смещения строятся, не применим для mlp сетей;

wts_only – флаг, указывающий, что возвращаются веса связей вместо графического изображения сети;

rel_rsc – числовое значение, определяющее максимальную толщину линий связи, по умолчанию 5;

circle_cex – числовое значение, определяющее размер узлов, по умолчанию 5;

node_labs – флаг, указывающий, что метки проставляются прямо на узлах.

var_labs – флаг, указывающий, что имена переменных проставляются рядом с узлами;

x_lab – строковые имена входных переменных, по умолчанию берутся из модели mod_in;

y_lab – строковые имена выходных переменных, по умолчанию берутся из модели mod_in;

line_stag – числовое значение, определяющее расстояние от связи до узла;

cex_val – числовое значение, определяющее размер текстовой метки;

alpha_val – числовое значение от 0 до 1, определяющее прозрачность связей;

circle_col – строка, определяющая цвет узлов или список из двух элементов: первый определяет цвет входных узлов, а второй – цвет всех остальных;

pos_col – определяет цвет связей с положительными весами;

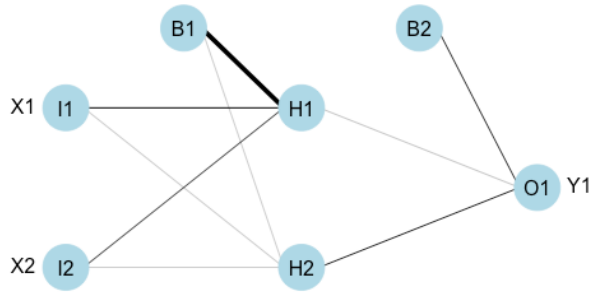
neg_col – определяет цвет связей с отрицательными весами;

bord_col – строка, определяющая цвет границы круга, обозначающего узел;

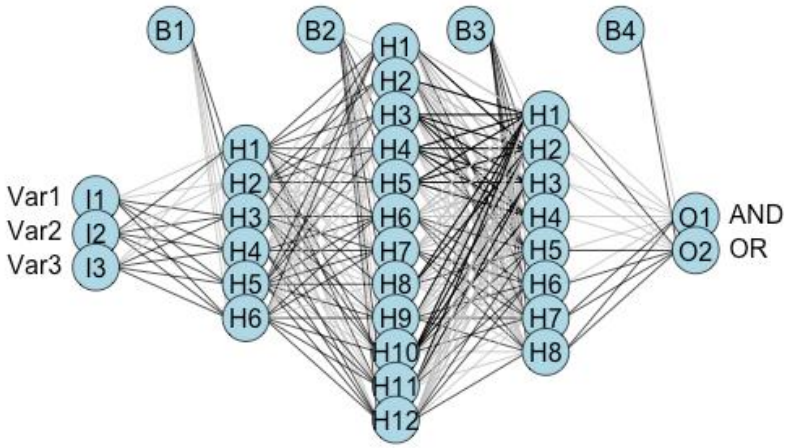
max_sp – флаг, указывающий, когда расстояния между узлами в каждом слое максимальные.

Пример 33

```
#
wts_in <- c(13.12, 1.49, 0.16, -0.11, -0.19, -0.16, 0.56, -0.52, 0.81)
struct <- c(2, 2, 1) #two inputs, two hidden, one output
plotnet(wts_in, struct = struct)
```




```
#
AND <- c(rep(0, 7), 1)
OR <- c(0, rep(1, 7))
binary_data <- data.frame(expand.grid(c(0, 1), c(0, 1), c(0, 1)), AND, OR)
mod<-neuralnet(AND + OR ~ Var1 + Var2 + Var3, binary_data,
            hidden = c(6, 12, 8), rep = 7,err.fct='ce',linear.output=FALSE)
 plotnet(mod,rel_rsc = 1, circle_cex = 4, bord_col = 'black')
```



5.6. Функция predsens()

