Глава 2.

Решение задач нейронными сетями

А.Н.Горбань

Вычислительный центр СО РАН в г.Красноярске[1]

Нейронные сети могут все. Но точно также «все» могут машины Тьюринга, интерполяционные многочлены, схемы Поста, ряды Фурье, рекурсивные функции, дизъюнктивные нормальные формы, сети Петри. В предыдущей главе было показано, что с помощью нейронных сетей можно сколь угодно точно аппроксимировать любую непрерывную функцию и имитировать любой непрерывный автомат. Это и слишком много - никому не нужен столь широкий класс функций, и слишком мало, так как далеко не все важные задачи ставятся как задачи аппроксимации.

Другое дело, что в конечном итоге решение практически любой задачи можно описать, как построение некоторой функции, перерабатывающей исходные данные в результат, но такое очень общее описание не дает никакой информации о способе построения этой функции. Мало иметь универсальные способности - надо для каждого класса задач указать, как применять эти способности. Именно здесь, при рассмотрении методов настройки нейронных сетей для решения задач должны выявиться реальные рамки их применимости. Конечно, такие реальные рамки изменяются со временем из-за открытия новых методов и решений.

В данной главе описано несколько базовых задач для нейронных сетей и основных или исторически первых методов настройки сетей для их решения:

1. Классификация (с учителем) (персептрон Розенблатта [1-3]).
2. Ассоциативная память (сети Хопфилда [4-7]).
3. Решение систем линейных уравнений (сети Хопфилда [8]).
4. Восстановление пробелов в данных (сети Хопфилда).
5. Кластер-анализ и классификация (без учителя) (сети Кохонена [9-12]).

Начнем мы, однако, не с сетей, а с систем, состоящих из одного элемента.

1. Настройка одноэлементных систем для решения задач

Даже системы из одного адаптивного сумматора находят очень широкое применение. Вычисление линейных функций необходимо во многих задачах. Вот неполный перечень «специальностей» адаптивного сумматора:

1. Линейная регрессия и восстановление простейших закономерностей [13-14];
2. Линейная фильтрация и адаптивная обработка сигналов [15];
3. Линейное разделение классов и простейшие задачи распознавания образов [16-18].

Задача линейной регрессии состоит в поиске наилучшего линейного приближения функции, заданной конечным набором значений: дана выборка значений вектора аргументов x^1 , ..., x^m, заданы значения функции F в этих точках: $F(x^i)=f_i$, требуется найти линейную (неоднородную) функцию $\varphi(x)=(\alpha,x)+\alpha_0$, ближайшую к F. Чтобы однозначно поставить задачу, необходимо доопределить, что значит «ближайшую». Наиболее популярен *метод наименьших квадратов*, согласно которому φ ищется из условия

$$\sum_{i=1}^{m}(F(x^i)-\varphi(x^i))^2 \to \min. \qquad (1)$$

Необходимо особенно подчеркнуть, что метод наименьших квадратов не является ни единственным, ни наилучшим во всех отношениях способом доопределения задачи регрессии. Его главное достоинство - квадратичность минимизируемого критерия и линейность получаемых уравнений на коэффициенты φ.

Явные формулы линейной регрессии легко получить, минимизируя квадратичный критерий качества регрессии. Обозначим

$$\Delta_i = (F(x^i)-\varphi(x^i));\ H=\sum_{i=1}^{m}\Delta_i^2=\sum_{i=1}^{m}(F(x^i)-\varphi(x^i))^2=$$

$$=\sum_{i=1}^{m}(f_i-(\alpha,x^i)-\alpha_0)^2.$$

Найдем производные минимизируемой функции H по настраиваемым параметрам:

$$\frac{\partial H}{\partial \alpha_j}=\sum_{i=1}^{m}\Delta_i x_j^i,\ (j=1,\ldots,n);\ \frac{\partial H}{\partial \alpha_0}=\sum_{i=1}^{m}\Delta_i.$$

где x_j^i - j-я координата вектора x^i.

---

[1] 660036, Красноярск-36, ВЦК СО РАН. E-mail: gorban@cc.krascience.rssi.ru

Приравнивая частные производные H нулю, получаем уравнения, из которых легко найти все α_j (j=0,...,n). Решение удобно записать в общем виде, если для всех i=1,...,m обозначить $x_0^i \equiv 1$ и рассматривать n+1-мерные векторы данных x^i и коэффициентов α. Тогда

$$\Delta_i = f_i - (\alpha, x^i),\ \partial H / \partial \alpha_j = \sum_{i=1}^{m} \Delta_i x_j^i \ \ (j = 0,1,\ldots,n).$$

Обозначим p n+1-мерный вектор с координатами $p_j = \frac{1}{m}\sum_{i=1}^{m} f_i x_j^i$; Q - матрицу размером (n+1)×(n+1) с элементами $q_{jk} = \frac{1}{m}\sum_{i=1}^{m} x_j^i x_k^i$.

В новых обозначениях решение задачи линейной регрессии имеет вид:

$$\varphi(x)=(\alpha,x),\ \alpha=Q^{-1}p. \qquad (2)$$

Приведем это решение в традиционных обозначениях *математической статистики*. Обозначим M_o среднее значение j-й координаты векторов исходной выборки:

$$M_j = \frac{1}{m}\sum_{i=1}^{m} x_j^i .$$

Пусть **M** - вектор с координатами M_o. Введем также обозначение s_j для выборочного среднеквадратичного отклонения:

$$s_j = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_j^i - M_j)^2}$$

Величины s_j задают естественный масштаб для измерения j-х координат векторов x. Кроме того, нам потребуются величина s_f и коэффициенты корреляции f с j-ми координатами векторов x - r_{fj}:

$$s_f = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(f_i - M_f)^2},\ M_f = \frac{1}{m}\sum_{i=1}^{m} f_i,\ r_{fj} = \frac{\frac{1}{m}\sum_{i=1}^{m}(f_i - M_f)(x_j^i - M_j)}{s_f s_j}.$$

