Нейронные сети

Идея нейронных сетей возникла в результате попыток смоделировать поведение живых существ, воспринимающих воздействия внешней среды и обучающихся на собственном опыте. Такого рода идеи на стыке различных областей знания характерны для науки современного времени.

Наша цель состоит в том, чтобы кратко описать идею нейронных сетей и научить читателя экспериментировать с нейронными сетями в системе STATISTICA. Более подробная информация о нейронных сетях доступна в текстах Уссермен Ф. Нейрокомпьютерная техника, М.: Мир, 1992, Lippman R. P. An introduction to computing with neural nets, IEEE ASSP Magazine. Apr. 1987, p. 4-22, и др.

Ключевым является понятие нейронов – специальных нервных клеток, способных воспринимать, преобразовывать и распространять сигналы.

Начнем со следующей модели нейрона. Хотя эта модель очень простая, она работает. Итак, нейрон имеет *несколько* каналов ввода информации – дендриты, и *один* канал вывода информации – аксон. Аксоны нейрона соединяются с дендритами других нейронов с помощью синапсов. При возбуждении нейрон посылает сигнал по своему аксону. Через синапсы сигнал передается другим нейронам, которые, в свою очередь, могут возбуждаться или, наоборот, оказываться в состоянии торможения.

Заметьте, биологические образы естественны при описании процесса обучения, создавая контекст для математических рассуждений.

Нейрон возбуждается, когда суммарный уровень сигналов, пришедших в него, превышает определенный уровень (порог возбуждения или активации). Интенсивность сигнала, получаемого нейроном, зависит от активности синапсов.

Итак, запомним следующее.

О Нейрон получает сигналы через *несколько* входных каналов. Каждый сигнал проходит через соединение – *синапс,* имеющее определенную интенсивность, или вес, который соответствует синаптической активности нейрона.

О Текущее состояние нейрона определяется формулой:

$$u_i = \sum \omega(i,j)x(j) + \omega(i,0) \quad (1),$$

где *x(j), j = 1,2...N* – входные сигналы. Коэффициенты *w(i,j)* называются весами синаптических связей, положительное значения которых соответствуют *возбуждающим* синапсам, отрицательные значения - *тормозящим* синапсам. Если *w(i,j) = 0,* то говорят, что связь между нейроном *г* и нейроном *j* отсутствует. Величина *w(i,0)* называется *пороговым* значением.

О Полученный нейроном сигнал преобразуется с помощью функции **активации или передаточной** функции / в *выходной* сигнал

$$y_i = f(u) \qquad (2)$$

Это одна из первых моделей нейрона предложена МакКаллоком и Питсом в 1943 году.

Заметим также, что имеется *стохастическая* модель нейрона, в которой выходной сигнал является *случайной* величиной, принимающей пару значений, которые , соответствуют торможению или возбуждению.

С математической точки зрения в модели нейрона мы имеем *нелинейное* преобразование вектора *x(1),x(2)...x(N)* в выходной сигнал y_i.

Функция активации или передаточная функция/ в формуле (2) – это некоторая нелинейная функция, моделирующая процесс передачи возбуждения.

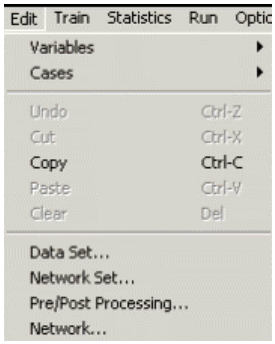
Простейшие пример такой функции – индикаторная или скачкообразная функция, определяемая равенствами: *f(u)*=1, если *u>=0, f(u)* = 1, если *u<0.*

Если выбрать функцию вида $f(u) = \frac{1}{1+e^{-bu}}$.

где *b > 0,* то получится так называемый *сигмоидальный* нейрон и т. д.

Объединенные между собой нейроны образуют сеть, с математической точки зрения задающую сложное многомерное преобразование, собранное из простейших преобразований Замечательно, что с помощью таких простейших преобразований можно приближать очень сложные *многомерные* функции, следовательно, оценивать сложные зависимости (заметим, замечательная теорема *Колмогорова* является математическим основанием нейронных сетей).

STATISTIC A позволяет задавать различные передаточные функции, например, линейную, логистическую и др (эти функции можно выбрать в диалоговом окне *Network Editor,* доступном из меню *File).*



Выходы нейронов соединяются с входами других нейронов, таким образом, сигнал от одного нейрона передается другим нейронам (нейрон информирует о своем состоянии другие нейроны). Конечно, с математической точки зрения мы имеем преобразование исходных значений *X* на входе сети в значения *У* на выходе. На биологическом языке входы и выходы соответствуют сенсорным и двигательным нервам. Кроме входных и выходных нейронов в сети могут присутствовать еще промежуточные (скрытые) слои нейронов. Простейшие сети имеют структуру прямой передачи сигнала: сигналы проходят от *входов* через скрытые элементы и в конце концов поступают на *выходные* элементы (см. рисунки).

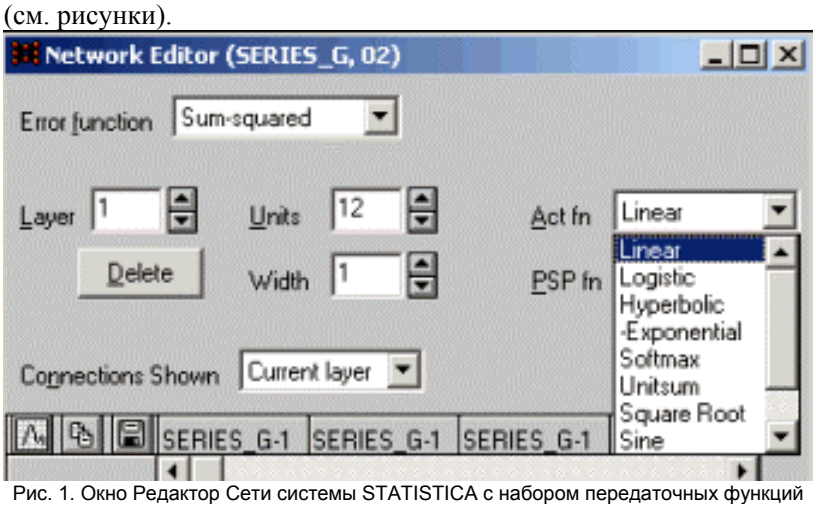


Рис. 1. Окно Редактор Сети системы STATISTICA с набором передаточных функций

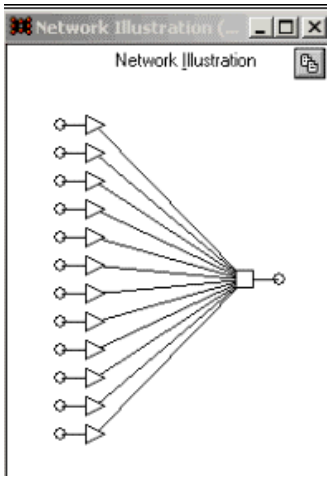
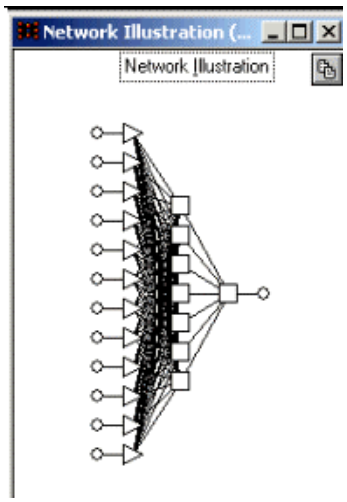


Рис. 2. Двухслойная сеть, имеющая 12 входов, 1 выход и 6 элементов на промежуточном слое



Рис, 3. Трехслойная сеть, имеющая 12 входов и 1 выход

Итак, каждый нейрон как элемент сети описывается своим набором параметров (см. формулы 1,2).

Входной слой служит для ввода значений входных переменных, выходной слой – для вывода результатов. Скрытые выходные нейроны соединены со всеми элементами предыдущего слоя. Последовательность слоев и их соединений называется *архитектурой* сети.

При работе сети на входные элементы подаются значения входных переменных (входной сигнал), затем возбуждаются нейроны первого промежуточного слоя, далее – второго промежуточного слоя, в итоге *преобразованный* сигнал поступает на выходной слой.

Преобразование сигнала проводится следующим образом.

Последовательно для каждого нейрона в сети вычисляется значение активации, берется взвешенная сумма выходов элементов предыдущего слоя и вычитается пороговое значение. Затем значения активации преобразуются с помощью передаточной функции, и в результате получается выход нейрона, поступающий на вход нейронов, с которыми он соединен.

Обучение сети

Обычно нейронные сети используют в задачах классификации, прогнозирования и построения нелинейных зависимостей (нелинейная регрессия).

Но для этого сеть нужно обучить. Замечательный факт состоит в том, что нейронную сеть *действительно* можно обучить!

Теорема Колмогорова – это высший уровень абстракции, рассмотрение нейронов – самый низкий или глубокий. Объединяя эти два уровня, мы пытаемся по существу понять, как организовано мышление, когда состоящий из простейших нейронов человеческий мозг постигает глубочайшие закономерности. Процесс получения знания моделируется с помощью нейронных сетей.

Мы знаем, что знания получаются последовательно, иными словами они не даются в законченном виде, а достигаются с помощью обучения, этот принцип использован в нейронных сетях. Итак, мы построили модель нейрона и нейронной сети, теперь нужно предложить модель обучения.

Как мы уже отмечали, формально соотношения (1), (2) задают простое преобразование величин с различными функциями f. Пусть мы имеем сложное преобразование F исходного набора данных (который поступает на вход сети) в выходной набор (который наблюдается на выходе сети).

Возникает вопрос: как реализовать преобразование F с помощью нейронной сети. На математическом языке мы должны приблизить неизвестную сложную функцию простейшими преобразованиями, задаваемыми уравнениями (1), (2). Теорема Колмогорова утверждает в *принципе,* что такая сеть существует, но не говорит, *как именно* ее настроить. Мы используем общий подход, связанный с обучением, то есть последовательным получением знаний, наказанием за неправильный ответ и поощрением за правильный ответ.

Вначале мы определяем архитектуру сети, то есть устанавливаем количество нейронов и связи между ними, выбираем конкретную синаптическую функцию, моделирующую процесс передачи возбуждения.

Разобьем данные на две части, обучающие и контрольные, на сленге нейронных сетей –

обучающую и *контрольную* выборку.

Общая идея состоит в следующем: вначале на вход сети подается *обучающая* выборка с *известными* результатами, величины *X*и наблюдаются отклики *Y=F(X)*.

Меняя веса *w(i,j)* и значения порога активации для каждого нейрона мы настраиваем сеть, иными словами, находим как можно более точное приближение функции *F*.

Далее на тестовой выборке экзаменуем простроенную сеть или сети, если их несколько (в общем случае мы получаем ансамбль сетей). Например, в задаче *классификации* мы можем потребовать, чтобы сеть правильно классифицировала не менее 90% наблюдений. В задаче *прогнозирования* мы можем стремиться к тому, чтобы точность прогноза на определенное количество шагов вперед была не ниже заданной. Если сеть прошла экзамен, мы можем использовать ее для анализа данных, построить прогноз или провести классификацию.

