МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Северо-Кавказский филиал
ордена Трудового Красного Знамени федерального государственного бюджетного образовательного учреждения высшего образования
«Московский технический университет связи и информатики»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ
для проведения практических занятий по дисциплине «Системы искусственного интеллекта»

Кафедра **«Информатика и вычислительная техника»**

Направление подготовки **09.03.01. Информатика и вычислительная техника**

Профиль **Вычислительные машины, комплексы, системы и сети, Программное обеспечение и интеллектуальные системы**

Формы обучения **очная, заочная**

Разработала:
Доцент кафедры ИВТ Швидченко С.А.

Содержание

Практическое занятие №1. Модели нейронов. Элементы нейронных сетей и задача разделения двух классов. Аппроксимация функций.

Аннотация: Рассматриваются структура и функции различных моделей нейрона: персептрон, сигмоидальный нейрон, адалайн, Паде-нейрон, нейрон с квадратичным сумматором, сигма-пи нейроны, нейрон Хебба, стохастическая модель нейрона, кубические модели нейронов.

Элементы нейронных сетей и задача разделения двух классов

Персептрон

Простой персептрон — это нейрон МакКаллока-Питса (рис.1). Весовые коэффициенты входов сумматора, на которые подаются входные сигналы $x_i, i = 1, \ldots, N$ обозначаются w_i, а пороговое значение — w_0. Нелинейная функция активации f персептрона является ступенчатой, вследствие чего выходной сигнал нейрона может принимать только два значения — 0 и 1 в соответствии с правилом

$$f(u) = \begin{cases} 1 \text{ для } u \geqslant 0 \\ 0 \text{ для } u < 0 \end{cases} \quad 1)$$

или -1 и 1 в соответствии с правилом

$$f(u) = \begin{cases} 1 \text{ для } u \geqslant 0 \\ -1 \text{ для } u < 0 \end{cases} \quad 2)$$

где u обозначает выходной сигнал сумматора

$$u = \sum_{i=0}^{N} w_i x_i. \quad 3)$$

В формуле (3) предполагается $x_0 = 1$.

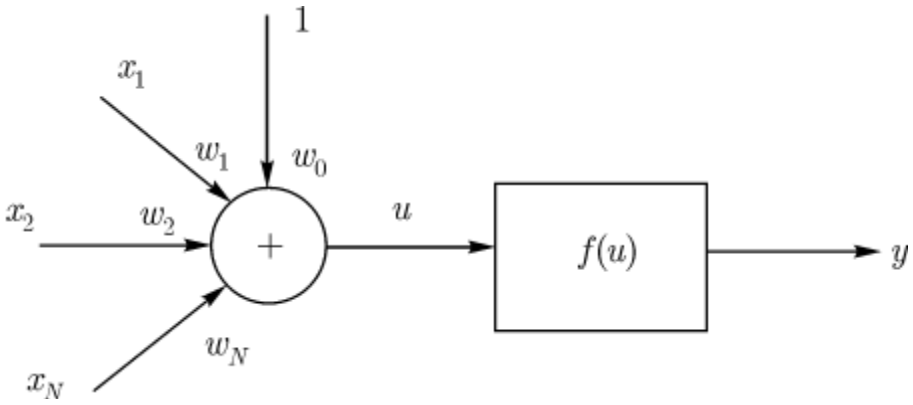


Рис. 1. Нейрон МакКаллока-Питтса

Обучение персептрона состоит в таком подборе весов w_i, чтобы выходной сигнал y совпадал с заданным значением $d \in \{0, 1\}$ или $d \in \{-1, 1\}$.

С персептроном связана задача четкого разделения двух классов по обучающей выборке, которая ставится следующим образом: имеется два набора векторов $X^1, \ldots, X^n$ и $Y^1, \ldots, Y^m$. Заранее известно, что X^i, $i = 1, \ldots, n$ относятся к первому классу, а Y^j, $j = 1, \ldots, m$ - ко второму. Требуется построить решающее правило, т.е. определить такую функцию $f(X)$, что при $f(X) > 0$ вектор X относится к первому классу, а при $f(X) < 0$ - ко второму.

Сигмоидальный нейрон

Нейрон сигмоидального типа имеет структуру, подобную модели МакКаллока-Питса, с той разницей, что функция активации является непрерывной и может быть выражена в виде сигмоидальной униполярной или биполярной функции. Униполярная функция, как правило, представляется формулой (рис.2)

$$f(x) = 1/(1 + exp(-\beta x)),$$

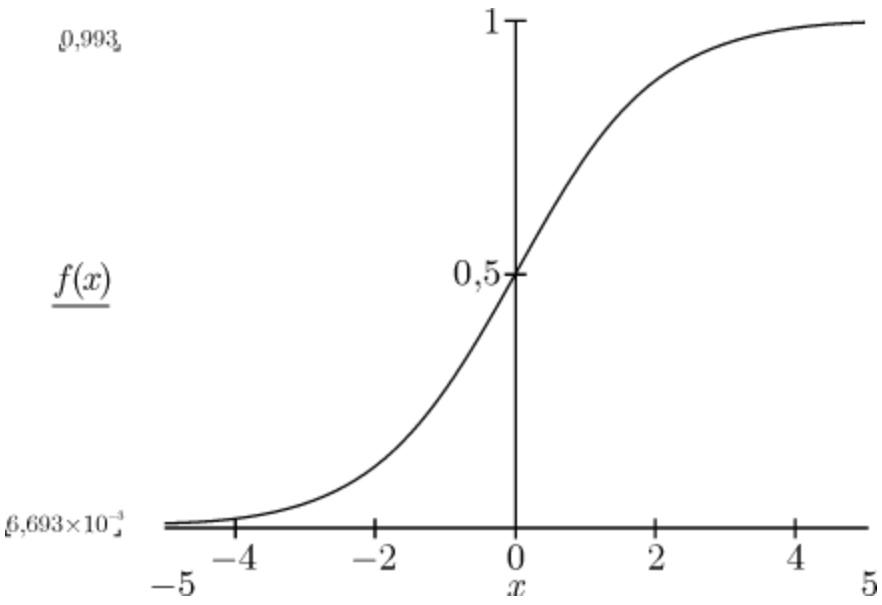


Рис. 2. Униполярная функция

тогда как биполярная функция задается в виде (рис.3) $f(x) = tanh(\beta x)$.