Функция позволяет получить предсказанные значения для метода профиля Лека. Она многократно используется в функции lekprofile(). Вызов функции:

```
pred_sens(mat_in, mod_in, var_sel, step_val, fun_in, resp_name)
```

Здесь mat_in – фрейм данных только входных переменных, используемых для создания модели;

mod_in – любой объект с методом прогнозирования;

var_sel – строка с именем выбираемой входной переменной;

step_val – число значений диапазона предсказаний для выбранной входной переменной;

fun_in – функция, определяющая способ удержания независимых переменных постоянными;

resp_name – строка с именами переменных отклика для правильности маркировки.

Прогноз значений выходной переменной получается на основе матрицы входных переменных, которые ограничены методом профиля Лека. Выбранная входная переменная изменяет значения в указанном диапазоне. Все остальные входные переменные остаются неизменными по величине, определяемой функцией fun_in.

Пример 34

```
library(nnet)
data(neuraldat)
set.seed(123)
mod <- nnet(Y1 ~ X1 + X2 + X3, data = neuraldat, size = 5)
mat_in <- neuraldat[, c('X1', 'X2', 'X3')]
pred_sens(mat_in, mod, 'X1', 10, function(x) quantile(x, 0.1), 'Y1')
```

```
              Y1          x_vars
1  0.4854596778       0.0000000000
2  0.5040814715       0.1111111111
3  0.5233131410       0.2222222222
4  0.5431867429       0.3333333333
5  0.5637203043       0.4444444444
6  0.5849154519       0.5555555556
7  0.6067549923       0.6666666667
8  0.6292005273       0.7777777778
9  0.6521902347       0.8888888889
10 0.6756369877       1.0000000000
```

ЗАКЛЮЧЕНИЕ

В настоящее время теория и практика машинного обучения переживают настоящую «глубинную революцию», вызванную успешным применением методов Deep Learning (глубокого обучения), представляющих собой третье поколение нейронных сетей Deep Neuronets (DN). Эти ассоциативные сети (накапливающие автоэнкодеры – StackedAuto Encoder) SAE и накапливающие сети Больцмана (Stacked Restricted Boltzmann Machine) RBM, натренированные с помощью алгоритмов глубокого обучения, – не просто превзошли по точности лучшие альтернативные подходы, но и в ряде задач проявили зачатки понимания смысла подаваемой информации, например при распознавании изображений и анализе текстовой информации.

Библиотеки языка R содержат в своем арсенале такие программные пакеты, как deepnet и darh, реализующие модели DN SAE и DN SRBM. Пакет H2O предназначен для обучения моделей глубоких сетей на «больших наборах данных» (> 1 Гбайт), записанных в csv-файлах.

Изучение этих и других программных средств основано на материале, приведенном в настоящем учебном пособии, и послужит целью следующих изданий.

ИСПОЛЬЗОВАННЫЕ ИСТОЧНИКИ

1. http://www.osp.ru/os/1997/04/179189/
2. http://www.intuit.ru/studies/courses/88/88/lecture/20527
3. https://cran.r-project.org/web/packages/neuralnet/neuralnet.pdf
4. https://cran.r-project.org/web/packages/nnet/nnet.pdf
5. https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf
6. https://cran.r-project.org/web/packages/NeuralNetTools/NeuralNet Tools.pdf
7. https://www.mql5.com/ru/articles/1103#3_1

Филиппов Феликс Васильевич

МОДЕЛИРОВАНИЕ НЕЙРОННЫХ СЕТЕЙ НА R

Учебное пособие

Редактор *И. И. Щенсняк*

Компьютерная верстка *Н. А. Ефремовой*

План издания 2016 г., п. 80

Подписано к печати 01.09.2016
Объем 5,25 усл.-печ. л. Тираж 26 экз. Заказ 692

Редакционно-издательский отдел СПбГУТ
191186 СПб., наб. р. Мойки, 61

Отпечатано в СПбГУТ