Очевидно, невозможно умозрительно организовать данный процесс в силу его трудоемкости и сложного преобразования данных, только компьютерные технологии позволяют эффективно сделать это.

Конечно, в данном процессе имеется определенный произвол связанный, например, с выбором обучающей выборки и риском применения сети на реальных данных, но тот же произвол возникает при применении любых математических методов на практике, именно потому, что эти методы имеют дело с *сырыми данными* (действительностью), а не с возвышенными числами, с которыми они призваны оперировать.

В замечательной модели нейронных мы имеем синтез различных методов, которые могут «ожить» только с помощью компьютерных технологий.

Рассмотрим идею обучения на простой и ясной модели Розенблатта *однослойного* персептрона. Анализируя алгоритм, вы можете заметить, что он основан на древнем как мир принципе кнута и пряника. Если сеть правильно классифицирует сигнал, она получает пряник, в противном случае кнут.

Модель Розенблатта (однослойный персептрон – single layer perceptron)

Как видно из названия, в этой модели число слоев равно 1, поэтому исключим второй индекс и рассмотрим только веса да(г'), $1 < i < N$ (см. формулы (1), (2)).

Конечно, заранее эти веса не известны, и их нужно найти с помощью разумной процедуры.

На вход сети подается сигнал *(xl, x2 ... xN)*. Пусть входной сигнал может принадлежать либо классу А, либо классу Б. Предположим, для простоты, что мы анализируем двумерный сигнал, иными словами, число *N- 2*.

Обучение однослойного персептрона

Шаг 0. Начальные установки: веса да!(1), да2(1) и порог Т задаются *случайным* образом.

Будем обозначать *t* шаг обучения. Вначале $t = 0$.

Шаг 1. Положим $t = t+l$. Предъявим сети входной сигнал из *обучающей* выборки: (лг! *(t),x2(t))*.

Определим *d{t)* = 1, если входной сигнал принадлежит классу А, и *d(J)* = -1, если входной сигнал принадлежит классу Б

Шаг 2. Вычислим *состояние* нейрона в момент времени t (просто суммируем входные сигналы с весами и вычитаем порог 7): *s(t) = w\(t)* x *x\(t) + w2(t)* x *x2(f) – T.*

Шаг 3. Вычислим *выходной* сигнал нейрона *y(f)* в момент *t* (заметьте, используется скачкообразная функция): *y(t)* = sign(s(Z:))

Шаг 4. С учетом результата обучения вычислим новые веса нейрона по формулам: *w\(t) = w\(t-l)* + rx *(y(t) - d(t)), w2(t) = w2(t-i) + r(y(f) - d(t)), г*- шаг обучения.

Шаг 5. Если шаг обучения г меньше объема обучающей выборки *L,* то переходим к шагу 1.

В противном случае обучение заканчивается.

Таким образом, получается обученный персептрон, который может решать простые задачи классификации. Если вы захотите доказать, что это *действительно* обученный персептрон, то вам следует воспользоваться методами теории вероятности или проверить это утверждение экспериментально, например с помощью статистического моделирования.

Многослойный персептрон

Обобщение однослойного персентрона приводит к *многослойному* персептрону (см. рис. 2 и 3).

В многослойном персептроне каждый элемент сети строит взвешенную сумму своих входов с поправкой в виде слагаемого, а затем пропускает вычисленное значение через передаточную функцию.

Таким образом, по общим правилам получается *выходное* значение персептрона.

Нейроны организованы в послойную структуру с прямой передачей сигнала. Веса и пороговые значения являются свободными параметрами модели, которые оцениваются в процессе обучения.

Многослойный персептрон может моделировать функцию практически любой степени сложности.

Имея в своем распоряжении STATISTICA, вы можете всесторонне экспериментировать с моделями, переходя от простых моделей к более сложным.

Конечно, с математической точки зрения, нейронная сеть осуществляет *преобразование* одного сигнала в другой. Фокус состоит в том, что это преобразование подчиняется рекурсивным правилам и может быть реализовано технически.

Общий взгляд

Сделаем шаг в сторону и посмотрим на нейронные сети с общих позиций. Как мы говорили (см. главу 2), одной из основных задач анализа данных является оценка зависимости между переменными, например, между переменной *X* и переменной *Y.* Наблюдая различные значения переменной *X* и соответствующие значения переменной *Y,* мы хотим оценить зависимость *Y = F(X).*

В частном случае мы хотим оценить *линейную* зависимость *F(X) = a*X + b,* где *a, b* неизвестные константы, или *полиномиальную* зависимость, когда *F*представляет собой полином некоторой степени. Можно также разложить функцию F в ряд Фурье и, используя комбинации синусов и косинусов или других базисных функций, последовательно приближать функцию F. В различных разделах анализа используются различные методы решения этой задачи.

В нейронных сетях мы *собираем* функцию F из простейших нейронов, комбинируя их разнообразным образом друг с другом. Получая на вход набор X, с помощью *простейших* функций мы преобразуем X в Y, ожидая при этом, что собранная сеть приближает искомую функцию F. Конечно, такая игра может показаться бессмысленной, но знаменитая теорема Колмогорова, о которой часто не подозревают практики, утверждает, что такие упражнения вполне оправданы, – действуя подобным образом можно в принципе собрать из *простейших* нейронов сколь угодно сложную функцию F. Теорема Колмогорова утверждает также, что достаточно иметь не более двух скрытых слоев нейронов в сети для восстановления зависимости

Заметьте, явный вид *собранной* функции нам не интересен, для нас важно в принципе знать, что она близка к искомой.

Как проверить, насколько собранная функция близка той, которую мы ищем?

Одним из естественных подходов к решению этой задачи является следующий данные разбиваются на две части, по одной из которых строится оценка функции, собранной из нейронов, на второй части данных проверяется, насколько построенная функция близка искомой (такая процедура называется кросс-проверкой, см. также раздел *Обучение сети).* Конечно, подобное решение нематематично (действительно, оно зависит, например, от того, как именно произведено разделение данных на обучающую и тестовую выборку) и не может удовлетворить любителей строгости, однако оказывается вполне приемлемым во многих прикладных задачах. Заметим, что программа SNN предлагает различные способы проверки качества построенной сети.

Теперь можно приступить к экспериментированию с нейронными сетями в системе STATISTICA.

Обратим внимание, что в модуле *Нейронные сети* системы STATISTICA имеется *Советник,* подсказывающий выбор архитектуры сети (см. описанный ниже пример классификации с помощью нейронных сетей).

Покажем, как построить многослойный персептрон в системе STATISTICA.

Построение многослойного персептрона в системе STATISTICA

Шаг 1. Запустите модуль *Нейронные сети.*

Шаг 2. Откройте, например, файл *senes_g.sta* из папки *Examples.* Используйте меню *File-Open.* Файл содержит данные о месячных авиаперевозках пассажиров.

Если вы хотите создать свой набор данных в модуле *Нейронные сети,* поступите следующим образом:

О Войдите в диалоговое окно *Создать набор данных – Create Data Set с* помощью команды *Набор данных – Data Set...* из меню *Файл–Новый – File–New.*

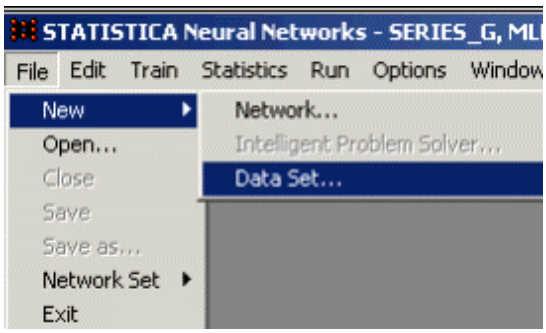


Рис. 4. Создание файла данных

О Введите нужные значения для количества *входных – Inputs* и *выходных Outputs* переменных в наборе данных. Введите, например, 17 и 7.

О Нажмите кнопку *Создать – Create.*

Заметьте, что имена входных переменных имеют черный цвет, имя выходных переменных – голубой цвет; входы от выходов отделяются темной вертикальной линией.

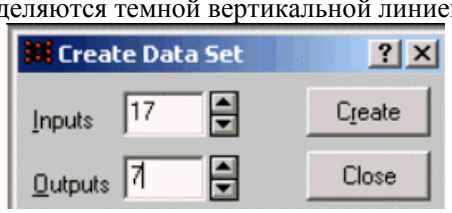


Рис. 5. Определение числа входных и выходных наблюдений

В данном примере, однако, мы не будем создавать нового файла, а будем работать с имеющимся файлом *series_g.sta.*

Шаг 3. После того как файл данных *series_g.sta* открыт, перейдем к созданию сети.

Для этого в меню *File* выберите команды: *New–Network – Новая Сеть* (см. рис. 6).

Шаг 4. Вначале создадим *структуру* сети. В появившемся диалоговом окне сделайте установки, как показано на рис. 7.

В поле *Type – Тип* выберите тип сети: *Многослойный персептрон.*

Задайте параметр *Временное окно – Steps* равным *12.* Мы выбрали эту установку, так как в ряде имеется сезонная составляющая с лагом 12.

Установите параметр *Горизонт – Lookahead* равным *1.*

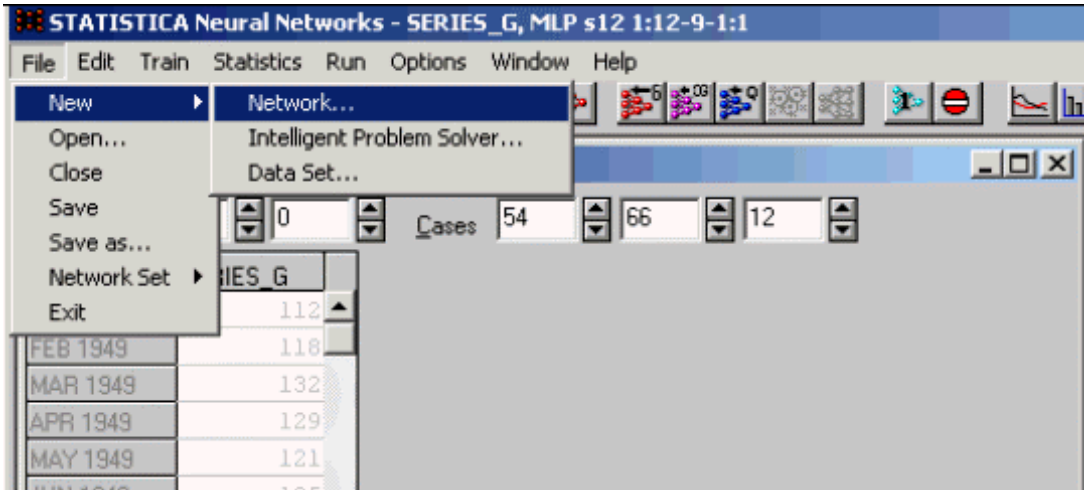


Рис. 6. Рабочее окно модуля Нейронные сети

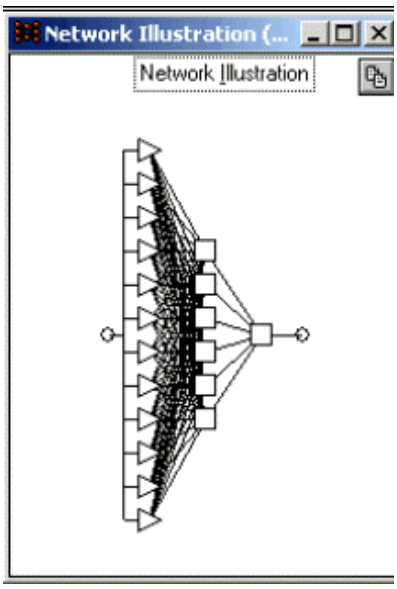


Рис. 7. Диалоговое окно построения сети

Данные содержат значения одной переменной. Для нейронной сети эта переменная будет служить одновременно входной и выходной (в разные моменты времени). Для того чтобы определить переменную как входную/выходную, нужно выделить ее щелчком на заголовке таблицы, а затем в появившемся меню выбрать пункт *Input/Output.*

Обратите внимание на установку в окне *No Layers – Число слоев.*

Мы выбрали сеть, содержащую 3 слоя. В таблице ниже для слоя 2 показано: *Layer 2 – Слой 2:1.*

В вашем распоряжении имеются две кнопки *Advise – Советовать* и *Create – Создать.*

Нажмите кнопку *Advise – Советовать.*

Заметьте, что после нажатия кнопки *Advise – Советовать* значение в поле *No Layers – Число слоев* изменится и станет равным 6.