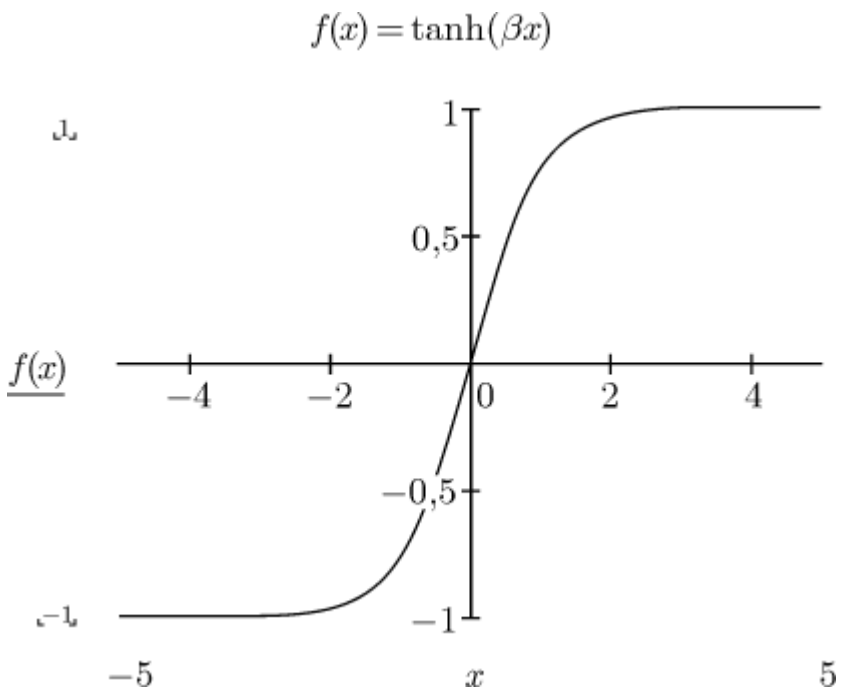


Рис. 3. Биполярная функция

Параметр β влияет на крутизну графика функции $f(x)$. При $\beta \to \infty$ сигмоидальная функция превращается в функцию ступенчатого типа, идентичную функции активации персептрона. На практике чаще всего используется значение $\beta = 1$.

Важным свойством сигмоидальной функции является ее дифференцируемость. Для униполярной функции имеем

$$df(x)/dx = \beta f(x)(1 - f(x))$$

тогда как для биполярной функции

$$df(x)/dx = \beta(1 - f(x))^2.$$

Применение непрерывной функции активации позволяет использовать при обучении *градиентные методы* оптимизации. Проще всего реализовать *метод наискорейшего спуска*, в соответствии с которым уточнение вектора весов $w = [w_0, w_1, \ldots, w_N]^T$ проводится в направлении отрицательного градиента целевой функции $E = (y - d)^2/2$, где

$$y = f(u) = f(\sum_{i=0}^{N} w_i x_i).$$

Компонента градиента имеет вид

$$\nabla_i E = dE/dw_i = ex_i df(u)/du,$$

где $e = y - d$ означает разницу между фактическим и ожидаемым значением выходного сигнала нейрона. Если ввести обозначение $\delta = e \cdot df(u)/du$, то можно получить выражение, определяющее i-ю составляющую градиента в виде

$$\nabla_i E = \delta x_i.$$

Значения весовых коэффициентов уточняются по формуле

$$w_i(t+1) = w_i(t) - \alpha \delta x_i,$$

где $\alpha \in (0,1)$.

Применение градиентного метода для обучения нейрона гарантирует достижение только локального минимума. Для выхода из окрестности локального минимума результативным может оказаться обучение с моментом. В этом методе процесс уточнения весов определяется не только информацией о *градиенте функции*, но и предыдущим изменением весов. Подобный способ может быть задан выражением

$$\triangle w_i(t+1) = -\alpha \delta x_i + \beta \triangle w_i(t),$$

в котором первый член соответствует обычному методу наискорейшего спуска, тогда как второй член, называемый моментом, отражает последнее изменение весов и не зависит от фактического значения градиента. Значение β выбирается из интервала (0,1).

Аппроксимация функций

Нейрон типа "адалайн"

В нейроне типа "адалайн" (□D□ptive LInear *Neuron* - адаптивный линейный нейрон) адаптивный подбор весовых коэффициентов осуществляется в процессе минимизации квадратичной ошибки, определяемой как

$$E(w) = e^2/2 = [d - (\sum_{i=0}^{N} w_i x_i)]^2/2.$$

В связи с выполнением условия дифференцируемости целевой функции стало возможным применение алгоритма градиентного обучения. Значения весовых коэффициентов уточняются следующим способом

$$w_i(t+1) = w_i(t) + \alpha e x_i.$$

Паде-нейрон

Паде-нейрон вычисляет произвольную дробно-линейную функцию вектора x. Так же, как и для адаптивного сумматора, числитель и знаменатель можно сделать линейными функциями x:

$$Ux/Lx, \quad Ux = \sum_{i=0}^{N} U_i x_i, \quad Lx = \sum_{i=0}^{N} L_i x_i.$$

Паде-нейрон может использоваться как обобщение нейрона типа "адалайн" в тех случаях, когда линейных функций становится недостаточно, в частности, в задачах интерполяции эмпирических зависимостей.

В случае Паде-нейрона квадратичная ошибка определяется как

$$E(U,L) = e^2/2 = (d - Ux/Lx)^2/2$$

и значения весовых коэффициентов уточняются по следующим формулам

$$U_i(t+1) = U_i(t) + \alpha e x_i / \sum_{j=0}^{N} L_j x_j,$$

$$L_i(t+1) = L_i(t) - \alpha e x_i \sum_{j=0}^{N} U_j x_j / (\sum_{j=0}^{N} L_j x_j)^2.$$

Нейрон с квадратичным сумматором

Квадратичный сумматор может вычислять произвольный полином второго порядка от вектора входных сигналов

$$Q(x) = \sum_{i,j} q_{ij} x_i x_j + \sum_i p_i x_i + r.$$

Для многомерных нормальных распределений нейрон с квадратичным сумматором является наилучшим классификатором. Минимум вероятности ошибки дает квадратичная разделяющая поверхность:

если $Q(x) > 0$, то объект принадлежит первому классу;

если $Q(x) \leqslant 0$, то объект принадлежит второму классу (при условии правильного выбора коэффициентов Q(□)).

Квадратичная ошибка здесь определяется как

$$E(q_{ij}, p_i, r) = e^2/2 = (d - Q(x))^2/2.$$

Коэффициенты квадратичного сумматора уточняются по формулам

$$q_{ij}(t+1) = q_{ij}(t) + 2\alpha e x_i x_j,$$
$$p_i(t+1) = p_i(t) + \alpha e x_i,$$
$$r(t+1) = r(t) + \alpha e.$$

Недостаток такого классификатора - большое число настраиваемых параметров.