Вернемся к n-мерным векторам данных и коэффициентов. Представим, что векторы сигналов проходят предобработку - центрирование и нормировку и далее мы имеем дело с векторами y^i:

$$y_j^i = \frac{x_j^i - M_j}{s_j}.$$

Это, в частности, означает, что все рассматриваемые координаты вектора x имеют ненулевую дисперсию, т.е. постоянные координаты исключаются из рассмотрения - они не несут полезной информации. Уравнения регрессии будем искать в форме: $\varphi(y)=(\beta,y)+\beta_0$. Получим:

$$\beta_0=M_f,\ \beta=s_f R^{-1} R_f, \tag{3}$$

где R_f - вектор коэффициентов корреляции f с j-ми координатами векторов x, имеющий координаты r_{fj}, R - матрица коэффициентов корреляции между координатами вектора данных:

$$r_{kj} = \frac{\frac{1}{m}\sum_{i=1}^{m}(x_k^i - M_k)(x_j^i - M_j)}{s_k s_j} = \frac{1}{m}\sum_{i=1}^{m} y_k^i y_j^i.$$

В задачах обработки данных почти всегда возникает вопрос о последовательном уточнении результатов по мере поступления новых данных (*обработка данных «на лету»*). Существует, как минимум, два подхода к ответу на этот вопрос для задачи линейной регрессии. Первый подход состоит в том, что изменения в коэффициентах регрессии при поступлении новых данных рассматриваются как малые и в их вычислении ограничиваются первыми порядками теории возмущений. При втором подходе для каждого нового вектора данных делается шаг изменений коэффициентов, уменьшающий ошибку регрессии Δ^2 на вновь поступившем векторе данных. При этом «предыдущий опыт» фиксируется только в текущих коэффициентах регрессии.

В рамках первого подхода рассмотрим, как будет изменяться α из формулы (2) при добавлении нового вектора данных. В первом порядке теории возмущений найдем изменение вектора коэффициента α при изменении вектора p и матрицы Q:

$$\alpha + \Delta\alpha = (Q + \Delta Q)^{-1}(p + \Delta p);$$
$$(Q + \Delta Q)^{-1} = (Q(1 + Q^{-1}\Delta Q))^{-1} = Q^{-1} - Q^{-1}\Delta Q Q^{-1} + o(\Delta Q);$$
$$\Delta\alpha \cong Q^{-1}(\Delta p - \Delta Q\alpha).$$

Пусть на выборке $\{x^i\}_{i=1}^m$ вычислены p, Q, Q^{-1}. При получении нового вектора данных x^{m+1} и соответствующего значения $F(x^{m+1})=f_{m+1}$ имеем[2]:

$$\begin{aligned}
&\Delta p = \tfrac{1}{m+1}(f_{m+1}x^{m+1} - p);\\
&\Delta q_{jk} = \tfrac{1}{m+1}(x_j^{m+1}x_k^{m+1} - q_{jk}) \text{ (т.е. } \Delta Q = \tfrac{1}{m+1}(x^{m+1} \otimes (x^{m+1})^{\mathrm{T}}));\\
&\Delta(Q^{-1}) = \tfrac{1}{m+1}(Q^{-1} - (Q^{-1}x^{m+1}) \otimes (Q^{-1}x^{m+1})^{\mathrm{T}});\\
&\Delta\alpha = \tfrac{1}{m+1}\Delta_{m+1}^0 Q^{-1}x^{m+1},
\end{aligned} \qquad (4)$$

где $\Delta_{m+1}^0 = f_{m+1} - (\alpha, x^{m+1})$ - ошибка на векторе данных x^{m+1} регрессионной зависимости, полученной на основании выборки $\{x^i\}_{i=1}^m$.

Пересчитывая по приведенным формулам p, Q, Q^{-1} и α после каждого получения данных, получаем процесс, в котором последовательно уточняются уравнения линейной регрессии. И требуемый объем памяти, и количество операций имеют порядок n^2 - из-за необходимости накапливать и модифицировать матрицу Q^{-1}. Конечно, это меньше, чем потребуется на обычное обращение матрицы Q на каждом шаге, однако следующий простой алгоритм еще экономнее. Он вовсе не обращается к матрицам Q, Q^{-1} и основан на уменьшении на каждом шаге величины $(\Delta_{m+1}^0)^2 = (f_{m+1} - (\alpha, x^{m+1}))^2$ - квадрата ошибки на векторе данных x^{m+1} регрессионной зависимости, полученной на основании выборки $\{x^i\}_{i=1}^m$.

Вновь обратимся к формуле (2) и будем рассматривать n+1-мерные векторы данных и коэффициентов. Обозначим $x = x^{m+1}$; $\Delta = \Delta_{m+1}^0 = f_{m+1} - (\alpha, x^{m+1})$. Тогда

$$grad_\alpha \Delta = -\Delta \times x\,. \qquad (5)$$

Последняя элементарная формула столь важна в теории адаптивных сумматоров, что носит «именное название» - формула Уидроу. «Обучение» адаптивного сумматора методом наискорейшего спуска состоит в изменении вектора коэффициентов α в направлении антиградиента Δ^2: на каждом шаге к α добавляется $h\times\Delta\times x$, где h - величина шага.

Если при каждом поступлении нового вектора данных x изменять α указанным образом, то получим последовательную процедуру построения линейной

[2] Напомним, что для векторов x, y матрица $x\otimes y^{\mathrm{T}}$ имеет своими элементами $x_i y_k$.

аппроксимации функции *F(x)*. Такой алгоритм обучения легко реализуется аппаратными средствами (изменение веса связи α_j есть произведение прошедшего по ней сигнала x_j на ошибку Δ и на величину шага). Возникает, однако, проблема сходимости: если *h* слишком мало, то сходимость будет медленной, если же слишком велико, то произойдет потеря устойчивости и сходимости не будет вовсе. Детальному изложению этого подхода и его приложений посвящен учебник [15].

Задача четкого разделения двух классов по обучающей выборке ставится так: имеется два набора векторов x^1 , ..., x^m и y^1,...,y^m . Заранее известно, что x^i относится к первому классу, а y^i - ко второму. Требуется построить решающее правило, то есть определить такую функцию $f(x)$, что при $f(x)>0$ вектор x относится к первому классу, а при $f(x)<0$ - ко второму.

Координаты классифицируемых векторов представляют собой значения некоторых признаков (свойств) исследуемых объектов.

Эта задача возникает во многих случаях: при диагностике болезней и определении неисправностей машин по косвенным признакам, при распознавании изображений и сигналов и т.п.