Система советует выбрать 6 элементов на промежуточном слое. Вы можете воспользоваться советом или построить персептрон со своей структурой.

Например, вы можете щелкнуть мышью на поле *Layer2* и ввести любое значение для числа нейронов на слое 2. Гибкий интерфейс позволяет вам задавать архитектуру сети.

Шаг 5. Нажмите кнопку *Create – Создать.* На экране появится следующая сеть:

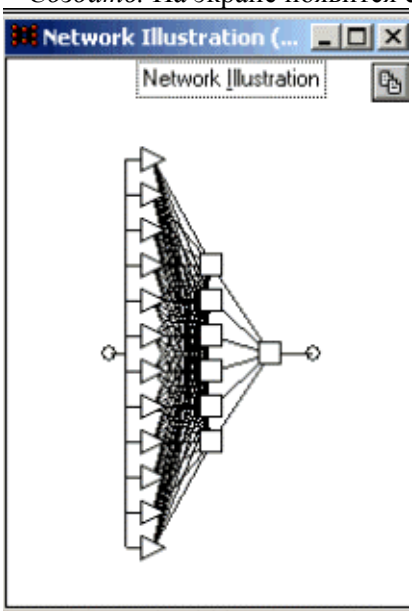


Рис. 8. Трехслойный персептрон с 7 элементами на 2-м слое

Таким образом, можно создать персептрон с нужным количеством слоев и числом элементов на каждом слое.

В окне *Редактор сети* STATISTICA можно послойно просмотреть и отредактировать сеть,

выбирать передаточную функцию для каждого слоя, а также постсинаптический потенциал или значение активации нейрона.

Итак, создана *архитектуру* сети. Мы продолжим рассмотрение этого примера, но вначале дадим необходимые теоретические сведения.

Обучение многослойного персептрона

После того как структура многослойного персептрона определена, его нужно обучить, то есть найти значения весов и порогов сети, являющиеся свободными (неизвестными) параметрами. Их нужно определить, чтобы сеть решала поставленную задачу. Представьте, вы случайным образом выбрали значения этих параметров. – вряд ли такая сеть будет для вас полезной. Трудно угадать нужные значения параметров, однако имеется процесс, называемый обучением, который позволяет *последовательно* находить эти параметры, приближаясь к лучшей сети.

Процесс обучения представляет собой подгонку модели, которая реализуется сетью, к обучающим данным, например, с известным ответом. Ошибка для конкретной сети определяется путем прогона всех имеющихся наблюдений и сравнения реально выдаваемых выходных значений сети с целевыми (правильными) значениями. Грубо говоря, мы обучаем сеть, продвигаясь в сторону уменьшения ошибок.

В качестве функции ошибки, например, можно взять среднеквадратичную ошибку, вычисляемую следующим образом: ошибки выходных элементов для всех наблюдений возводятся в квадрат и затем суммируются.

В модуле *Нейронные сети* выдается так называемая среднеквадратичная ошибка: описанная выше величина нормируется на число наблюдений и переменных, после чего из нее извлекается квадратный корень.

Это достаточно разумная мера ошибки, усредненная по всему обучающему множеству и по всем выходным элементам. Конечно, эта мера ошибки естественна в нелинейной регрессии, но вряд ли она полезна в задачах классификации, где критерием качества может являться доля правильно классифицированных наблюдений. Заметим, что разнообразные функции ошибок можно выбрать в окне *Редактор Сети.*

Итак, после того как мы задали архитектуру сети, нам нужно найти параметры, минимизирующие ошибку или максимизирующие качество работы сети.

В *линейных* моделях можно определить параметры, дающие *абсолютный* минимум ошибки.

С нелинейными моделями дело обстоит гораздо сложнее. Настраивая сеть с целью минимизации ошибки, нельзя быть уверенным, что алгоритм обучения достиг *глобального* минимума, иными словами, утверждать, что нельзя добиться лучшего результата.

Поверхность ошибок

Для контроля обучения сети полезна поверхность ошибок, к описанию которой мы сейчас переходим.

Каждому из весов и порогов сети (то есть свободных параметров модели; их общее число мы обозначим через *N)* соответствует одно измерение в многомерном пространстве. *(N*+ 1)-мерное измерение соответствует ошибке сети.

Для данного набора весов соответствующую ошибку сети можно изобразить точкой в *(N+ 1)-мерном* пространстве. В итоге все такие точки образуют некоторую поверхность – поверхность ошибок.

Цель обучения нейронной сети состоит в том, чтобы найти самую низкую точку этой поверхности В случае линейной модели с суммой квадратов в качестве функции ошибок поверхность ошибок представляет собой параболоид, и минимум находится легко.

В общем случае поверхность ошибок имеет сложную структуру, в частности, может иметь локальные минимумы (точки, самые низкие в некоторой своей окрестности, но лежащие выше глобального минимума), седловые точки и т. д.

Обучение нейронной сети заключается в исследовании поверхности ошибок. Отталкиваясь от некоторой начальной конфигурации весов и порогов, алгоритм обучения производит поиск глобального минимума.

Как правило, для этого вычисляется градиент в данной точке, а затем эта информация используется для продвижения вниз по склону на поверхности. В конце концов, алгоритм приводит к некоторой нижней точке (ниже спуститься нельзя), которая, однако, может оказаться лишь точкой *локального* минимума. Очевидно, следует использовать различные начальные приближения.

STATISTICA предлагает следующие методы обучения многослойного персептрона:

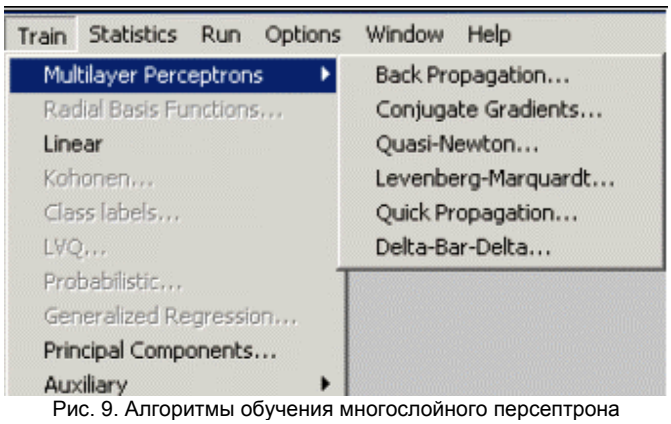


Рис. 9. Алгоритмы обучения многослойного персептрона

Для обучения многослойных персептронов в пакете *Neural Networks* реализовано пять различных алгоритмов: алгоритм обратного распространения, быстрые методы второго порядка – методы сопряженных градиентов и Левенберга–Маркара, а также методы быстрого распространения и Дельта-дельта с чертой (вариация метода обратного распространения). Все эти методы являются итеративными, то есть последовательно приближаются к минимуму, начиная с некоторого начального значения.

Выбор алгоритма обучения

В большинстве случаев вначале следует испытать метод сопряженных градиентов – в этом случае обучение происходит достаточно быстро (иногда на порядок быстрее, чем, например, методом обратного распространения).

Последний метод следует предпочесть в случае, когда в очень сложной задаче требуется быстро найти *удовлетворительное* решение или когда данных очень много (порядка десятков тысяч наблюдений).

Метод Левенберга–Маркара для некоторых типов задач может оказаться эффективнее метода сопряженных градиентов, но его можно использовать только

в сетях с одним выходом, квадратичной функцией ошибок и не очень большим числом весов. Фактически область его применения ограничивается небольшими по объему задачами нелинейной регрессии.

Итеративное обучение. Итеративный алгоритм обучения последовательно проходит ряд так называемых *эпох – Epochs,* на каждой из которых на вход сети подается наблюдение за наблюдением – весь набор обучающих данных, вычисляются ошибки и по ним подправляются веса сети.

Известно, что итеративные алгоритмы подвержены нежелательному явлению переобучения (когда сеть хорошо учится выдавать те же выходные значения, что и в обучающем множестве, но оказывается не способна обобщить закономерность на новые данные). Поэтому качество работы сети следует проверять на каждой эпохе с помощью специального проверочного множества (для этого нужно выбрать опцию *Кросс-проверка – Cross verification* в диалоговом окне обучения).

Контроль обучения

За ходом обучения можно следить в окне *График ошибки обучения – Training Error Graph* (оно открывается из меню *Статистики – Statistics),* где на графике отображается среднеквадратичная ошибка на обучающем множестве на данной эпохе.

Если выбрана опция *Кросс-проверка – Verification,* выводится также среднеквадратичная ошибка на проверочном множестве.

С помощью расположенных под графиком элементов управления можно менять масштаб изображения, а если график целиком не помещается в окне, под ним появляются линейки прокрутки.

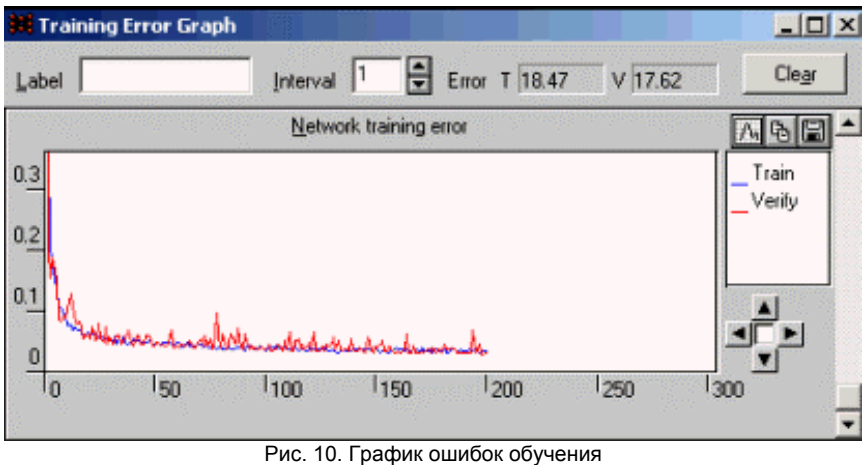


Рис. 10. График ошибок обучения

Если требуется сопоставить результаты различных этапов обучения, нажмите кнопку *Переустановить – Reinitialize* в окне обучения, а затем еще раз нажмите кнопку *Обучить – Train* (повторное нажатие кнопки *Обучить – Train* без *Переустановки – Reinitialize* просто продолжит обучение сети с того места, где оно было прервано).