Сигма-Пи нейроны

Выше были рассмотрены нейроны с линейной и квадратичной функциями активации. Сигма-пи нейроны являются их обобщением на случай представления функции активации u полиномом степени N, N - число входов нейрона:

$$U = \sum_{k=1}^{M} w_k \prod_{i \in I_k} x_i,$$

где I_k - множество индексов, содержащее одну из возможных 2^N комбинаций первых N целых чисел, $M = 2^N$.

Модель нейрона Хебба

Структурная схема нейрона Хебба соответствует стандартной форме модели нейрона (рис.1). Д.Хебб предложил формальное правило, в котором вес w_i нейрона изменяется пропорционально произведению его входного и выходного сигналов

$$\Delta w_i = \alpha x_i y,$$

где $\alpha \in (0, 1)$ - коэффициент обучения.

При обучении с учителем вместо выходного сигнала y используется ожидаемая от этого нейрона реакция d. В этом случае правило Хебба записывается в виде

$$\Delta w_i = \alpha x_i d,$$

В каждом цикле обучения происходит суммирование текущего значения веса и его приращения Δw_i:

$$w_i(t+1) = w_i(t) + \Delta w_i.$$

В результате применения правила Хебба веса нейрона могут принимать произвольно большие значения. Один из способов стабилизации процесса обучения по правилу Хебба состоит в учете последнего значения w_i, уменьшенного на коэффициент забывания γ. При этом правило Хебба представляется в виде

$$w_i(t+1) = w_i(t)(1-\gamma) + \Delta w_i.$$

Значение γ выбирается из интервала (0,1) и чаще всего составляет некоторый процент от коэффициента обучения α. Рекомендуемые значения коэффициента забывания - $\gamma < 0.1$, при которых нейрон сохраняет большую часть информации, накопленной в процессе обучения, и получает возможность стабилизировать значения весов на определенном уровне.

Стохастическая модель нейрона

В *стохастической модели* выходное состояние нейрона зависит не только от взвешенной суммы входных сигналов, но и от некоторой случайной переменной, значения которой выбираются при каждой реализации из интервала (0,1).

В *стохастической модели* нейрона выходной сигнал y принимает значения ± 1 с вероятностью

$$P(y=1) = 1/(1+\exp(-2\beta u)),$$
$$P(y=-1) = 1/(1+\exp(2\beta u)),$$

где u обозначает взвешенную сумму входных сигналов нейрона, а β - положительная константа, которая чаще всего равна 1. Процесс обучения нейрона в *стохастической модели* состоит из следующих этапов:

1) расчет взвешенной суммы

$$u = \sum_{i=0}^{N} w_i x_i$$

для каждого нейрона сети.

2) расчет вероятности P того, что y принимает значение ± 1.

3) генерация значения случайной переменной $R \in (0,1)$ и формирование выходного сигнала y, если $R < P(y)$, или $-y$ в противном случае.

При обучении с учителем по правилу Видроу-Хоффа адаптация весов проводится по формуле

$$\Delta w_i = \alpha x_i (d - y).$$

Нейроны типа W□□

Нейроны типа *WTA* (Winner □akes □ll — "Победитель получает все") имеют входной модуль в виде адаптивного сумматора. Выходной сигнал i -го сумматора определяется по формуле

$$u_i = \sum_{j=0}^{N} w_{ij} x_j.$$

По результатам сравнения сигналов $u_i, i = 1, 2, \ldots, N$ отдельных нейронов победителем признается нейрон, у которого u_i оказался наибольшим. Нейрон-победитель вырабатывает на своем выходе состояние 1, а остальные (проигравшие) нейроны переходят в состояние 0.

Для обучения нейронов *WTA* учитель не требуется. На начальном этапе случайным образом выбираются весовые коэффициенты w_{ij} каждого нейрона, нормализуемые относительно 1 по формуле

$$w_{ij} \leftarrow w_{ij}/(w_{i1}^2 + w_{i2}^2 + \ldots + w_{iN}^2)^{1/2}.$$

После подачи входного вектора x, компоненты которого нормализованы по формуле

$$x_{ij} \leftarrow x_{ij}/(x_{i1}^2 + x_{i2}^2 + \ldots + x_{iN}^2)^{1/2},$$

определяется победитель этапа. Победитель переходит в состояние 1, что позволяет произвести уточнение весов его входных линий w_{ij} по правилу

$$w_{ij}(t+1) = w_{ij}(t) + \alpha[x - w_{ij}(t)].$$

Проигравшие нейроны формируют на своих выходах состояние 0, что блокирует процесс уточнения их весовых коэффициентов.

Выходной сигнал i -го нейрона может быть описан векторным отношением

$$u_i = w_i^T x = ||w_i||||x|| \cos\varphi_i.$$

Поскольку $||w_i|| = ||x|| = 1$, значение u_i определяется углом между векторами x и $w_i, u_i = cos\varphi_i$. Поэтому победителем оказывается нейрон, вектор весов которого оказывается наиболее близким текущему обучающему вектору x. В результате победы нейрона уточняются его весовые коэффициенты, значения которых приближаются к значениям текущего обучающего вектора x.

Следствием конкуренции нейронов становится самоорганизация процесса обучения. Нейроны уточняют свои веса таким образом, что при предъявлении группы близких по значениям входных векторов победителем всегда оказывается один и тот же нейрон. Системы такого типа чаще всего применяются для классификации векторов.

Кубические модели нейронов

Вектор x входных двоичных сигналов рассматривается как адрес ячейки памяти, содержимое которой равно 0 или 1. Для размерности N вектора x существует 2^N возможных адресов.

Можно рассматривать ячейки памяти, как вершины N -мерного *гиперкуба*. Ячейки памяти получают значения независимо друг от друга. Полезно рассматривать ячейки памяти как содержащие поляризованные двоичные значения ± 1. Тогда работа кубического модуля описывается следующим образом.

Двоичный вход x используется как адрес памяти, поляризованная двоичная величина считывается и конвертируется в неполяризованную форму функцией h (см. формулу 1). Обозначим значения по адресу x через S_χ, так что $y = h(S_\chi)$. Такие модули мы будем называть кубическими, чтобы подчеркнуть геометрическое представление множества адресов значений активации как множество вершин *гиперкуба*.