Строго говоря, классифицируются не векторы свойств, а объекты, которые обладают этими свойствами. Это замечание становится важным в тех случаях, когда возникают затруднения с построением решающего правила - например тогда, когда встречаются принадлежащие к разным классам объекты, имеющие одинаковые признаки. В этих случаях возможно несколько решений:

1) искать дополнительные признаки, позволяющие разделить классы;

2) примириться с неизбежностью ошибок, назначить за каждый тип ошибок свой штраф (c_{12} - штраф за то, что объект первого класса отнесен ко второму, c_{21} - за то, что объект второго класса отнесен к первому) и строить разделяющее правило так, чтобы минимизировать математическое ожидание штрафа;

3) перейти к нечеткому разделению классов - строить так называемые "функции принадлежности" $f_1(x)$ и $f_2(x)$ - $f_i(x)$ оценивает степень уверенности при отнесении объекта к i-му классу (i=1,2), для одного и того же x может быть так, что и $f_1(x)>0$, и $f_2(x)>0$.

Линейное разделение классов состоит в построении линейного решающего правила - то есть такого вектора α и числа α_0 (называемого порогом), что при $(x,\alpha)>\alpha_0$ x относится к первому классу, а при $(x, \alpha)<\alpha_0$ - ко второму.

Поиск такого решающего правила можно рассматривать как разделение классов в проекции на прямую. Вектор α задает прямую, на которую ортогонально проектируются все точки, а число α_0 - точку на этой прямой, отделяющую первый класс от второго.

Простейший и подчас очень удобный выбор состоит в проектировании на прямую, соединяющую центры масс выборок. Центр масс вычисляется в предположении, что массы всех точек одинаковы и равны 1. Это соответствует заданию α в виде

$$\alpha= (y^1 + y^2 +... + y^m)/m - (x^1 + x^2 +... + x^n)/n. \qquad (6)$$

Во многих случаях удобнее иметь дело с векторами единичной длины. Нормируя α, получаем:

$$\alpha= ((y^1 + y^2 +... + y^m)/m - (x^1 + x^2 +... + x^n)/n)/\|(y^1 + y^2 +... + y^m)/m - (x^1 + x^2 +... + x^n)/n\|.$$

Выбор α_0 может производиться из различных соображений. Простейший вариант - посередине между центрами масс выборок:

$$\alpha_0=(((y^1 + y^2 +... + y^m)/m,\alpha)+((x^1 + x^2 +... + x^n)/n,\alpha))/2.$$

Более тонкие способы построения границы раздела классов α_0 учитывают различные вероятности появления объектов разных классов, и оценки плотности распределения точек классов на прямой. Чем меньше вероятность появления данного класса, тем более граница раздела приближается к центру тяжести соответствующей выборки.

Можно для каждого класса построить приближенную плотность вероятностей распределения проекций его точек на прямую (это намного проще, чем для многомерного распределения) и выбирать α_0, минимизируя вероятность ошибки. Пусть решающее правило имеет вид: при $(x, \alpha)>\alpha_0$ x относится к первому классу, а при $(x, \alpha)<\alpha_0$ - ко второму. В таком случае вероятность ошибки будет равна

$$P = p_1 \int_{-\infty}^{\alpha_0} \rho_1(\chi) d\chi + p_2 \int_{\alpha_0}^{\infty} \rho_2(\chi) d\chi$$

где p_1, p_2 - априорные вероятности принадлежности объекта соответствующему классу,

$\rho_1(\chi)$, $\rho_2(\chi)$ - плотности вероятности для распределения проекций χ точек x в каждом классе.

Приравняв нулю производную вероятности ошибки по α_0, получим: число α_0, доставляющее минимум вероятности ошибки, является корнем уравнения:

$$p_1\rho_1(\chi)=p_2\rho_2(\chi), \qquad (7)$$

либо (если у этого уравнения нет решений) оптимальным является правило, относящее все объекты к одному из классов.

Если принять гипотезу о нормальности распределений:

$$\rho(y)=\frac{1}{\sqrt{2\pi}\sigma}e^{-(y-a)^2/2\sigma^2},$$

то для определения α_0 получим:

$$\frac{p_1}{\sqrt{2\pi}\sigma_1}e^{-(y-a_1)^2/2\sigma_1^2}=\frac{p_2}{\sqrt{2\pi}\sigma_2}e^{-(y-a_2)^2/2\sigma_2^2},$$

$$\ln(p_1/p_2)-\ln(\sigma_1/\sigma_2)-(y-a_1)^2/2\sigma_1^2+(y-a_2)^2/2\sigma_2^2=0$$

Если это уравнение имеет два корня $y=\alpha_1$, α_2, ($\alpha_1<\alpha_2$) то наилучшим решающим правилом будет: при $\alpha_1<(x, \alpha)<\alpha_2$ объект принадлежит одному классу, а при $\alpha_1>(x, \alpha)$ или $(x, \alpha)>\alpha_2$ - другому (какому именно, определяется тем, которое из произведений $p_i\rho_i(\chi)$ больше). Если корней нет, то оптимальным является отнесение к одному из классов. Случай единственного корня представляет интерес только тогда, когда $\sigma_1=\sigma_2$. При этом уравнение превращается в линейное и мы приходим к исходному варианту - единственной разделяющей точке α_0.

Таким образом, разделяющее правило с единственной разделяющей точкой α_0 не является наилучшим для нормальных распределений и надо искать две разделяющие точки.

Если сразу ставить задачу об оптимальном разделении многомерных нормальных распределений, то получим, что наилучшей разделяющей поверхностью является квадрика (на прямой типичная «квадрика» - две точки). Предполагая, что ковариационные матрицы классов совпадают (в одномерном случае это предположение о том, что $\sigma_1=\sigma_2$), получаем линейную разделяющую поверхность. Она ортогональна прямой, соединяющей центры выборок не в обычном скалярном произведении, а в

специальном: $\langle x, y\rangle = \left(x, \Sigma^{-1} y\right)$, где Σ - общая ковариационная матрица классов. За деталями отсылаем к прекрасно написанной книге [17], см. также [11,12,16].

Важная возможность усовершенствовать разделяющее правило состоит с использовании оценки не просто вероятности ошибки, а среднего риска: каждой ошибке приписывается «цена» c_i и минимизируется сумма $c_1 p_1 \rho_1(\chi)+c_2 p_2 \rho_2(\chi)$. Ответ получается практически тем же (всюду p_i заменяются на $c_i p_i$), но такая добавка важна для многих приложений.

Требование безошибочности разделяющего правила на обучающей выборке принципиально отличается от обсуждавшихся критериев оптимальности. На основе этого требования строится **персептрон Розенблатта** - «дедушка» современных нейронных сетей [1].