Чтобы облегчить сравнение результатов, имеется возможность перед нажатием кнопки *Обучить – Train* задать для графика *Метку – Label:* тогда очередная линия будет рисоваться новым цветом, а информация о ней будет добавлена в легенду в правой части окна. По окончании обучения график можно переслать в STATISTICA (кнопка Щ).

На графике обучения можно легко заметить эффект переобучения. Вначале ошибка обучения и проверочная ошибка убывают. При возникновении эффекта *переобучения* ошибка обучения продолжает убывать, а ошибка проверки растет. Рост проверочной ошибки сигнализирует о начале переобучения. Если наблюдается переобучение, то обучение следует прервать, нажав кнопку *Стоп – Stop* в окне обучения или нажав клавишу ESCAPE.

Можно также задать автоматическую остановку программы *STNeural Networks с* помощью условий остановки, которые задаются в окне *Условия остановки – Stopping Conditions* (доступ к которому происходит через меню *Обучение-дополнительные – Train-Auxihary).*

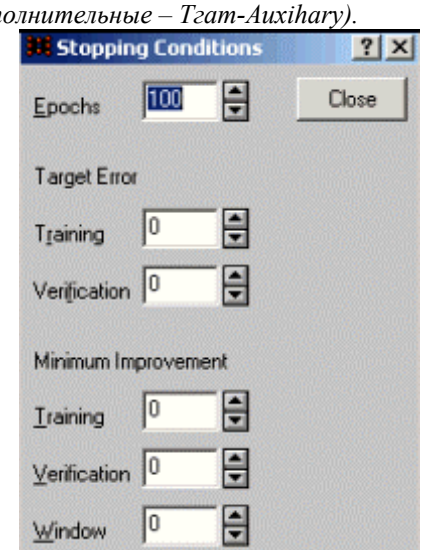


Рис. 11. Задание условий остановки обучения

Кроме максимального числа эпох, отводимого на обучение, можно потребовать, чтобы обучение прекращалось при достижении определенного уровня ошибки или когда ошибка перестает уменьшаться на определенную величину (остановка по невязке).

Борьба с переобучением

Самое лучшее средство борьбы с переобучением – задать нулевой уровень минимального улучшения. Однако поскольку при обучении присутствует шум, обычно не рекомендуется прекращать обучение лишь потому, что на очередной эпохе ошибка ухудшилась. Поэтому в диалоге *Stopping Conditions – Условия остановки* имеется специальное *Окно – Window,* в котором задается число эпох, на протяжении которых должно наблюдаться ухудшение, и только после этого обучение будет остановлено. Обычно в этом окне устанавливают значение 5.

Сохранение лучшей сети

Вы можете восстановить наилучшую конфигурацию сети из всех, полученных в процессе обучения, с помощью опции *Лучшая сеть – Best Network...* (меню *Обучение-дополнительные – Train-Auxiliary).*

Если опция *Сохранить лучшую – Retain Best* включена, программа *Neural Networks* автоматически сохраняет наилучшую из сетей, полученную в ходе обучения.

Если включена опция *Учитывать все прогоны обучения – Span training runs,* то это делается и для прогонов обучения различных сетей.

Таким образом, программа *Neural Networks* автоматически хранит наилучший результат всех экспериментов.

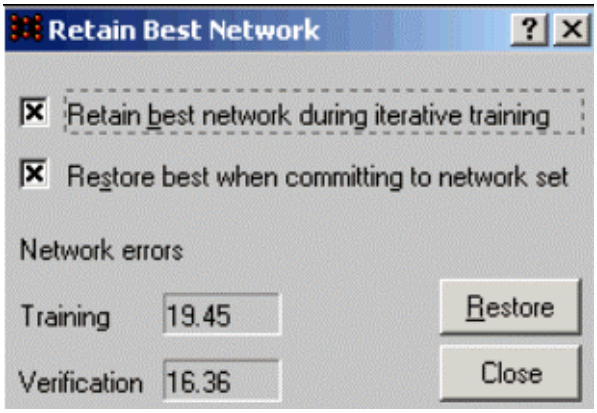


Рис. 12. Опция: лучшая сеть

Можно также установить *Штраф за элемент – Unit Penalty с* тем, чтобы при сравнении штрафовать сети с большим числом элементов (наилучшая сеть обычно представляет собой компромисс между качеством работы и размером).

Наилучшая сеть

Для того чтобы вызвать наилучшую сеть, нажмите кнопку *Восстановить – Restore* Такая возможность, как правило, очень помогает, однако ясно, что она отрицательно сказывается на эффективности (программа *Neural Networks* должна копировать и сохранять сеть каждый раз, когда достигается улучшение), поэтому в некоторых случаях имеет смысл отключить эту опцию.

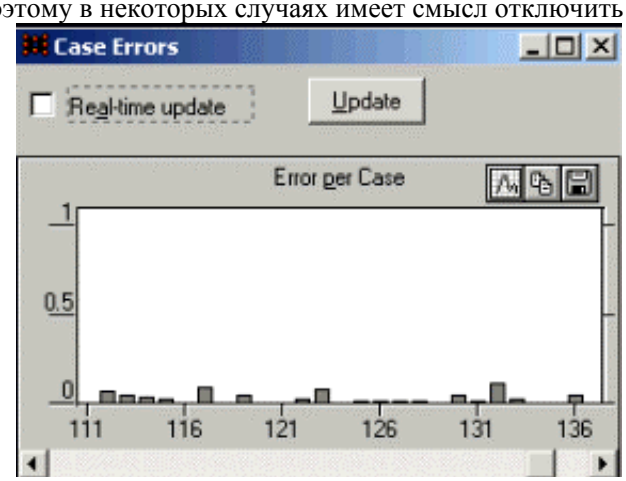


Рис. 13. Ошибки обучения

Ошибки сети (во время и по результатам обучения) можно наблюдать также в окне *Ошибки наблюдений – Case Errors* (доступ – через меню *Статистики – Statistics).* Здесь выводится диаграмма ошибок для отдельных наблюдений Установив опцию *Пересчитывать по ходу – Real-time Update,* можно следить за изменением ошибок от эпохи к эпохе.

Обратное распространение

Перед применением алгоритма обратного распространения необходимо задать значения ряда управляющих параметров.

Наиболее важными параметрами являются скорость обучения, инерция и перемешивание наблюдений в процессе обучения.

Скорость обучения – Learning rate задает величину шага при изменении весов: в случае недостаточной скорости алгоритм медленно сходится, а при слишком большой алгоритм неустойчив. К сожалению, величина наилучшей скорости зависит от конкретной задачи; для быстрого и грубого

обучения подойдут значения от 0,1 до 0,6; для достижения точной сходимости требуются гораздо меньшие значения (например, 0,01 или даже 0,001, если эпох много тысяч).

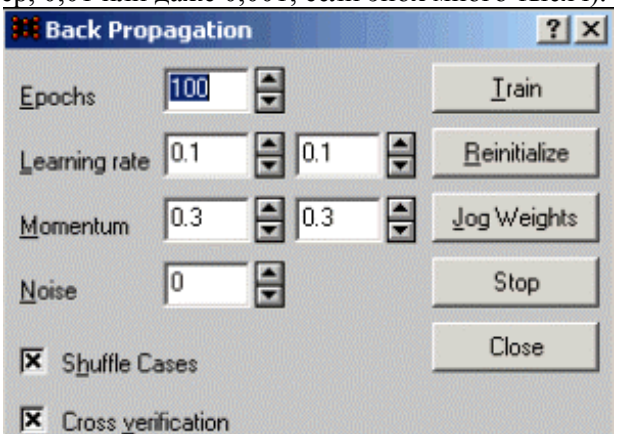


Рис. 14. Опции алгоритма обратное распространение

Иногда полезно уменьшить скорость обучения. В программе *Neural Networks* можно задать начальное и конечное значения скорости, и по мере обучения производится интерполяция между ними. Начальная скорость задается в левом поле, конечная – в правом.

Инерция – Momentum помогает алгоритму, когда он застревает в низинах и локальных минимумах. Этот коэффициент может иметь значения в интервале от нуля до единицы.

Реально «правильное» значение можно найти только опытным путем, и для этого в STATISTICA имеются все возможности.

Перемешивание наблюдений

Перемешивать порядок наблюдения обычно рекомендуется, когда для решения задачи используется метод обратного распространения, поскольку этот способ уменьшает вероятность того, что алгоритм застрянет в локальном минимуме, а также уменьшает эффект переобучения. Чтобы воспользоваться такой возможностью, установите опцию *Перемешивать наблюдения – Shuffle Cases.*

При работе с нейросетями следует помнить о важном моменте – процессировании, или преобразовании, данных.

Пре/постпроцессирование

Передаточная функция для каждого элемента сети обычно выбирается так, чтобы ее входной аргумент мог принимать произвольные значения, а выходные значения лежали бы в строго ограниченном диапазоне. При этом возможен эффект насыщения, когда элемент сети оказывается чувствительным лишь к входным значениям, лежащим в некоторой ограниченной области.

На этом рис. 15 представлена логистическая функция.

Логистическая функция является гладкой, ее производная легко вычисляется, что существенно для алгоритмов минимизации на этапе обучения сети (в этом также кроется причина того, что ступенчатая функция для этой цели практически не используется). Если применяется логистическая функция для вычисления выходного сигнала (см. формулу (1)), то выходное значение всегда лежит в интервале (0;1), а область чувствительности для *входов* чуть шире интервала (-2;+2).

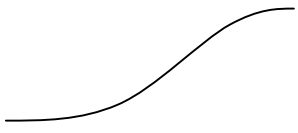


Рис. 15. Логистическая функция

Чтобы согласовать вход-выход при решении задач методами нейронных сетей, требуются этапы предварительной обработки (Bishop, (1995) Neural Networks with Pattern recognition, Oxford: University Press). Эти преобразования включают, в частности, шкалирование и преобразование категориальных переменных в числовую форму.

Шкалирование

Числовые значения должны быть приведены в масштаб, подходящий для сети. В пакете *Нейронные сети ST*реализованы алгоритмы минимакса и среднего/стандартного отклонения, которые автоматически находят масштабирующие параметры для преобразования числовых

значений в нужный диапазон.

В некоторых случаях более подходящим может оказаться нелинейное шкалирование (например, если заранее известно, что переменная имеет экспоненциальное распределение, есть смысл взять ее логарифм). Можно шкалировать переменную с помощью средств преобразования данных в STATISTICA, а затем работать с ней в модуле *Нейронные сети ST.*

Номинальные переменные

Номинальные, или категориальные, переменные преобразовываются в числовую форму (например, *Муж = 0, Жен = 1).* Для кодирования *многомерных* номинальных переменных используется так называемый метод 1-из-M, так как при наивном способе кодирования, например *Собака = 0, Овца = 1, Кошка = 2,* может возникнуть ложное упорядочивание значений категориальной переменной: *Овца* окажется чем-то средним между *Собакой* и *Кошкой.*

В методе 1-Н3-N одна номинальная переменная представляется несколькими числовыми переменными. Количество числовых переменных равно числу возможных значений номинальной переменной; при этом всякий раз ровно одна из Af переменных принимает ненулевое значение (например, *Собака = {1,0,0}, Овца = {0,1,0}, Кошка = {0,0,1}).* Заметим, что этот метод кодирования требует большого количества числовых переменных, если номинальная переменная принимает много значений.