Запись активации в замкнутой форме

Рассмотрим двухвходовый кубический модуль. Существует 4 значения активации $\{S_{00}, S_{01}, S_{10}, S_{11}\}$. Выражение для активации будет иметь следующий вид:

$$u = S_{00}(1 - x_1)(1 - x_2) + S_{01}(1 - x_1)x_2 + S_{10}x_1(1 - x_2) + S_{11}x_1x_2,$$

$x = (x_1, x_2)$ - входной вектор. Такая запись вызвана тем, что только одно из произведений в сумме должно быть ненулевым. Для поляризованных входов x_1 и x_2 активация

$$u = [S_{00}(1 - x_1)(1 - x_2) + S_{01}(1 - x_1)(1 + x_2) + \\ + S_{10}(1 + x_1)(1 - x_2) + S_{11}(1 + x_1)(1 + x_2)]/4.$$

В случае N - входового модуля получим

$$u = [\sum_{\chi} S_{\chi} \prod_{i=1}^{N} (1 + x_i)]/2^N.$$

Обучение кубических нейронов

Кубические нейроны обучаются путем изменения содержимого ячейки их памяти. Обозначим через `+' операцию инкремента-установки содержимого ячейки в +1, через `-' операцию декремента-установки в -1.

Пусть в начальном состоянии все ячейки кубического нейрона установлены в ноль. Обозначим ячейки, адресуемые обучающей выборкой, как центральные ячейки или центры. Ячейки, близкие к центрам в смысле расстояния Хемминга, будем настраивать на те же или близкие к ним значения, что и сами центры, т.е. должна происходить кластеризация значений ячейки вокруг центра. Это условие должно выполнятся для сети из кубических нейронов. Алгоритм обучения строит так называемое разбиение Вороного, при котором значение в ячейке определяется значением в ближайшем центре, а ячейки, равноудаленные от центров, остаются установленными в ноль. Кубические нейроны допускают большую функциональность, чем полулинейные, и поэтому, возможно, позволяют решать те же задачи при меньшем количестве модулей.

Контрольные вопросы ПЗ1(ПК-4):

1. Основы искусственного интеллекта. Основные понятия и определения
2. Основы искусственного интеллекта. . Область применения.
3. Краткий исторический обзор развития работ в области ИИ.
4. Как указать в программе Проводник директорию, содержащую все файлы Lazarus. – проекта?
5. Как добавить в проект еще одну форму и указать связанный с ней файл-модуль?
6. Как удалить из проекта форму (с помощью □r□ject Manager)?
7. Обращение к справочной системе □elp
8. Меню и команды Lazarus: меню File, меню Edit, меню Search, меню View, меню Run, меню c□mp□nent, меню □□ls.
9. Полоска кнопок быстрого доступа Speedbar, локальные меню. speedmenu.
10. Работа с формами, палитра компонентов, Object Inspect□r, написание кода.
11. Компиляция проекта, интегрированный отладчик, файлы, создаваемые системой.

Практическое занятие №2. Модели нейронов. Часть 2. Элементы нейронных сетей и задача разделения двух классов. Аппроксимация функций.

Аннотация: Рассматриваются: решение задачи линейного разделения двух классов методом центров масс, алгоритм обучения персептрона, виды обучения, геометрическая интерпретация задачи разделения двух классов.

Линейное разделение классов

состоит в построении линейного решающего правила, т.е. такого вектора $w = (w_0, \ldots, w_n)$, где w_0 — порог, что при $wx > 0$ *вектор* x относится к первому классу, а при $wx \leqslant 0$ — ко второму.

Разделение центров масс - простейший способ построения решающего правила. Суть этого способа заключается в вычислении вектора весов персептрона *по* следующей формуле

$$w = (\sum_{i=1}^{n} X^i - \sum_{j=1}^{m} Y^j)/(n+m),$$

где $X^i, i = 1, \ldots, n$ относятся к первому классу, а $Y^j, j = 1, \ldots, m$ - ко второму.

Линейные решающие правила, построенные на основании разделения центров масс, могут ошибаться на примерах из обучающей выборки даже в тех случаях, когда существует и безошибочное линейное разделение. Однако метод центров масс полезен как средство определения начального значения вектора весов для алгоритма обучения персептрона.

Алгоритм обучения персептрона по отдельным примерам

1. При изначально заданных значениях весов w_i на вход нейрона подается обучающий *вектор* x и рассчитывается *значение* выходного сигнала y. *По* результатам сравнения y с d уточняются значения весов.

2. Если $y = d$, то $w_i, i = 1, \ldots, N$ не изменяются.

3. Если $y = 0$, а $d = 1$, то значения весов уточняются *по* формуле

$$w_i(t+1) = w_i(t) + \alpha x_i, \alpha \in (0, 1),$$

где α — коэффициент обучения, t — номер предыдущего *цикла*.

4. Если $y = 1$, а $d = 0$, то значения весов уточняются *по* формуле

$$w_i(t+1) = w_i(t) - \alpha x_i.$$

В обобщенной форме обучение персептрона на векторе x выражается формулой

$$w_i(t+1) = w_i(t) + \alpha(d-y)x_i, i = 1, \ldots, N.$$

По завершении уточнения весовых коэффициентов представляются очередной обучающий *вектор* x и связанное с ним *ожидаемое значение* d, и значения весов уточняются заново. Этот процесс многократно повторяется на всей обучающей выборке, пока не будут ликвидированы различия между всеми значениями y и соответствующими им *ожидаемыми значениями* d.

Обучение по всему задачнику

Построим обучающую выборку

$$(V^1, \ldots, V^n, V^{n+1}, \ldots, V^{n+m}) = (X^1, \ldots, X^n, -Y^1, \ldots, -Y^m).$$

В обучающей выборке выделяются все $V_i, i \in (1, \ldots, n, n+1, \ldots, n+m)$, для которых не выполняется *неравенство* $(V^i, w) > 0$, где w — *вектор* весовых коэффициентов нейрона. Обозначим это множество через *Err*. *Вектор* w модифицируется только после проверки всей обучающей выборки:

$$w = w + \alpha \sum_{V^i \in Err} V^i.$$

Не требуется хранить все множество *Err* - достаточно накапливать сумму тех V^i, на которых *персептрон* ошибается:

$$\Delta w = w + \alpha \sum_{V^i \in Err} V^i.$$

Как показывают испытания, обучение *по* всему задачнику, как правило, сходится быстрее, чем обучение *по* отдельным примерам.