Возьмем за основу при построении гиперплоскости, разделяющей классы, отсутствие ошибок на обучающей выборке. Чтобы удовлетворить этому условию, придется решать систему линейных неравенств:

$$(x^i, \alpha)>\alpha_0 \ (i=1,...,n)$$

$$(y^j, \alpha)<\alpha_0 \ (j=1,...,m).$$

Здесь x^i (i=1,..,n) - векторы из обучающей выборки, относящиеся к первому классу, а y^j (j=1,..,n) - ко второму.

Удобно переформулировать задачу. Увеличим размерности всех векторов на единицу, добавив еще одну координату $-\alpha_0$ к α, x_0 =1 - ко всем x и y_0 =1 - ко всем y . Сохраним для новых векторов прежние обозначения - это не приведет к путанице.

Наконец, положим $z^i = x^i$ (i=1,...,n), $z^j = -y^j$ (j=1,...,m).

Тогда получим систему $n+m$ неравенств

$$(z^i,\alpha)>0 \ (i=1,...,n+m),$$

которую будем решать относительно α. Если множество решений непусто, то любой его элемент α порождает решающее правило, безошибочное на обучающей выборке.

Итерационный алгоритм решения этой системы чрезвычайно прост. Он основан на том, что для любого вектора x его скалярный квадрат (x,x) больше нуля. Пусть α - некоторый вектор, претендующий на роль решения неравенств $(z^i,\alpha)>0$ (i=1,...,$n+m$), однако часть из них не выполняется. Прибавим те z^i, для которых неравенства имеют

неверный знак, к вектору α и вновь проверим все неравенства $(z^i,\alpha)>0$ и т.д. Если они совместны, то процесс сходится за конечное число шагов. Более того, добавление z^i к α можно производить сразу после того, как ошибка ($(z^i,\alpha)<0$) обнаружена, не дожидаясь проверки всех неравенств - и этот вариант алгоритма тоже сходится [2].

2. Сети Хопфилда

Перейдем от одноэлементных систем к нейронным сетям. Пусть α_{ij} - вес связи, ведущей от *j*-го нейрона к *i*-му (полезно обратить внимание на порядок индексов). Для полносвязных сетей определены значения α_{ij} при всех *i*,*j*, для других архитектур связи, ведущие от *j*-го нейрона к *i*-му для некоторых *i*,*j* не определены. В этом случае положим $\alpha_{ij}=0$.

В данном разделе речь пойдет в основном о полносвязных сетях. Пусть на выходах всех нейронов получены сигналы x_j (*j*-номер нейрона). Обозначим *x* вектор этих выходных сигналов. Прохождение вектора сигналов *x* через сеть связей сводится к умножению матрицы (α_{ij}) на вектор сигналов *x*. В результате получаем вектор входных сигналов нелинейных элементов нейронов: $y_i = \sum_j \alpha_{ij} x_j$.

Это соответствие «*прохождение сети ≡ умножение матрицы связей на вектор сигналов*» является основой для перевода обычных численных методов на нейросетевой язык и обратно. Практически всюду, где основной операцией является умножение матрицы на вектор, применимы нейронные сети. С другой стороны, любое устройство, позволяющее быстро осуществлять такое умножение, может использоваться для реализации нейронных сетей.

В частности, *вычисление градиента квадратичной формы* $H = \frac{1}{2}(x, Qx)$ может осуществляться полносвязной сетью с *симметричной* матрицей связей: $\mathrm{grad}H = Qx$ ($\alpha_{ij} = q_{ij} = q_{ji}$). Именно это наблюдение лежит в основе данного раздела.

Что можно сделать, если мы умеем вычислять градиент квадратичной формы?

В первую очередь, можно методом наискорейшего спуска **искать точку минимума многочлена второго порядка**. Пусть задан такой многочлен: $P(x) = \frac{1}{2}(x, Qx) + (b, x)$. Его градиент равен $\mathrm{grad}P = Qx + b$. Этот вектор может быть

получен при прохождении вектора x через сеть с весами связей $\alpha_{ij} = q_{ij} = q_{ji}$ при условии, что на входной сумматор каждого нейрона по дополнительной связи веса b подается стандартный единичный сигнал.

Зададим теперь функционирование сети формулой

$$x' = x - h\mathrm{grad}P = x - h(Qx + b) \qquad (8)$$

Нелинейных элементов вовсе не нужно! Каждый (j-й) нейрон имеет входные веса $\alpha_{ij} = -hq_{ij}$ для связей с другими нейронами ($i \neq j$), вес $-b_j$ для постоянного единичного входного сигнала и вес $\alpha_{jj} = 1 - hq_{jj}$ для связи нейрона с самим собой (передачи на него его же сигнала с предыдущего шага). Выбор шага h>0 может вызвать затруднение (он зависит от коэффициентов минимизируемого многочлена). Есть, однако, простое решение: в каждый момент дискретного времени T выбирается свое значение h_T. Достаточно, чтобы шаг стремился со временем к нулю, а сумма шагов - к бесконечности (например, $h_T = \frac{1}{T}$, или $h_T = \frac{1}{\sqrt{T}}$).

Итак, простая симметричная полносвязная сеть без нелинейных элементов может методом наискорейшего спуска искать точку минимума квадратичного многочлена.

Решение системы линейных уравнений $Ax = b$ сводится к минимизации многочлена

$$P = \frac{1}{2}((Ax - b),(Ax - b)) = \frac{1}{2}(x, A^{\mathrm{T}}Ax) - (A^{\mathrm{T}}b, x) - \frac{1}{2}(b, b).$$

Поэтому решение системы может производиться нейронной сетью. Простейшая сеть, вычисляющая градиент этого многочлена, не полносвязна, а состоит из двух слоев: первый с матрицей связей A, второй - с транспонированной матрицей A^{T}. Постоянный единичный сигнал подается на связи с весами b_j на первом слое. Минимизация этого многочлена, а значит и решение системы линейных уравнений, может проводиться так же, как и в общем случае, в соответствии с формулой $x' = x - h\mathrm{grad}P$. Усовершенствованные варианты алгоритма решения можно найти в работах Сударикова [8].

Небольшая модификация позволяет вместо безусловного минимума многочлена второго порядка P искать точку условного минимума с условиями $x_i = c_i$ для $i = i_1, \ldots, i_k$, то есть точку минимума P в ограничении на аффинное

многообразие, параллельное некоторым координатным плоскостям. Для этого вместо формулы

$$x' = x - h\mathrm{grad}P = x - h(Qx + b)$$

следует использовать:

$$x'_i = c_i \text{ при } i = i_1, \ldots, i_k;$$

$$x'_i = x_i - h\frac{\partial P}{\partial x_i} = x_i - h\left(\sum_j q_{ij}x_j + b_i\right) \text{ при } i \neq i_1, \ldots, i_k. \qquad (9)$$

Устройство, способное *находить точку условного минимума многочлена второго порядка* при условиях вида $x_i = c_i$ для $i = i_1, \ldots, i_k$ позволяет решать важную задачу - **заполнять пробелы в данных** (и, в частности, строить линейную регрессию).