Оценка качества работы сети

После того как сеть обучена, стоит проверить, насколько хорошо она работает. Для этого доступны несколько показателей.

Среднеквадратичная ошибка, которая выдается в окне *График ошибки обучения – Training Error Graph,* представляет лишь грубую меру производительности. Более полезные характеристики выводятся в окнах *Статистики классификации – Classification Statistics и Статистики регрессии – Regression Statistics* (доступ к обоим происходит через меню *Статистики – Statistics).*

Окно *Статистики классификации – Classification Statistics* применяется для номинальных выходных переменных. Здесь выдаются сведения о том, сколько наблюдений каждого класса (классы соответствуют номинальным значениям) из файла данных было классифицировано правильно, сколько неправильно и сколько не классифицировано, а также приводятся подробности об ошибках классификации. Обучив сеть, нужно просто открыть это окно и нажать в нем кнопку *Запуск –Run.* Статистики выдаются раздельно для обучающего, проверочного и тестового множеств *(внимание:* чтобы увидеть тестовые статистики, нужно прокрутить таблицу вправо). В верхней части таблицы приводятся суммарные статистики (общее число наблюдений в каждом классе, сколько из них классифицировано правильно, неправильно и не классифицировано), а в нижней части – кросс-результаты классификации (сколько наблюдений из данного столбца было отнесено к данной строке).

Окно *Статистики регрессии – Regression Statistics* действует в случае числовых выходных переменных. В нем суммируется точность регрессионных оценок.

Наиболее важной статистикой является S. D. ratio – отношение стандартного отклонения ошибки прогноза к стандартному отклонению исходных данных.

Если бы у нас вообще не было входных данных, то лучшее, что мы могли бы взять в качестве прогноза для выходной переменной, – это ее выборочное среднее, а ошибка такого прогноза была бы равна стандартному отклонению выборки.

Если нейронная сеть работает результативно, мы вправе ожидать, что ее средняя ошибка на имеющихся наблюдениях будет близка к нулю, а стандартное отклонение этой ошибки будет меньше стандартного отклонения выборочных значений (иначе сеть давала бы результат не лучше, чем простое угадывание).

Таким образом, если S. D. ratio значительно меньше единицы, то сеть эффективна.

Величина, равная единице минус S. D. ratio, является долей объясненной дисперсии модели.

Перейдем к работе с нейронными сетями в системе STATISTICA

Для того чтобы понять, как решаются задачи прогнозирования с помощью нейро-сетей, мы будем использовать файл *series_g.sta,* для задач классификации используем файл *iris.sta.*

Диалог в модуле Нейронные сети STATISTICA

Мы продолжаем работать с файлом *Series j>.sta.* Это классический файл данных, обычно используемый для тестирования методов прогнозирования (см., например, книгу Бокс Дж., Дженкинс Г. Д. Анализ временных рядов и прогнозирование. М.: Мир, 1974).

Шаг 1. Откройте файл данных *Senes_gsta* из папки *Examples* Данные содержат значения одной переменной месячные перевозки пассажиров Как мы уже заметили, для нейронной сети эта переменная будет служить *входной/выходной* (так как мы прогнозируем будущие значения ряда на основе предыдущих значений)

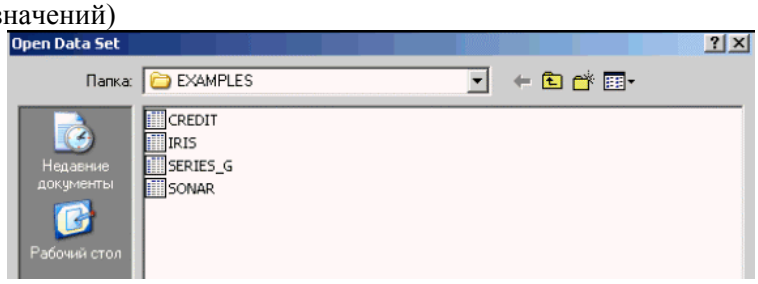


Рис 16 Выбор файла

Поэтому задайте тип переменной как *входная/выходная*

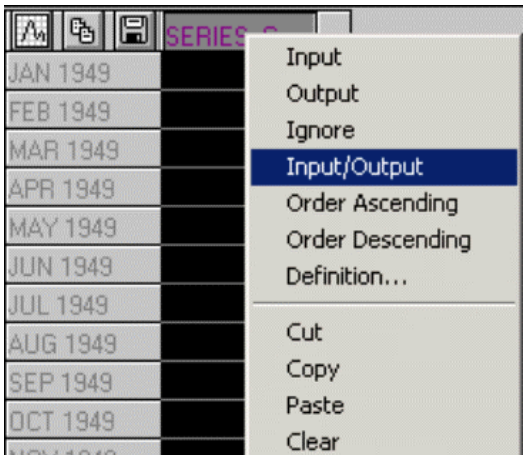


Рис 17 Задание типа переменной

Для этого выделите переменную в открытом файле данных (щелчком на заголовке столбца) Затем нажмите правую кнопку мыши и выберите из появившегося контекстного меню пункт *Входная/выгодная – Input/Output* Имя переменной высветится зеленым цветом

Шаг 2. С помощью мыши выберите команду *Сеть – Network* из меню *Файл-Новый – File–New*

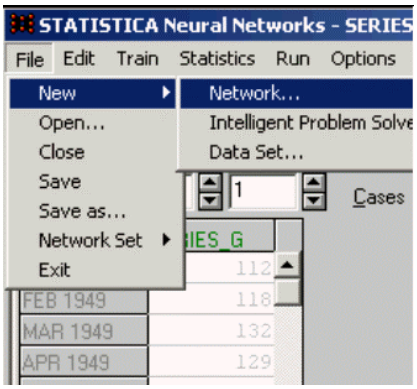


Рис 18 Построение сети

На экране появится диалоговое окно *Создать сеть – Create Network.*

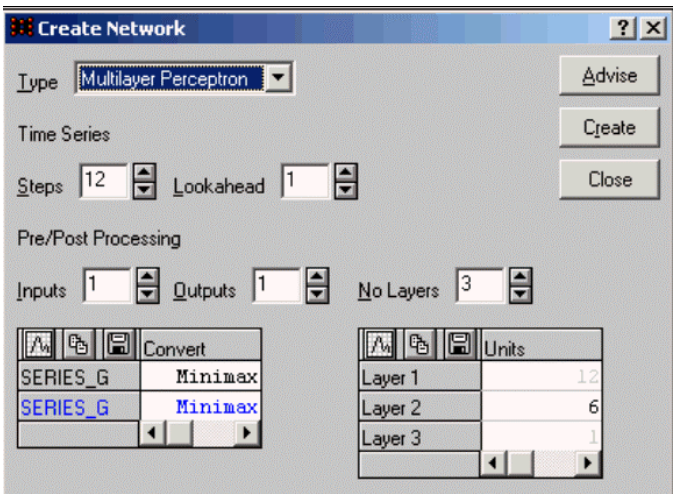


Рис. 19. Задание параметров персептрона

В поле *Type – тип* выберите тип сети *Многослойный персептрон – Multilayer Perceptron* и сделайте следующие установки-Входы – Inputs = 1, Выходы – Outputs = 1.

Задайте число слоев равным трем, *No Layers = 3.* Выберите трехслойный персептрон. Задайте параметр *Временное окно – Steps* равным *12* (данные представляют собой ежемесячные авиаперевозки с присутствующей в них сезонной составляющей), а параметр *Горизонт – Lookahead* – равным /. После этого нажмите кнопки *Совет – Advise* и *Создать – Create.* На экране появится схема трехслойного персептрона. Этот персептрон имеет 12 входов.

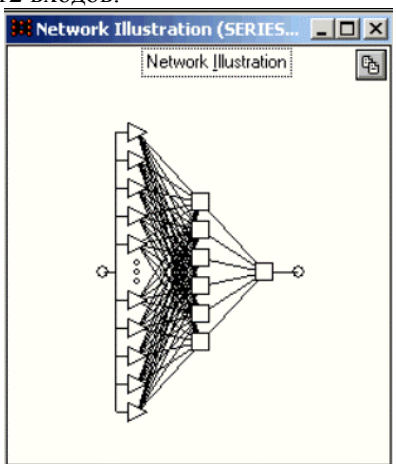


Рис. 20. Трехслойный персептрон

Шаг 3. Обучение сети. Структура сети определена. Теперь ее нужно обучить.

В файле данных выберите *66 обучающих – Training* и *66 контрольных – Verification* наблюдений. Всего в файле содержится 144 наблюдения. Первые 12 резервируются для построения прогноза на первом шаге.

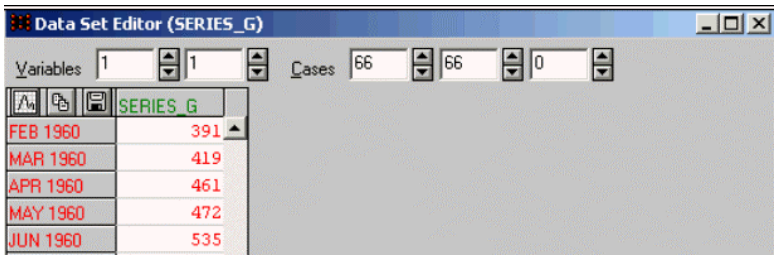


Рис. 21. Из файла данных выбрано 66 обучающих и 66 контрольных наблюдений

Далее воспользуйтесь опцией *Shuffle – Перемешать.*

Заметьте, во временном ряде наблюдения упорядочены во времени, поэтому при перемешивании нельзя пользоваться функцией *Сгруппировать множества – Group Sets.*

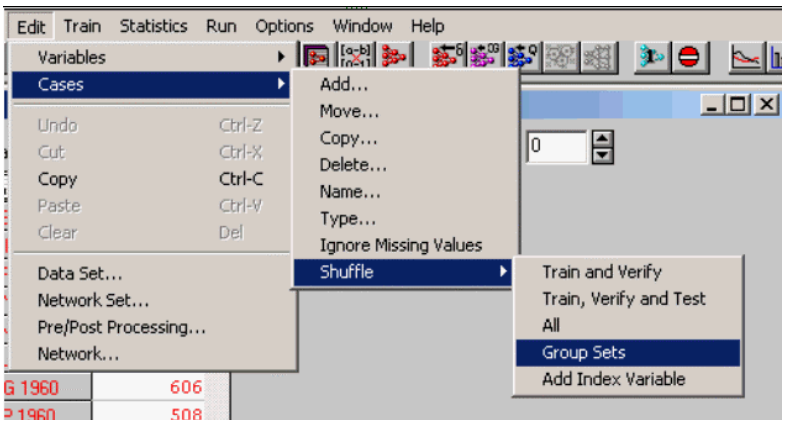


Рис. 22. Выбор функции Shuffle – Перемешивание позволяет случайным образом перемешать наблюдения в процедуре обучения

Опция перемешивания позволяет распределить обучающие и контрольные наблюдения в файле данных. Для обучения сети воспользуемся методом сопряженных градиентов.