Промежуточный вариант: обучение по страницам

Обучающее множество разбивается на подмножества (страницы) и задается последовательность прохождения страниц: столько-то циклов *по* первой странице, потом столько-то *по* второй и т. д. *Коррекция* вектора w проводится после прохождения страницы. Задачник разбивается на страницы *по* различным эвристическим правилам, например, *по* правилу "от простого к сложному". Как показывает практика, чаще всего наилучшим является обучение *по* всему задачнику, иногда (при большом задачнике) - обучение *по* страницам, размеры которых определяются объемом доступной оперативной памяти.

Геометрическая интерпретация линейного разделения классов

Пусть в нейроне в качестве функции активации используется ступенчатая *функция* (см. формулу (1) Лекции 2). Линейное разделяющее правило делит входное *пространство* на две части гиперплоскостью, классифицируя входные векторы как относящиеся к 1-му классу (выходной сигнал - 1) или 2-му классу (выходной сигнал - 0). Критическое условие классификации (уравнение разделяющей *гиперплоскости*)

$$(w, x) = \sum_{i=0}^{N} w_i x_i = 0$$

В $\{N\}$-мерном пространстве (пространстве входных сигналов) разделяющая *гиперплоскость* перпендикулярна вектору $w' = (w_1, \ldots, w_N)$. *Вектор* входных сигналов $x' = (x_1, \ldots, x_N)$ дает *выход* 1, если его *проекция* $x'_w = (x', w')/||w'||$ на *вектор* w' больше, чем *расстояние* $-w_0/||w'||$ от нуля до *гиперплоскости*. В $N+1$-мерном (расширенном) пространстве *гиперплоскость*, описываемая уравнением $(w, x) = 0$, ортогональна вектору w и проходит через начало координат *пространства признаков* (образов).

Пример

В двухмерном пространстве входных сигналов уравнение *гиперплоскости* имеет вид

$$w_0 + w_1 x_1 + w_2 x_2 = 0.$$

При $w_1 = w_2 = 1$ и $w_0 = -1.5$ получаем уравнение $x_1 + x_2 - 1,5 = 0$ *гиперплоскости*, которая представлена на рис.1 пунктирной линией, пересекающей оси координат в точках (1.5, 0) и (0, 1.5) соответственно. Здесь: $w = (1, 1)$ — нормаль к разделяющей *гиперплоскости*; P — *вектор*, относящийся к первому классу, поскольку *проекция* (w, P) вектора P на нормаль w больше $-w_0/||w||$; Q — *вектор*, относящийся ко второму классу, поскольку $(w, Q) < -w_0/||w||$.

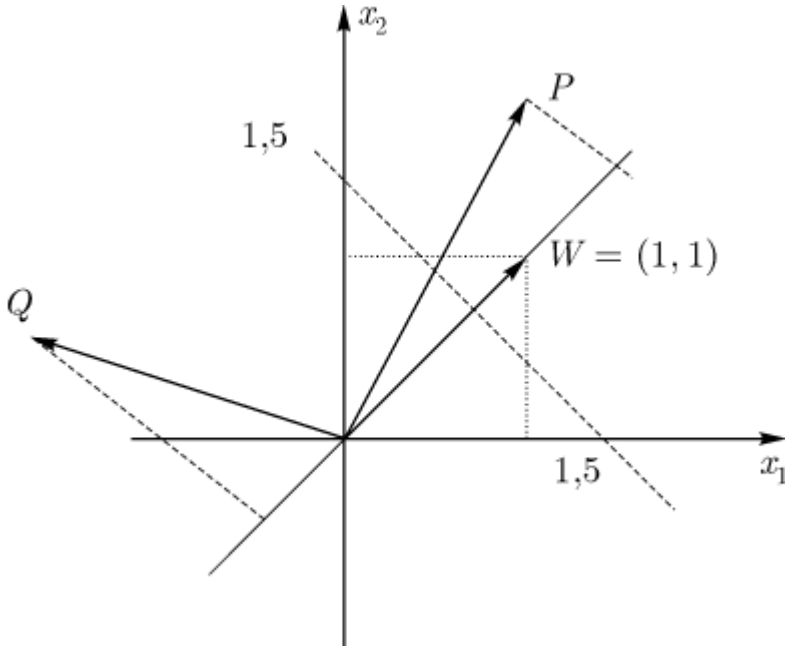


Рис. 1.

Настройка весового вектора

Мы требуем, чтобы *вектор* весов в расширенном пространстве был ортогонален решающей *гиперплоскости*, и *плоскость* проходила через начало координат. Обучающую выборку (задачник) для нейрона можно рассматривать как множество пар (V, d), где V - *входной вектор*, d - *класс* (*выход*, принимающий одно из двух значений, например, 0 или 1), которому принадлежит V. Такой тип обучения называется обучением с учителем, т.к. мы сообщаем сети, каким должен быть выходной сигнал для каждого вектора входных сигналов.

Пусть для некоторого V выполняется $d = 1$, но *выход* сети

$$y = f[(V, W)] = 0,$$

где $f(u) = 1$ при $u > 0$, и $f(u) = 0$ при $u < 0$, т.е. $(V, W) < 0$ (угол φ на рис.2 между векторами V и W больше $\pi/2$). Чтобы исправить ситуацию, нужно повернуть *вектор* весов W, приближая его направление к направлению вектора V. В то же время изменение не должно быть слишком резким, чтобы не испортить уже выполненное обучение. Мы достигнем обеих целей, если добавим к вектору W часть вектора V, чтобы получить новый *вектор*

$$W' = W + \alpha V, \quad 0 < \alpha < 1.$$

Предположим теперь, что $d = 0$, а $y = 1$ (угол φ на рис.2 между векторами V и W меньше $\pi/2$). Теперь нужно увеличить угол между W и V, что получается путем вычитания части V из W:

$$W' = W - \alpha V.$$

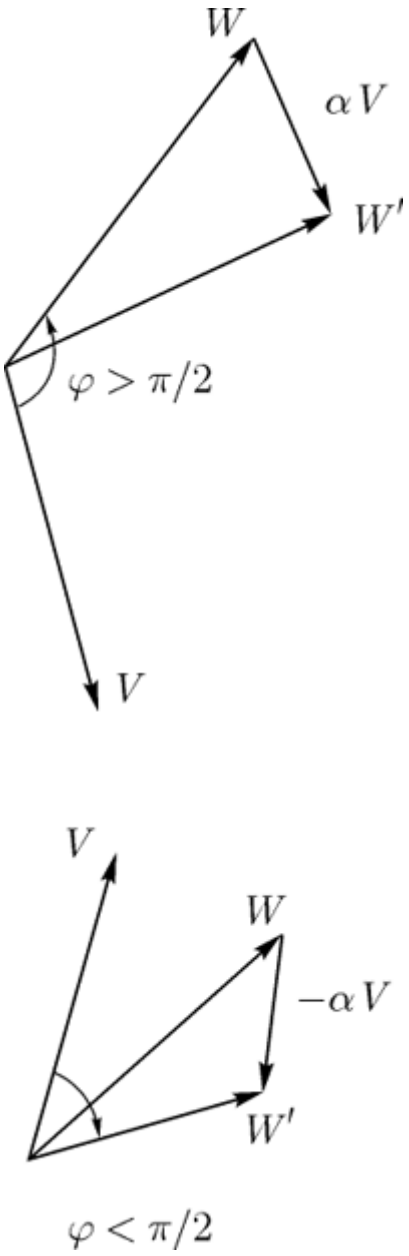


Рис. 2. Настройка вектора весов

Результирующая *запись* имеет вид:

$$W' = W + \alpha(d - y)V.$$

Параметр α называется скоростью обучения.

Алгоритм обучения нейрона (персептрона) будет иметь вид:

repeat
for $\forall(V, d)$
  begin
    y = h[(W,V)]; $if y \neq d then W' = W + \alpha(d - y)V;$
  end
until $(y = d\ for\ \forall(V, d))$

Контрольные вопросы ПЗ2(ПК-4):

1. Модели и методы решения задач.
2. Сетевые модели. Продукционные модели.
3. Модели и методы решения задач. Сценарии.
4. Интеллектуальный интерфейс.
5. Классификация уровней понимания.
6. Нейрофизиологическая аналогия.
7. Реализация булевых функций нейронными сетями.
8. Предопределенные типы данных: перечислимые типы, вещественные типы, типы данных, специфичные для Wind□ws, приведение и преобразование типов, массивы, длинные строки.
9. Элементы управления редактированием..
10. Классы и сокрытие информации.
11. Классы и модули.
12. Модули и область видимости.
13. Модули и программы
14. информация о типе на этапе выполнения.
15. буксировка из одного компонента в другой.

Практическое занятие №3. Задача нелинейного разделения двух классов. Метод максимума правдоподобия. Нейрофизиологическая аналогия.

Метод максимума правдоподобия

Рассмотрим задачу разделения двух классов, с каждым из которых связано вероятностное распределение в пространстве векторов x значений признаков. Будем обозначать плотности этих распределений $P(x|C_i), i = 1, 2, C_i$ - событие, состоящее в том, что *объект* принадлежит $\{ i \}$-му классу. Нас интересует апостериорная *вероятность*: $P(C_i|x)$ — *вероятность* принадлежности объекта к $\{ i \}$-му классу при условии, что он характеризуется вектором признаков x. Известная из теории вероятности формула Байеса дает

$$P(C_i|x) = P(C_i)P(x|C_i)/\sum_j P(C_j)P(x|C_j)$$

где $P(C_i)$ — *вероятность* появления объектов $\{ i \}$-го класса. Для нормальных k - мерных распределений

$$P(x|Ci) = 1/\{(2\pi)^{k/2}(det\sum^i)^{1/2}exp[-\tfrac{1}{2}(x-M^i),(\sum^i)^{-1}(x-M^i)]\},$$

где M^i — математическое ожидание x в $\{ i \}$-м классе, $\{\sum^i\}$ — ковариационная *матрица* для $\{ i \}$-го класса. В результате обработки данных находят статистические оценки $\{\sum^i\}$ и M^i: пусть для $\{ i \}$-го класса имеются векторы $x^1 \dots x^r$, тогда полагаем

$$M^i = (\sum_{j=1}^{r} x^j)/r, (\sum^i)_{pq} = \frac{1}{r}\sum_{j=1}^{r}(x_p^j - M_p^i)(x_q^j - M_q^i).$$

Минимизация в формуле Байеса дает простое *решающее правило*: x принадлежит i -му классу, если $P(C_i|x) > P(C_j|x)$ для всех $j \neq i$, т.е выбирается такой *класс*, для которого *вероятность* $P(C_i|x)$ максимальна. Поскольку в формуле Байеса для всех C_i знаменатель общий, то *решающее правило* приобретает следующий вид: выбираем то i, для которого $P(C_i)P(x|C_i)$ максимально. Для нормального распределения удобно прологарифмировать эту величину. Окончательно получаем:

x принадлежит i -му классу, если среди величин

$$P_j = lnP(C_j) - (lndet\sum^j)/2 - [(x - M^j),(\sum^j)^{-1}(x - M^j)]/2$$

величина P_i - максимальная. Таким образом, разделяющей является поверхность второго порядка, а операцию разделения на два класса выполняет квадратичный адаптивный *сумматор* в комбинации с пороговым нелинейным элементом. Пороговый элемент вычисляет ступенчатую функцию $f(P_1 - P_2)$, в результате для первого класса получим ответ 1, для второго - 0.

Нейрофизиологическая аналогия

Идея использования НС с квадратичными сумматорами для улучшения способности сети к обобщению базируется на хорошо известном факте индукции в естественных НС, когда возбуждение в одних областях мозга влияет на возбуждение в других. Простейшей формализацией этого является введение коэффициента, пропорционального сигналу от j -го нейрона, в величину веса i -го сигнала k -го нейрона. Снабдив такое *произведение* весом q_{ij} — "коэффициентом индукции", получим рассматриваемую архитектуру

$$y = f[Q(x) + L(x) + P],$$

где $Q(x)$ и $L(x)$ - соответственно квадратичная и линейная *функция*, $P = const$, f - *функция* активации нейрона. *Коэффициенты* функций Q, L и константа P являются подстроечными параметрами, определяющимися в ходе обучения.

Реализация булевых функций нейронными сетями

Простой *персептрон* (*нейрон* МакКаллока-Питса) с весовым вектором $w = (-0.5, 1, 1)$ реализует *гиперплоскость*

$$x_1 + x_2 = 0.5$$

и булеву функцию ИЛИ от двух аргументов x_1 и x_2, каждый из которых может быть нулем или единицей. При $w = (-1.5, 1, 1)$ *персептрон* реализует *гиперплоскость*

$$x_1 + x_2 = 1.5$$

и булеву функцию И. Однако, *персептрон* не может воспроизвести даже такую простую функцию как ИСКЛЮЧАЮЩЕЕ ИЛИ. Она принимает *значение* единицы, когда один из аргументов равен единице (но не оба) (табл.1).