Предположим, что получаемые в ходе испытаний векторы данных подчиняются многомерному нормальному распределению:

$$\rho(x) = Ce^{-\frac{1}{2}((x-\mathbf{M}x),Q(x-\mathbf{M}x))},$$

где $\mathbf{M}x$ - вектор математических ожиданий координат, $Q = \Sigma^{-1}$, Σ - ковариационная матрица, n - размерность пространства данных,

$$C = \frac{1}{(2\pi)^{n/2}\sqrt{\det\Sigma}}.$$

Напомним определение матрицы Σ:

$$(\Sigma)_{ij} = \mathbf{M}((x_i - \mathbf{M}x_i)(x_j - \mathbf{M}x_j)),$$

где $\mathbf{M}$ - символ математического ожидания, нижний индекс соответствует номеру координаты.

В частности, простейшая оценка ковариационной матрицы по выборке дает:

$$S = \frac{1}{m}\sum_j (x^j - \mathbf{M}x) \otimes (\mathbf{M}x^j - \mathbf{M}x)^{\mathrm{T}},$$

где m - число элементов в выборке, верхний индекс j - номер вектора данных в выборке, верхний индекс Т означает транспонирование, а $\otimes$ - произведение вектора-столбца на вектор-строку (тензорное произведение).

Пусть у вектора данных x известно несколько координат: $x_i = c_i$ для $i = i_1, \ldots, i_k$. Наиболее вероятные значения неизвестных координат должны доставлять условный максимум показателю нормального распределения - многочлену второго порядка

$((x-\mathbf{M}x),Q(x-\mathbf{M}x))$ (при условии $x_i = c_i$ для $i = i_1,\ldots,i_k$). Эти же значения будут условными математическими ожиданиями неизвестных координат при заданных условиях.

Таким образом, чтобы построить сеть, заполняющую пробелы в данных, достаточно сконструировать сеть для поиска точек условного минимума многочлена $((x-\mathbf{M}x),Q(x-\mathbf{M}x))$ при условиях следующего вида: $x_i = c_i$ для $i = i_1,\ldots,i_k$. Матрица связей Q выбирается из условия $Q = \Sigma^{-1}$, где Σ - ковариационная матрица (ее оценка по выборке).

На первый взгляд, пошаговое накопление $Q = \Sigma^{-1}$ по мере поступления данных требует слишком много операций - получив новый вектор данных требуется пересчитать оценку Σ, а потом вычислить $Q = \Sigma^{-1}$. Можно поступать и по-другому, воспользовавшись формулой приближенного обращения матриц первого порядка точности:

$$(\Sigma + \varepsilon\Delta)^{-1} = \Sigma^{-1} - \varepsilon\Sigma^{-1}\Delta\Sigma^{-1} + o(\varepsilon).$$

Если же добавка Δ имеет вид $\Delta = y \otimes y^{\mathrm{T}}$, то

$$(\Sigma + \varepsilon\Delta)^{-1} = \Sigma^{-1} - \varepsilon(\Sigma^{-1}y) \otimes (\Sigma^{-1}y)^{\mathrm{T}} + o(\varepsilon). \qquad (10)$$

Заметим, что решение задачи (точка условного минимума многочлена) не меняется при умножении Q на число. Поэтому полагаем:

$$Q_0 = 1,\ Q_{k+1} = Q_k + \varepsilon(Q_k(x^{k+1} - (\mathbf{M}x)^{k+1})) \otimes (Q_k(x^{k+1} - (\mathbf{M}x)^{k+1}))^{\mathrm{T}},$$

где 1 - единичная матрица, ε>0 - достаточно малое число, x^{k+1} - k+1-й вектор данных, $(\mathbf{M}x)^{k+1}$ - среднее значение вектора данных, уточненное с учетом x^{k+1}:

$$(\mathbf{M}x)^{k+1} = \frac{1}{k+1}(k(\mathbf{M}x)^k + x^{k+1})$$

В формуле для пошагового накопления матрицы Q ее изменение ΔQ при появлении новых данных получается с помощью вектора $y = x^{k+1} - (\mathbf{M}x)^{k+1}$, пропущенного через сеть: $(\Delta Q)_{ij} = \varepsilon z_i z_j$, где $z = Qy$. Параметр ε выбирается достаточно малым для того, чтобы обеспечить положительную определенность получаемых матриц (и, по возможности, их близость к истинным значениям Q).

Описанный процесс формирования сети можно назвать обучением. Вообще говоря, можно проводить формальное различение между формированием сети по *явным*

формулам и по алгоритмам, не использующим явных формул для весов связей (*неявным*). Тогда термин «обучение» предполагает неявные алгоритмы, а для явных остается название «формирование». Здесь мы такого различия проводить не будем.

Если при обучении сети поступают *некомплектные данные* x^{k+1} с отсутствием значений некоторых координат, то сначала эти значения восстанавливаются с помощью имеющейся сети, а потом используются в ее дальнейшем обучении.

Во всех задачах оптимизации существенную роль играет вопрос о *правилах остановки:* когда следует прекратить циклическое функционирование сети, остановиться и считать полученный результат ответом? Простейший выбор - остановка по малости изменений: если изменения сигналов сети за цикл меньше некоторого фиксированного малого δ (при использовании переменного шага δ может быть его функцией), то оптимизация заканчивается.

До сих пор речь шла о минимизации положительно определенных квадратичных форм и многочленов второго порядка. Однако самое знаменитое приложение полносвязных сетей связано с увеличением значений положительно определенных квадратичных форм. Речь идет о системах **ассоциативной памяти** [4-7,9,10,12].

Предположим, что задано несколько эталонных векторов данных $x^1,\ldots,x^m$ и при обработке поступившего на вход системы вектора x требуется получить на выходе ближайший к нему эталонный вектор. Мерой сходства в простейшем случае будем считать косинус угла между векторами - для векторов фиксированной длины это просто скалярное произведение. Можно ожидать, что изменение вектора x по закону

$$x' = x + h\sum_k x^k (x^k, x), \qquad (11)$$

где h - малый шаг, приведет к увеличению проекции x на те эталоны, скалярное произведение на которые (x^k, x) больше.