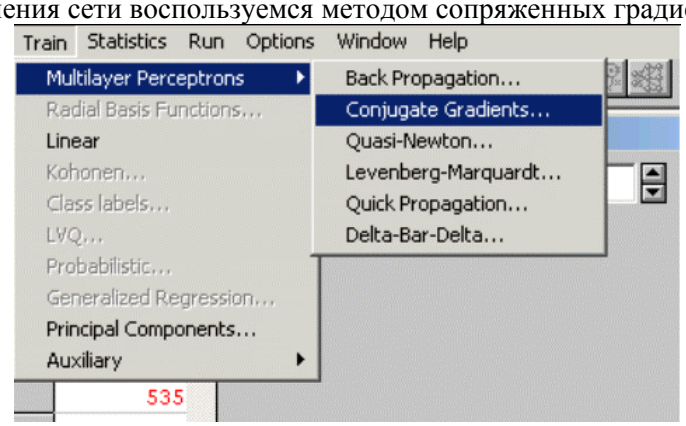


Рис. 23. Выбор метода сопряженных градиентов для обучения сети

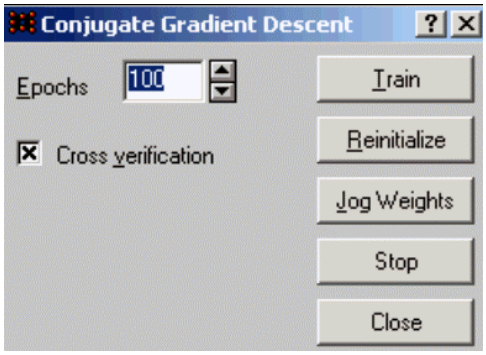


Рис. 24. Окно минимизации методом сопряженных градиентов

Обратите внимание на кнопку *Переустановить – Reinitialize:* она позволяет случайным образом выбрать *новые* начальные значения свободных параметров сети и провести обучения, исходя из этих установок.

Опция *Кросс-проверка – Cross verification* позволяет провести обучение с кросс-проверкой (проверять сеть на контрольном множестве на каждой эпохе обучения).

Шаг 4. Проекция временного ряда.

Проекция ряда строится следующим образом:

О сеть обрабатывает начальный набор значений (первые 12 наблюдений) и выдает прогноз;

О первое наблюдение из исходного набора отбрасывается, вместо него ставится прогноз, полученный на первом шаге;

О по новому набору входных значений строится следующий прогноз и т. д.

Процесс проектирования можно продолжать неограниченно.

Для построения проекции откройте окно *Проекция временного ряда – Time Series Injection* командой *Проекция временного ряда – Time Series Projection...* меню *Запуск – Run.*

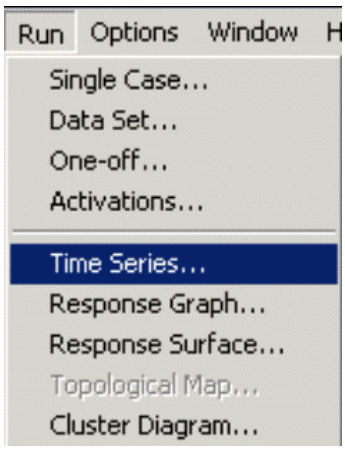


Рис. 25. Открытие окна Проекция временного ряда

В модуле *Нейронные сети* можно построить проекцию временного ряда с некоторого наблюдения текущего набора (см. опции окна). Выбирая опции окна, можно получить разнообразные проекции и прогноз ряда с помощью построенной и обученной сети.

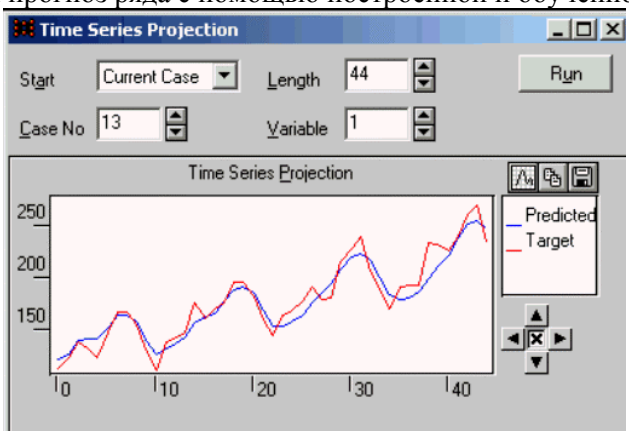


Рис. 26. Проекция временного ряда на 44 наблюдения

Чтобы оценить качество работы сети, откройте окно *Статистики регрессии Regression Statistics* и нажмите кнопку *Запуск – Run*

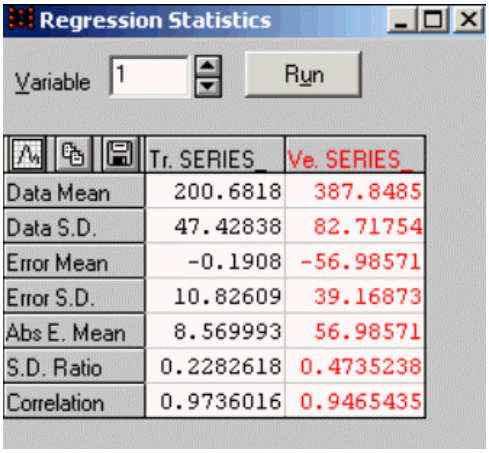


Рис 27 Описательные статистики позволяют оценить качество прогноза

Шаг 5. Для того чтобы построить прогноз на 1 шаг с помощью обученной сети, выберите команду меню *Run – Single Case*

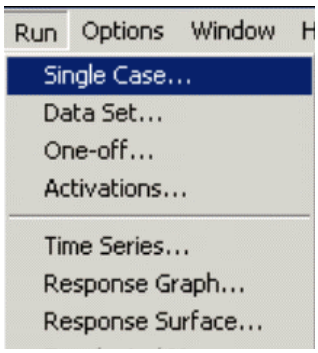


Рис 28. Выбор команды Run Single Case

На экране появится диалоговое окно *Run Single Case* В поле *Case No* введите номер наблюдения, для которого нужно построить прогноз **(В исходной таблице необходимо предварительно добавить еще один ”Case” в конце списка. Его номер будет “145”)**, и нажмите кнопку *Run*

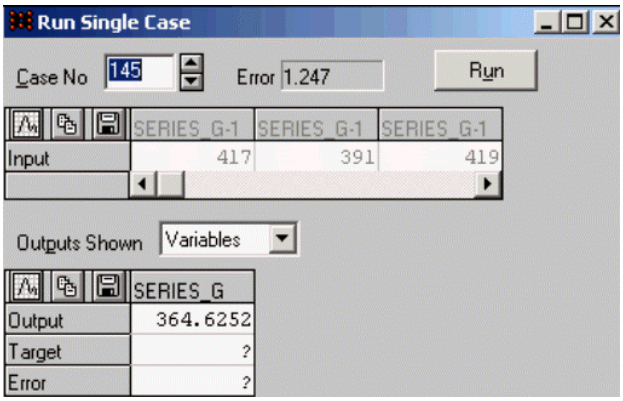


Рис. 29 Прогноз на один шаг вперед, построенный с помощью обученной сети

В строке *Output* появится прогноз ряда на один шаг вперед В строке *Target* стоит знак ?, так как в исходном файле всего 144 наблюдения

Классификация

Для решения задачи классификации воспользуемся файлом данных *Iris.sta* и *Мастером решения задач.*

Это классические данные Фишера, для классификации которых применяется дискриминантный анализ, дающий оптимальное линейное решающее правило. Заметим, что альтернативным вариантом исследования являются деревья классификации.

Мы используем эти данные только в иллюстративных целях: на простых и ясных примерах можно познакомиться с возможностями нейронных сетей по классификации данных.

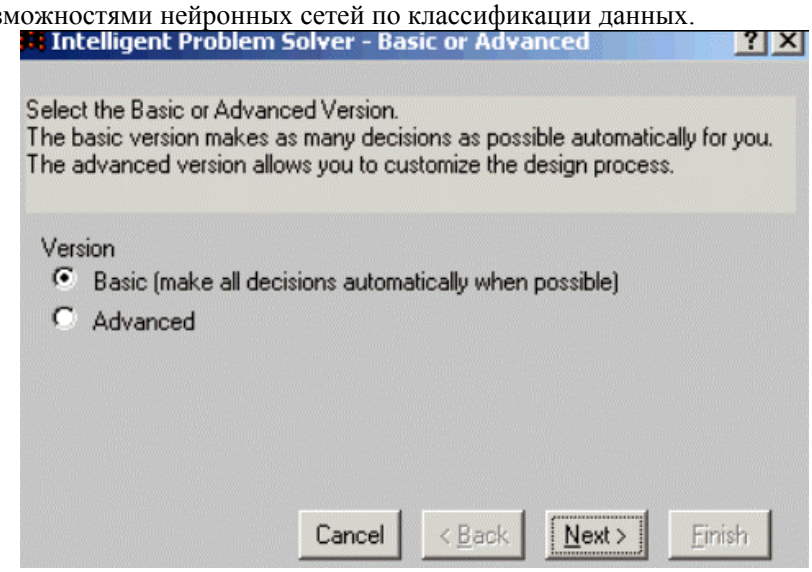


Рис. 30. Мастер решения задач (начало диалога)

Шаг *i.* Откройте файл *Iris.sta.* Первые 4 переменные – это параметры цветков ириса. Категориальная переменная *IRISTYPE* обозначает тип ириса. Измеряя параметры цветка, нужно отнести его к одному из трех типов *(Setosa, Versicol, Virginic).*

Мастер решения задач последовательно открывает диалоговые окна, в которых просит сделать несложный выбор.

Шаг 2. Одно окно уже открыто – это *Problem Type – Тип задачи.* Укажите тип задачи и нажмите кнопку *Next.*

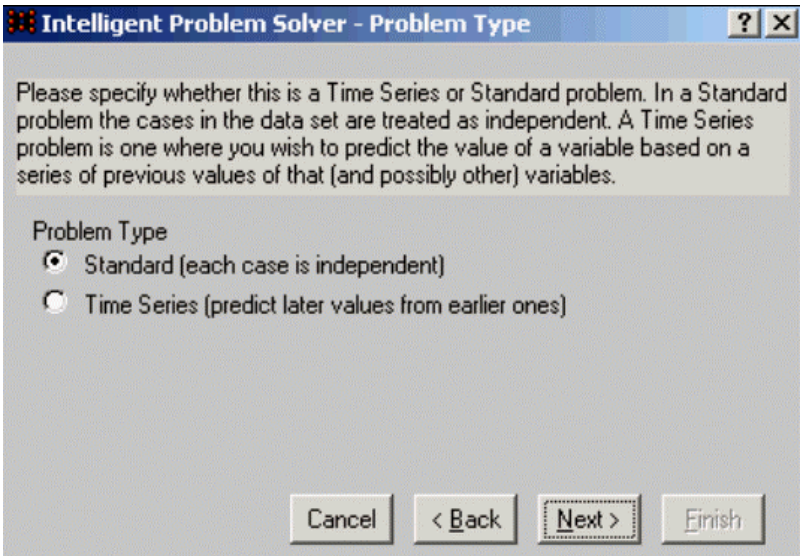


Рис. 31. Выберите стандартный тип

Шаг 3. В следующем окне выберите зависимую переменную.