Таблица 1. Булева функция ИСКЛЮЧАЮЩЕЕ ИЛИ

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Эту функцию реализует двухслойная нейронная *сеть*, представленная на рис.1 (сигнал $x_0 = 1$ не указан). Первый слой такой сети состоит из двух нейронов, каждый из которых реализует разделяющую *гиперплоскость* в двумерном пространстве входных данных. Первая *гиперплоскость* описывается уравнением

$$x_1 + x_2 = 0.5,$$

а вторая - уравнением

$$-x_1 - x_2 = -1.5.$$

Соответствующие векторы весов имеют вид $w^1 = (-0.5, 1, 1)$ и $w^2 = (1.5, -1, -1)$. *Нейрон* во втором слое реализует функцию И от двух выходных сигналов нейронов первого слоя.

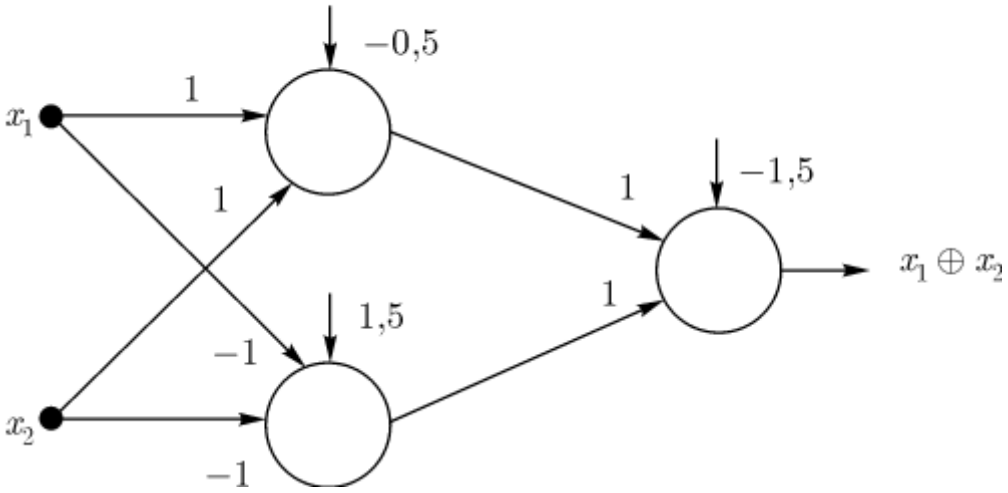


Рис. 1. Двухслойная сеть, реализующая функцию ИСКЛЮЧАЮЩЕЕ ИЛИ

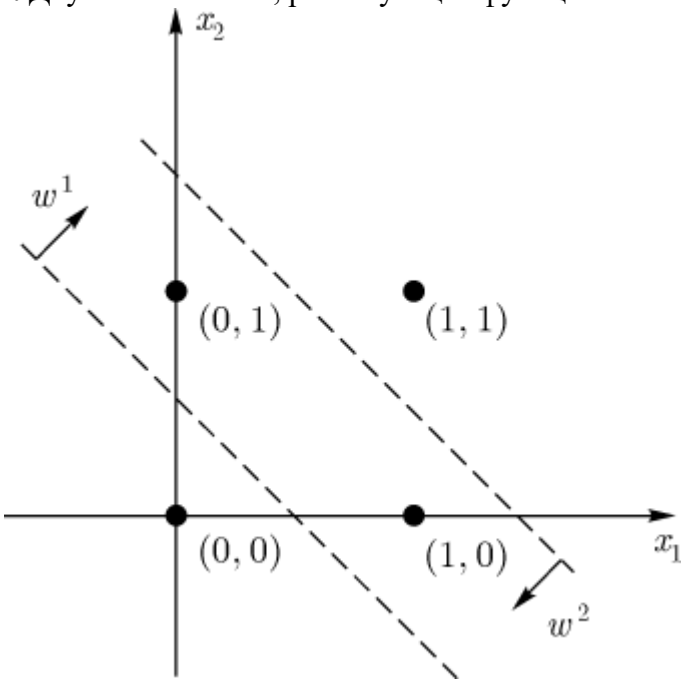


Рис. 2. Гиперплоскости, реализующие функцию ИСКЛЮЧАЮЩЕЕ ИЛИ

Выходным сигналом сети будет 1, если входные сигналы сети соответствуют точкам пространства входных сигналов, расположенным между вышеуказанными гиперплоскостями, т.е. точкам (0,1) и (1,0) (рис.2).

Выделение выпуклых областей

Серьезное ограничение разделяющих поверхностей однослойными сетями можно преодолеть, добавив дополнительные слои. Например, двухслойные сети, получаемые каскадным соединением однослойных сетей, способны выполнять более общие классификации, отделяя точки, содержащиеся в выпуклых ограниченных и неограниченных областях. Область выпуклая, если для каждых двух её точек соединяющий их *отрезок* целиком лежит в области. Область ограничена, если её можно заключить в некоторый шар.

Выше приведен пример выделения выпуклой области двумя гиперплоскостями (реализация функции ИСКЛЮЧАЮЩЕЕ ИЛИ). Аналогично в первом слое может быть использовано 3 нейрона с дальнейшим разбиением плоскости и созданием области треугольной формы (на рис. 3, 4, $w^1 = (0,1,0), w^2 = (0,0,1), w^3 = (-1,-1,-1)$, входы с нулевыми весами не указаны).

Включением достаточного числа нейронов во *входной* слой может быть образован выпуклый многоугольник (многогранник) желаемой формы. Так как такие многогранники образованы с помощью операций И над областями, задаваемыми разделяющими линиями (гиперплоскостями единичной размерности), то все они выпуклы.