Ограничимся рассмотрением эталонов, и ожидаемых результатов обработки с координатами ± 1. Развивая изложенную идею, приходим к дифференциальному уравнению

$$\frac{dx}{dt} = -\text{grad}H,$$

$$H = H_0 + \theta H_1,\ H_0(x) = -\frac{1}{2}\sum_k (x^k, x)^2,\ H_1(x) = \frac{1}{2}\sum_i (x_i^2 - 1)^2, \quad (12)$$

$$\text{grad}H_0 = -\sum_k x^k (x^k, x),\ (\text{grad}H_1)_j = (x_j^2 - 1)x_j,$$

где верхними индексами обозначаются номера векторов-эталонов, нижними - координаты векторов.

Функция H называется «*энергией*» сети, она минимизируется в ходе функционирования. Слагаемое H_0 вводится для того, чтобы со временем возрастала проекция вектора x на те эталоны, которые к нему ближе, слагаемое H_1 обеспечивает стремление координат вектора x к ± 1. Параметр θ определяет соотношение между интенсивностями этих двух процессов. Целесообразно постепенно менять θ со временем, начиная с малых θ<1, и приходя в конце концов к θ>1.

Подробнее системы ассоциативной памяти рассмотрены в отдельной главе. Здесь же мы ограничимся обсуждением получающихся весов связей. Матрица связей построенной сети определяется функцией H_0, так как $(\text{grad}H_1)_j = (x_j^2 - 1)x_j$ вычисляется непосредственно при j-м нейроне без участия сети. Вес связи между i-м и j-м нейронами не зависит от направления связи и равен

$$\alpha_{ij} = \sum_k x_i^k x_j^k\ . \quad (13)$$

Эта простая формула имеет чрезвычайно важное значение для развития теории нейронных сетей. Вклад k-го эталона в связь между i-м и j-м нейронами ($x_i^k x_j^k$) равен +1, если i-я и j-я координаты этого эталона имеют одинаковый знак, и равен -1, если они имеют разный знак.

В результате возбуждение i-го нейрона передается j-му (и симметрично, от j-го к i-му), если у большинства эталонов знак i-й и j-й координат совпадают. В противном случае эти нейроны тормозят друг друга: возбуждение i-го ведет к торможению j-го, торможение i-го - к возбуждению j-го (воздействие j-го на i-й симметрично). Это правило образования ассоциативных связей (правило Хебба) сыграло огромную роль в теории нейронных сетей.

3. Сети Кохонена для кластер-анализа и классификации без учителя

Построение отношений на множестве объектов - одна из самых загадочных и открытых для творчества областей применения искусственного интеллекта. Первым и наиболее распространенным примером этой задачи является классификация без учителя. Задан набор объектов, каждому объекту сопоставлен вектор значений признаков (строка таблицы). Требуется разбить эти объекты на классы эквивалентности.

Естественно, прежде, чем приступать к решению этой задачи, нужно ответить на один вопрос: зачем производится это разбиение и что мы будем делать с его результатом? Ответ на него позволит приступить к формальной постановке задачи, которая всегда требует компромисса между сложностью решения и точностью формализации: буквальное следование содержательному смыслу задачи нередко порождает сложную вычислительную проблему, а следование за простыми и элегантными алгоритмами может привести к противоречию со здравым смыслом.

Итак, зачем нужно строить отношения эквивалентности между объектами? В первую очередь - для фиксации знаний. Люди накапливают знания о классах объектов - это практика многих тысячелетий, зафиксированная в языке: знание относится к имени класса (пример стандартной древней формы: "люди смертны", "люди" - имя класса). В результате классификации как бы появляются новые имена и правила их присвоения.

Для каждого нового объекта мы должны сделать два дела:

1) найти класс, к которому он принадлежит;

2) использовать новую информацию, полученную об этом объекте, для исправления (коррекции) правил классификации.

Какую форму могут иметь правила отнесения к классу? Веками освящена традиция представлять класс его "типичным", "средним", "идеальным" и т.п. элементом. Этот типичный объект является идеальной конструкцией, олицетворяющей класс.

Отнесение объекта к классу проводится путем его сравнения с типичными элементами разных классов и выбора ближайшего. Правила, использующие типичные объекты и меры близости для их сравнения с другими, очень популярны и сейчас.

Простейшая мера близости объектов - квадрат евклидового расстояния между векторами значений их признаков (чем меньше расстояние - расстояние - тем ближе объекты). Соответствующее определение признаков типичного объекта - среднее арифметическое значение признаков по выборке, представляющей класс.

Мы не оговариваем специально существование априорных ограничений, налагаемых на новые объекты - естественно, что "вселенная" задачи много 'уже и гораздо определеннее Вселенной.

Другая мера близости, естественно возникающая при обработке сигналов, изображений и т.п. - квадрат коэффициента корреляции (чем он больше, тем ближе объекты). Возможны и иные варианты - все зависит от задачи.

Если число классов m заранее определено, то задачу классификации без учителя можно поставить следующим образом.

Пусть $\{x^p\}$ - векторы значений признаков для рассматриваемых объектов и в пространстве таких векторов определена мера их близости $\rho\{x,y\}$. Для определенности примем, что чем ближе объекты, тем меньше ρ. С каждым классом будем связывать его типичный объект. Далее называем его ядром класса. Требуется определить набор из m ядер $y^1, y^2, \ldots y^m$ и разбиение $\{x^p\}$ на классы:

$$\{x^p\} = Y_1 \cup Y_2 \cup \ldots \cup Y_m$$

минимизирующее следующий критерий

$$Q = \sum_{i=1}^{m} D_i \to \min, \tag{14}$$

где для каждого (i-го) класса D_i - сумма расстояний от принадлежащих ему точек выборки до ядра класса:

$$D_i = \sum_{x^p \in Y_i} \rho(x^p, y^i). \tag{15}$$

Минимум Q берется по всем возможным положениям ядер y^i и всем разбиениям $\{x^p\}$ на m классов Y_i.

Если число классов заранее не определено, то полезен критерий слияния классов: классы Y_i и Y_j сливаются, если их ядра ближе, чем среднее расстояние от элемента класса до ядра в одном из них. (Возможны варианты: использование среднего расстояния по обоим классам, использование порогового коэффициента, показывающего, во сколько раз должно расстояние между ядрами превосходить среднее расстояние от элемента до ядра и др.)