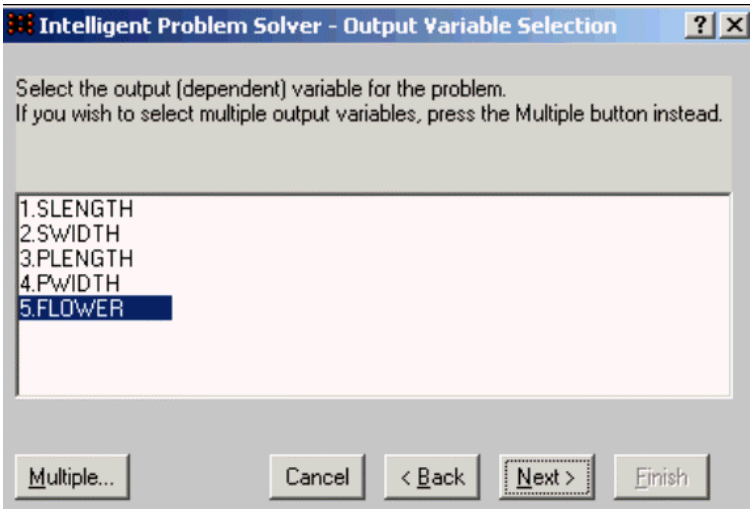


Рис. 32. Выберите переменную flower как выходную (зависимую) переменную

Выходная переменная – номинальная, она принимает три значения: *Setosa, Versicol, Virginia.* Нажмите кнопку *Next.*

Шаг 4. В следующем диалоговом окне выберите входные (независимые) переменные.

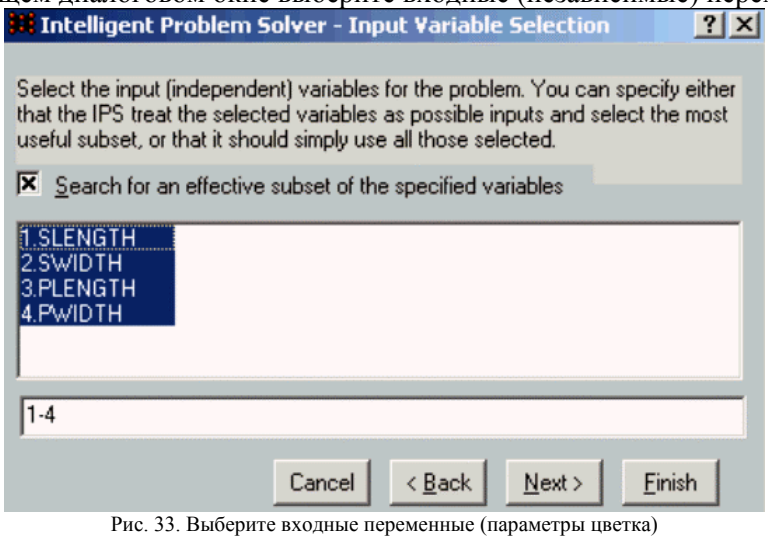


Рис. 33. Выберите входные переменные (параметры цветка)

Нажмите кнопку *Next. Мастер решения* разобьет выборку на обучающую (черный цвет значений), контрольную (синий цвет) и тестовую выборку (красный цвет). Также автоматически будет произведено перемешивание наблюдений.

Шаг 5. На экране появится окно *Duration of Design Process – Длительность поиска.*

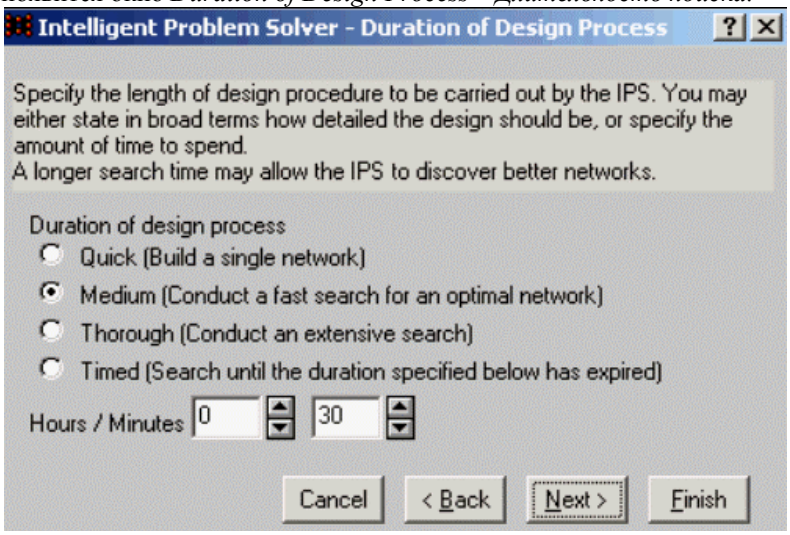


Рис. 34. В окне Длительность поиска можно задать длительность поиска (быстрый, средний, полный, ограниченный по времени)

Шаг 6. Далее на экране появится окно *Saving Networks – Сохранение сетей.* В этом окне можно задать способы сохранения сетей, например, максимальное число сохраняемых сетей, сохранить сети с лучшим качеством решения и т.д.

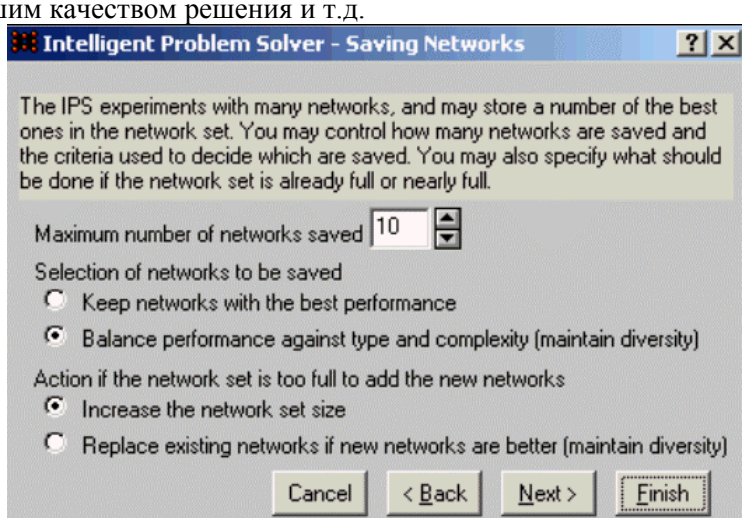


Рис. 35. В этом окне можно задать способы сохранения сетей.

Затем откроется окно, в котором указаны опции представления результатов.

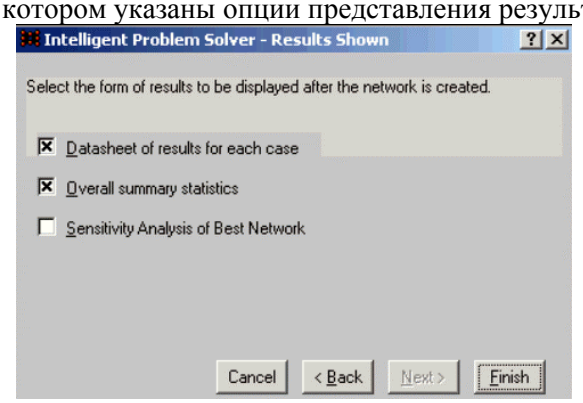
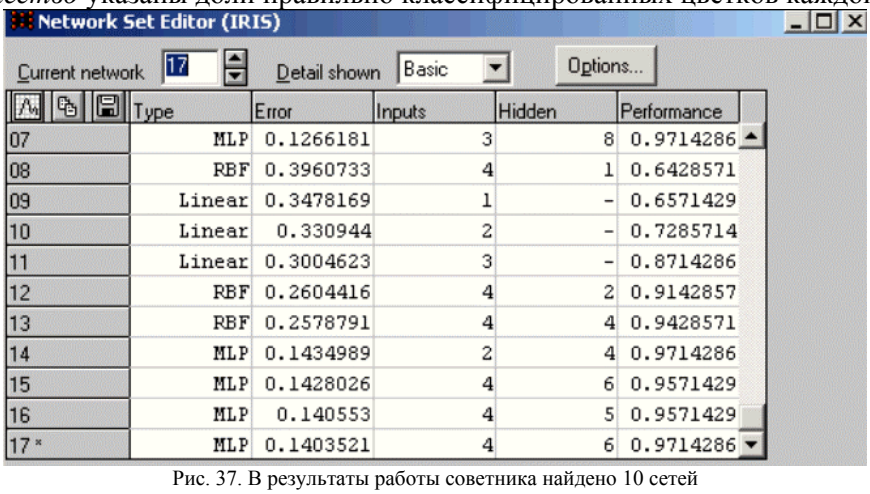


Рис. 36. В этом окне выбираются опции представления результатов

Шаг 7. Нажмите кнопку *Finish.* STATISTICA произведет вычисления и представит результаты в следующем виде таблицы.

В этой таблице показаны 10 лучших сетей, найденных советником. В столбце *Type – Тип* указан тип сетей: *RBF* – радиальные базисные функции, *Linear* – линейные, *MLP* – многослойный персептрон. Далее в таблице результатов идут столбцы: *ошибка, входы, скрытые.* В столбце *Perfomance – Качество* указаны доли правильно классифицированных цветков каждой сетью.



	Type	Error	Inputs	Hidden	Performance
07	MLP	0.1266181	3	8	0.9714286
08	RBF	0.3960733	4	1	0.6428571
09	Linear	0.3478169	1	-	0.6571429
10	Linear	0.330944	2	-	0.7285714
11	Linear	0.3004623	3	-	0.8714286
12	RBF	0.2604416	4	2	0.9142857
13	RBF	0.2578791	4	4	0.9428571
14	MLP	0.1434989	2	4	0.9714286
15	MLP	0.1428026	4	6	0.9571429
16	MLP	0.140553	4	5	0.9571429
17 *	MLP	0.1403521	4	6	0.9714286

Рис. 37. В результаты работы советника найдено 10 сетей

Заключительные комментарии

Указанная в таблице на рис. 37 сеть радиальной базисной функции (RBF) имеет промежуточный слой из радиальных элементов, каждый из которых воспроизводит гауссову поверхность отклика. Сети RBF иногда имеют некоторое преимущество перед сетями MLP. Во-первых, они моделируют любую *нелинейную* функцию с помощью только *одного* промежуточного слоя. Во-вторых, параметры линейной комбинации в выходном слое можно оптимизировать с использованием известных методов линейного программирования. В задачах классификации выходной элемент должен выдавать большой сигнал, если данное наблюдение принадлежит к интересующему нас классу, и слабый – в противоположной ситуации. Имеется и более тонкий способ интерпретации уровней выходного сигнала сети – вероятностный. В этом случае сеть выдает несколько большую информацию, чем просто «да/нет»: она сообщает, с какой вероятностью наблюдение принадлежит данному классу. В модуле *Нейронные сети* имеются методы, позволяющие интерпретировать выходной сигнал сети как вероятность, в результате чего сеть, по существу, учится моделировать плотность вероятности распределения для наблюдений из данного класса.