Использовать критерий слияния классов можно так: сначала принимаем гипотезу о достаточном числе классов, строим их, минимизируя Q, затем некоторые Y_i объединяем, повторяем минимизацию Q с новым числом классов и т.д.

Существует много эвристических алгоритмов классификации без учителя, основанных на использовании мер близости между объектами. Каждый из них имеет свою область применения, а наиболее распространенным недостатком является отсутствие четкой формализации задачи: совершается переход от идеи кластеризации прямо к алгоритму, в результате неизвестно, что ищется (но что-то в любом случае находится, иногда - неплохо).

Сетевые алгоритмы классификации без учителя строятся на основе итерационного метода динамических ядер. Опишем его сначала в наиболее общей абстрактной форме. Пусть задана выборка предобработанных векторов данных $\{x^p\}$. Пространство векторов данных обозначим E. Каждому классу будет соответствовать некоторое ядро a. Пространство ядер будем обозначать A. Для каждых $x \in E$ и $a \in A$ определяется мера близости $d(x,a)$. Для каждого набора из k ядер $a_1,...,a_k$ и любого разбиения $\{x^p\}$ на k классов $\{x^p\}=P_1 \cup P_2 \cup ... \cup P_k$ определим критерий качества:

$$D = D(a_1, a_2, \ldots, a_k, P_1, P_2, \ldots P_k) = \sum_{i=1}^{k} \sum_{x \in P_i} d(x, a_i) \qquad (16)$$

Требуется найти набор $a_1,...,a_k$ и разбиение $\{x^p\}=P_1 \cup P_2 \cup ... \cup P_k$, минимизирующие D.

Шаг алгоритма разбивается на два этапа:

1-й этап - для фиксированного набора ядер $a_1,...,a_k$ ищем минимизирующее критерий качества D разбиение $\{x^p\}=P_1 \cup P_2 \cup ... \cup P_k$; оно дается решающим правилом: $x \in P_i$, если $d(x, a_i) < d(x, a_j)$ при $i \neq j$, в том случае, когда для x минимум $d(x,a)$ достигается при нескольких значениях i, выбор между ними может быть сделан произвольно;

2-й этап - для каждого P_i (i=1,...,k), полученного на первом этапе, ищется $a_i \in A$, минимизирующее критерий качества (т.е. слагаемое в D для данного i - $D_i = \sum_{x \in P_i} d(x, a_i)$

Начальные значения $a_1,...,a_k$, $\{x^p\}=P_1 \cup P_2 \cup ... \cup P_k$ выбираются произвольно, либо по какому-нибудь эвристическому правилу.

На каждом шаге и этапе алгоритма уменьшается критерий качества D, отсюда следует сходимость алгоритма - после конечного числа шагов разбиение $\{x^p\}=P_1\cup P_2\cup...\cup P_k$ уже не меняется.

Если ядру a_i сопоставляется элемент сети, вычисляющий по входному сигналу x функцию $d(x,a_i)$, то решающее правило для классификации дается интерпретатором "победитель забирает все": элемент x принадлежит классу P_i, если выходной сигнал i-го элемента $d(x,a_i)$ меньше всех остальных

Единственная вычислительная сложность в алгоритме может состоять в поиске ядра по классу на втором этапе алгоритма, т.е. в поиске $a\in A$, минимизирующего $D_i = \sum_{x\in P_i} d(x,a)$

В связи с этим, в большинстве конкретных реализаций метода мера близости d выбирается такой, чтобы легко можно было найти a, минимизирующее D для данного P .

В простейшем случае пространство ядер A совпадает с пространством векторов x, а мера близости $d(x,a)$ - положительно определенная квадратичная форма от x-a, например, квадрат евклидового расстояния или другая положительно определенная квадратичная форма. Тогда ядро a_i, минимизирующее D_i, есть центр тяжести класса P_i :

$$a_i = \frac{1}{|P_i|}\sum_{x\in P_i} x, \qquad (17)$$

где $|P_i|$ - число элементов в P_i.

В этом случае также упрощается и решающее правило, разделяющее классы. Обозначим $d(x,a)=(x-a,x-a)$, где (.,.) - билинейная форма (если d - квадрат евклидового расстояния между x и a, то (.,.) - обычное скалярное произведение). В силу билинейности

$d(x,a)=(x-a,x-a)=(x,x)-2(x,a)+(a,a)$.

Чтобы сравнить $d(x,a_i)$ для разных i и найти среди них минимальное, достаточно вычислить линейную неоднородную функцию от x:

$d_1(x,a_i) = (a_i,a_i)-2(x,a_i)$.

Минимальное значение $d(x,a_i)$ достигается при том же i, что и минимум $d_1(x,a_i)$, поэтому решающее правило реализуется с помощью k сумматоров, вычисляющих $d(x,a)$

и интерпретатора, выбирающего сумматор с минимальным выходным сигналом. Номер этого сумматора и есть номер класса, к которому относится *x*.

Пусть теперь мера близости - коэффициент корреляции между вектором данных и ядром класса:

$$d(x,a) = r(x,a) = \sum_j \frac{(x_j - M_x)(a_j - M_a)}{\sigma_x \sigma_a}$$

где x_j, a_j - координаты векторов, $M_x = \frac{1}{n}\sum_j x_j$ (и аналогично M_a), n - размерность пространства данных, $\sigma_x = \sqrt{\frac{1}{n}\sum_j (x_j - M_x)^2}$ (и аналогично σ_a).

Предполагается, что данные предварительно обрабатываются (нормируются и центрируются) по правилу:

$$x \to \frac{x_j - M_x}{\sigma_x}.$$

Точно также нормированы и центрированы векторы ядер *a*. Поэтому все обрабатываемые векторы и ядра принадлежат сечению единичной евклидовой сферы ($\|x\|=1$) гиперплоскостью ($\sum_i x_i = 0$). В таком случае $d(x,a) = (x,a)$.

Задача поиска ядра для данного класса *P* имеет своим решением

$$a_P = \sum_{x \in P} x \Big/ \left\| \sum_{x \in P} x \right\|. \qquad (18)$$

В описанных простейших случаях, когда ядро класса точно определяется как среднее арифметическое (или нормированное среднее арифметическое) элементов класса, а решающее правило основано на сравнении выходных сигналов линейных адаптивных сумматоров, нейронную сеть, реализующую метод динамических ядер, называют сетью Кохонена. В определении ядер *a* для сетей Кохонена входят суммы $\sum_{x \in P} x$. Это позволяет накапливать новые динамические ядра, обрабатывая по одному примеру и пересчитывая a_i после появления в P_i нового примера. Сходимость при такой модификации, однако, ухудшается.