Линейная модель представляет собой сеть без промежуточных слоев, которая в выходном слое содержит только линейные элементы (то есть элементы с линейной функцией активации). Линейная модель обычно записывается с помощью матрицы *N* x *Nw* вектора смещения размера *N*. Веса соответствуют элементам матрицы, а пороги – компонентам вектора смещения. Сеть умножает вектор входов на матрицу весов, а затем к полученному вектору прибавляет вектор смещения. Можно создать линейную сеть и обучить ее с помощью стандартного алгоритма оптимизации, основанного на псевдообратных матрицах. Тот же алгоритм реализован в модуле *Множественная регрессия* системы STATISTICA. Это самый простой тип сетей. Линейная сеть позволяет сравнить качество построенных сетей. Может оказаться так, что задача успешно решается не только с помощью сложных нейронных сетей, но и простыми линейными методами. Заметим, что в модуле *Нейронные сети* реализованы также другие типы нейронных сетей, например, сети Кохонена, вероятностные сети, обобщенно-регрессионные нейронные сети (GRNN), предназначенные для решения задач регрессии, однако описание этих сетей выходит за рамки данной главы.

Рассмотрим подробнее столбцы таблицы на рис. 37.

Тип – Type. В этом столбце указан *тип* нейронной сети. В большинстве случаев это многослойные персептроны (MLP), радиальные базисные функции (RBF) или линейные сети.

Ошибка – Error. Здесь указана ошибка сети, полученная на *контрольном* подмножестве, которая вычисляется по всем контрольным наблюдениям. Чем меньше значение ошибки, тем лучше качество сети.

Входы – Inputs. В этом столбце указано число входных переменных, используемых нейронной сетью. Заметим, что лучше использовать сеть с меньшим числом входных переменных, если это не ухудшает существенно ее качество по сравнению с сетями, использующими большее количество переменных на входе.

Скрытые – Hidden. Здесь указано число *скрытых* элементов сети. Заметьте, линейные сети не имеют скрытых элементов, поэтому для них в этом столбце указан пропуск.

Качество – Performance. В этом столбце показано *качество* сети, которое определяется по контрольному подмножеству. Для задач классификации качество – это доля *правильно* классифицированных наблюдений. Очевидно, предпочтительнее использовать сети с лучшими показателями качества. Однако заметим, что в задачах классификации меньшее значение ошибки не всегда соответствует лучшему качеству. Иногда сеть может улучшить ошибку на некотором множестве уже правильно классифицированных наблюдений за счет неправильной классификации дополнительного наблюдения. В результате может оказаться, что такой вариант имеет меньшую ошибку и одновременно худшее качество по сравнению с другим вариантом сети.

Лучшая сеть отмечена * (в данном примере это сеть с номером 10, см. рис. 37).

Заметьте, что в набор сетей включены и некоторые сети с плохим качеством (см. например, сеть с номером 2, которая правильно классифицирует лишь 65% наблюдений). На примере таких сетей можно понять, какой результат дают простые модели.

Сети низкого качества легко удалить из набора. Чтобы сделать это, выделите сеть, щелкнув на ней мышью, а затем нажмите правую кнопку мыши и выберите из появившегося меню команду *Удалить – Delete.* Выделенная сеть будет удалена.

Можно сделать выделенную нейронную сеть активной с помощью команды всплывающего меню *Выбрать – Select.*

Если набор нейронных сетей заполнен, программа *ST Neural Networks* должна определить, какие из имеющихся сетей заменять вновь создаваемыми. Нажмите кнопку *Опции – Options...в диалоговом окне Редактор набора сетей – Network Set Editor.*

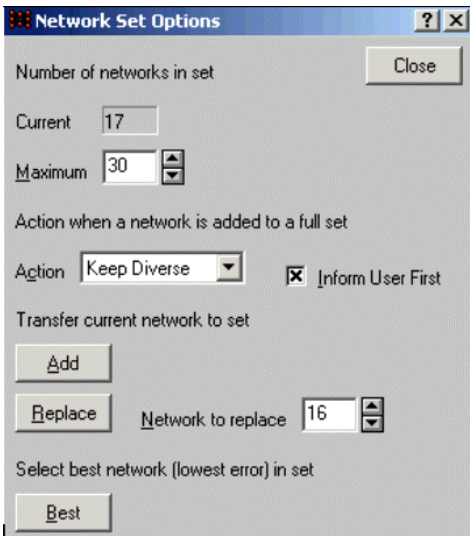


Рис. 38. Настройка параметров набора сетей

На экране появится диалоговое окно *Параметры набора сетей – Network Set Options.*

В этом окне задается максимальное количество сетей в наборе. По умолчанию максимальный размер составляет 30 нейронных сетей

Если вы хотите, чтобы программа сообщала вам об удалении сети, включите режим *Вначале сообщать пользователю – Inform User First.*

Кроме того, взглянув на окно (рис. 38), мы видим, что при попытке добавить сеть в уже полный набор программа по умолчанию будет использовать режим *Сохранять разнообразие – Keep Diverse.*. В этом случае решение о том, заменить ли новой сетью какую-либо из существующих, будет принято с учетом необходимости сохранить в наборе разнообразные соотношения между качеством и сложностью сетей (при этом всегда сохраняется лучшая сеть каждого типа, независимо от ее сложности).

Установив нужные значения параметров набора сетей, нажмите кнопку *Закрыть – Close.*

Если вы не хотите удалять некоторую сеть из списка, заблокируйте ее командой *Блокировать – Lock* из выпадающего меню правой кнопки мыши. Заблокированные сети выделяются голубым цветом и никогда не удаляются, независимо от их качества. Чтобы разблокировать сеть, используйте команду *Разблокировать – Unlock*

Иногда требуется изменить порядок сетей в списке, например, сгруппировать их по типам или рассортировать по величине ошибки или качеству. Чтобы осуществить это, щелкните правой кнопкой мыши на названии столбца и выберите из выпадающего меню команду *Сортировать по возрастанию – Sort Ascending* или *Сортировать по убыванию – Sort Descending*

Для исследования важности входных переменных обученной сети полезен *анализ чувствительности.*

Представьте, вы имеете обученную сеть и вам нужно знать, как изменится качество работы сети, если некоторые *входные* переменные будут удалены. Чтобы ответить на этот вопрос, выберите команду *Чувствительность – Sensitivity.* из выпадающего меню *Статистики – Statistics.*

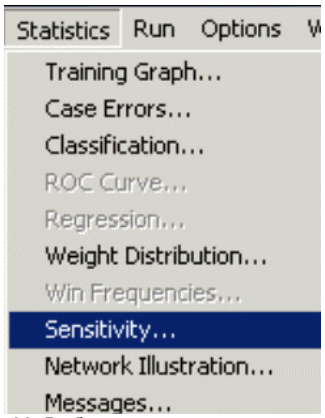


Рис. 39. Выбор анализа чувствительности

В появившемся окне *Анализ чувствительности* нажмите кнопку *Обновить Update.*

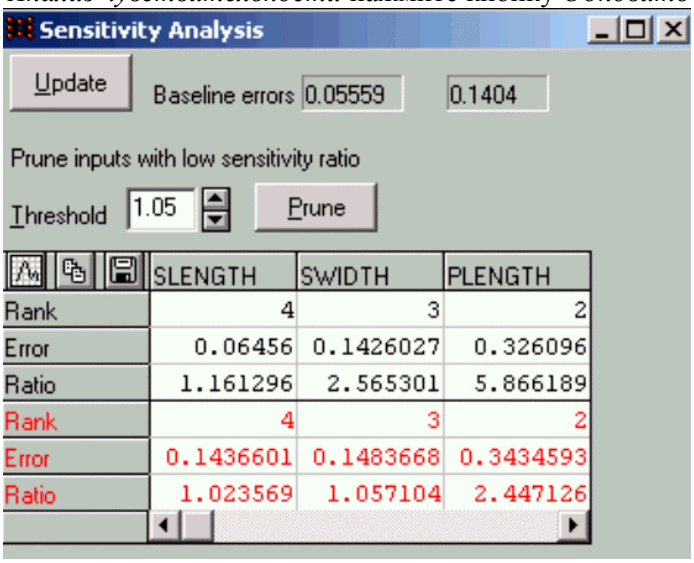


Рис. 40. Диалоговое окно Анализ чувствительности

Программа построит таблицу, в которой будет показана чувствительность сети по отношению к *каждой* переменной. Посмотрите на таблицу (рис. 40).

В таблице приводятся три показателя: *Ранг – Rank, Ошибка – Error и Отношение –Ratio.* Показатели чувствительности даются отдельно для обучающего (первые три строки) и контрольного набора наблюдений. Столбцы таблицы – это переменные исходного файла данных.

Вначале рассмотрим строку *Error.* Для каждой переменной значение *Error* показывает, каким будет качество сети, если данную переменную исключить из числа входных переменных. Очевидно, более важным для классификации переменным отвечают большие значения ошибок.

Отношение – Ratio представляет собой отношение между значением в строке *Ошибка – Error* и основной ошибкой *(Baseline Error). Baseline Error* вычисляется для сети со *всеми* входными переменными. Если *Отношение – Ratio* меньше единицы, то *исключение* данной переменной *улучшает* качество работы сети.

В строке *Ранг – Rank* переменные просто ранжированы в порядке убывания ошибки.

Упражнение. **Исследуйте данные об ирисах и найдите параметры цветов, наиболее важные для классификации. Сравните результаты, полученные с помощью нейронных сетей, с результатами классических методов классификации.**

Заметим, что для экспериментирования с набором входных переменных в SNN имеются *Алгоритмы отбора входных переменных – Feature Selection Algorithms,* чтобы проверять различные комбинации входных переменных и строить так называемые вероятностные сети, используемые для поиска лучшего набора входных переменных.

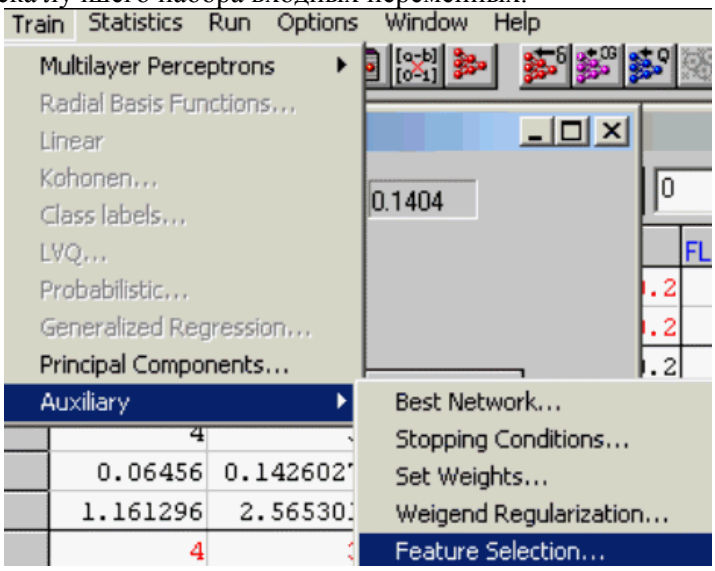


Рис. 41. Выбор алгоритма отбора входных переменных нейронной сети

Эти алгоритмы, включающие в себя пошаговое включение, пошаговое исключение входных переменных и так называемый генетический алгоритм отбора входных переменных, иногда позволяют найти варианты, пропущенные процедурой *Intelligent Problem Solver*.

Упражнение, Постройте с помощью нейронных сетей прогноз продаж бензина (см данные в приложении 1) и сравните с результатами классических методов прогнозирования.