Закончим раздел рассмотрением различных способов использования полученных классификаторов.

1. *Базовый способ*: для вектора данных x^i и каждого ядра a_i вычисляется $y_i=d(x,a_i)$ (условимся считать, что правильному ядру отвечает максимум d, изменяя, если надо, знак d); по правилу «победитель забирает все» строка ответов y_i преобразуется в строку, где только один элемент, соответствующий максимальному y_i, равен 1, остальные - нули. Эта строка и является результатом функционирования сети. По ней может быть определен номер класса (номер места, на котором стоит 1) и другие показатели.

2. *Метод аккредитации*: за слоем элементов базового метода, выдающих сигналы 0 или 1 по правилу "победитель забирает все" (далее называем его слоем базового интерпретатора), надстраивается еще один слой выходных сумматоров. С каждым (i-м) классом ассоциируется q-мерный выходной вектор z^i с координатами z^i_j . Он может формироваться по-разному: от двоичного представления номера класса до вектора ядра класса. Вес связи, ведущей от i-го элемента слоя базового интерпретатора к j-му выходному сумматору определяется в точности как z^i_j . Если на этом i-м элементе базового интерпретатора получен сигнал 1, а на остальных - 0, то на выходных сумматорах будут получены числа z^i_j.

3. *Нечеткая классификация*. Пусть для вектор данных x обработан слоем элементов, вычисляющих $y_i=d(x,a_i)$. Идея дальнейшей обработки состоит в том, чтобы выбрать из этого набора $\{y_i\}$ несколько самых больших чисел и после нормировки объявить их значениями функций принадлежности к соответствующим классам. Предполагается, что к остальным классам объект наверняка не принадлежит. Для выбора семейства G наибольших y_i определим следующие числа:

$$y_{\max} = \max\{y_i\},\ M_y = \frac{1}{k}\sum_i y_i,\ s = (1-\alpha)M_y + \alpha y_{\max}$$

где число α характеризует отклонение "уровня среза" s от среднего значения M_y $\alpha\in$[-1,1], по умолчанию обычно принимается α=0.

Множество $J=\{i|y_i\in G\}$ трактуется как совокупность номеров тех классов, к которым может принадлежать объект, а нормированные на единичную сумму неотрицательные величины

$$f_i = \frac{y_i - s}{\sum_{j\in J}(y_i - s)}$$ (при $i\in J$ и $f = 0$ в противном случае)

интерпретируются как значения функций принадлежности этим классам.

4. *Метод интерполяции* надстраивается над нечеткой классификацией аналогично тому, как метод аккредитации связан с базовым способом. С каждым классом связывается q-мерный выходной вектор z^i. Строится слой из q выходных сумматоров, каждый из которых должен выдавать свою компоненту выходного вектора. Весовые коэффициенты связей, ведущих от того элемента нечеткого классификатора, который вычисляет f_i, к j-му выходному сумматору определяются как z^i_j. В итоге вектор выходных сигналов сети есть

$$z = \sum_i f_i z^i$$

В отдельных случаях по смыслу задачи требуется нормировка f_i на единичную сумму квадратов или модулей.

Выбор одного из описанных четырех вариантов использования сети (или какого-нибудь другого) определяется нуждами пользователя. Предлагаемые четыре способа покрывают большую часть потребностей.

За пределами этой главы остался наиболее универсальный способ обучения нейронных сетей методами гладкой оптимизации - минимизации функции оценки. Ему посвящена следующая глава.

Работа над главой была поддержана Красноярским краевым фондом науки, грант 6*F*0124.

Литература

1. Розенблатт Ф. Принципы нейродинамики. Перцептрон и теория механизмов мозга. М.: Мир, 1965. 480 с.

2. Минский М., Пайперт С. Персептроны. - М.: Мир, 1971.

3. Ивахненко А.Г. Персептроны. - Киев: Наукова думка, 1974.

4. Hopfield J.J. Neural Networks and Physical systems with emergent collective computational abilities//Proc. Nat. Sci. USA. 1982. V.79. P. 2554-2558.

5. Уоссермен Ф. Нейрокомпьютерная техника.- М.: Мир, 1992.

6. Итоги науки и техники. Сер. "Физ. и Матем. модели нейронных сетей" / Под ред. А.А.Веденова. - М.: Изд-во ВИНИТИ, 1990-92 - Т. 1-5.

7. Фролов А.А., Муравьев И.П. Нейронные модели ассоциативной памяти.- М.: Наука, 1987.- 160 с.

8. Судариков В.А. Исследование адаптивных нейросетевых алгоритмов решения задач линейной алгебры // Нейрокомпьютер, 1992. № 3,4. С. 13-20.

9. Кохонен Т. Ассоциативная память. - М.: Мир, 1980.

10. Кохонен Т. Ассоциативные запоминающие устройства. - М.: Мир, 1982.

11. Фор А. Восприятие и распознавание образов.- М.: Машиностроение, 1989.- 272 с.

12. Горбань А.Н., Россиев Д.А. Нейронные сети на персональном компьютере. Новосибирск: Наука (Сиб. отделение), 1996. 276 с.

13. Кендалл М., Стьюарт А. Статистические выводы и связи.- М.: Наука, 1973.- 900 с.

14. Мостеллер Ф., Тьюки Дж. Анализ данных и регрессия.- М.: Финансы и статистика, 1982.- 239 с.

15. Уидроу Б., Стирнз С. Адаптивная обработка сигналов. М.: Мир, 1989. 440 с.

16. Айвазян С.А., Бежаева З.И., Староверов О.В. Классификация многомерных наблюдений.- М.: Статистика, 1974.- 240 с.

17. Дуда Р., Харт П. Распознавание образов и анализ сцен.- М.: Мир, 1976.- 512 с.

18. Ивахненко А.Г. Самообучающиеся системы распознавания и автоматического регулирования.- Киев: Техника, 1969.- 392 с.

19. Искусственный интеллект: В 3-х кн. Кн. 2. Модели и методы: Справочник / под ред. Д.А. Поспелова.- М.: Радио и связь, 1990.- 304 с.

20. Zurada J. M. Introduction to artificial neural systems. PWS Publishing Company, 1992. 785 P.

21. Горбань А.Н. Обучение нейронных сетей. М.: изд. СССР-США СП "ПараГраф", 1990. 160 с.