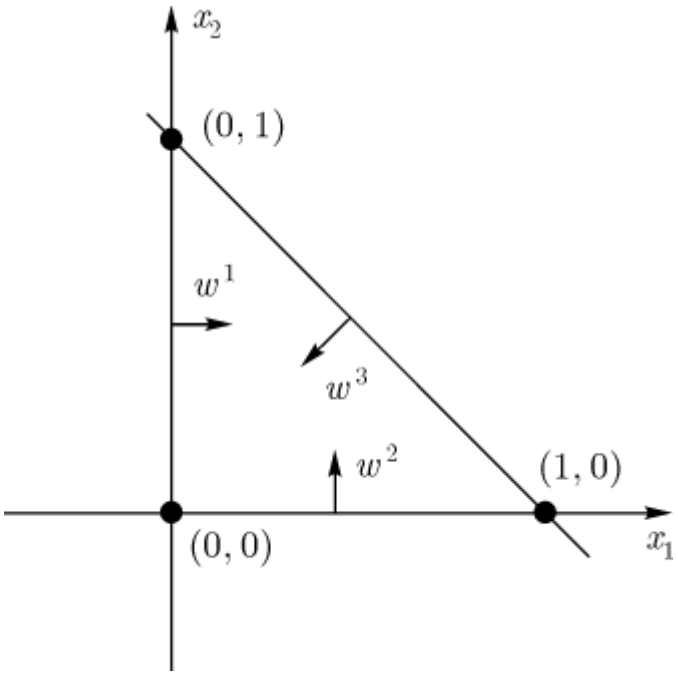


Рис. 3. Гиперплоскости, выделяющие на плоскости выпуклую (треугольную) область

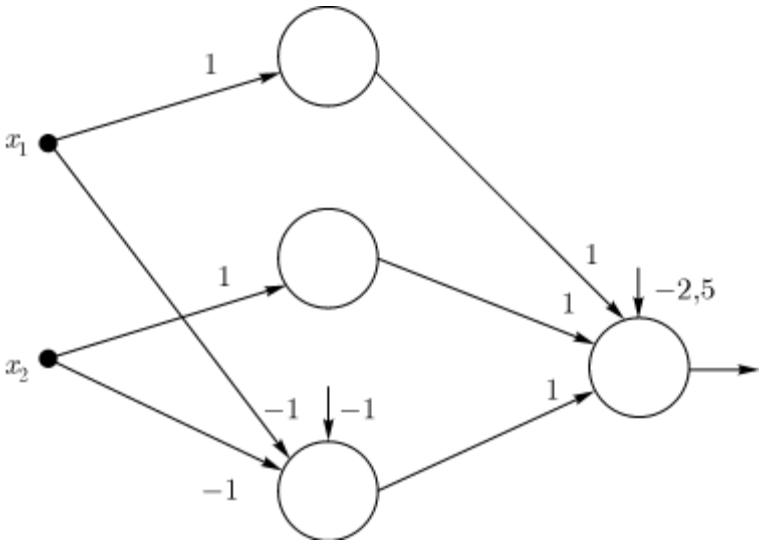


Рис. 4. Нейронная сеть, выделяющая на плоскости выпуклую (треугольную) область

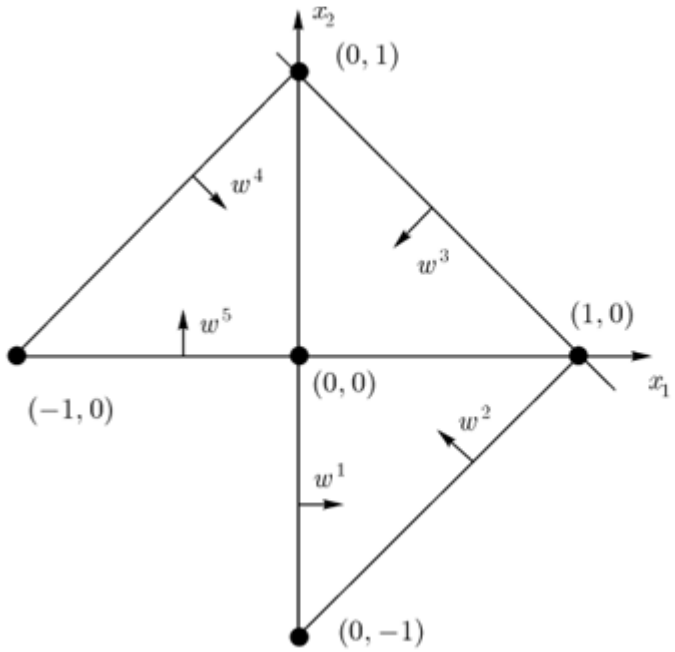
Выделение невыпуклых областей

Точки, не составляющие выпуклой области, не могут быть отделены от других точек пространства двухслойной сетью. *Нейрон* второго слоя не ограничен функцией И. Трехслойная *сеть* является более общей. Ограничения на выпуклость отсутствуют. *Нейрон* третьего слоя принимает в качестве входа набор выпуклых многоугольников, и их логическая комбинация может быть невыпуклой.

При добавлении нейронов число сторон многоугольников может неограниченно возрастать. Это позволяет аппроксимировать область любой формы с любой точностью. Вдобавок не все выходные области второго слоя должны пересекаться. Следовательно, возможно объединить различные области, выпуклые и невыпуклые, выдавая на выходе 1 всякий раз, когда *входной вектор* принадлежит одной из них.

На рис.5 приведен пример выделения невыпуклой области, представленной в виде объединения двух треугольных областей. Пять нейронов первого слоя реализуют разделяющие гиперплоскости, два нейрона второго слоя реализуют трехвходовые функции И, *нейрон* третьего слоя реализует функцию ИЛИ. Весовые векторы, описывающие соответствующие гиперплоскости, имеют вид:

$$w^1 = (0, 1, 0), \quad w^2 = (-1, -1, 1), \quad w^3 = (-1, -1, -1),$$
$$w^4 = (-1, 1, -1), \quad w^5 = (0, 0, 1).$$



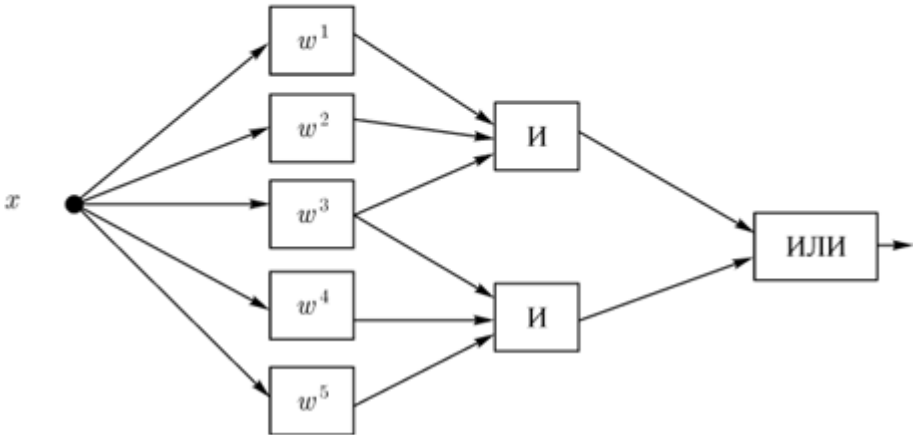


Рис. 5. Пример выделения невыпуклой области

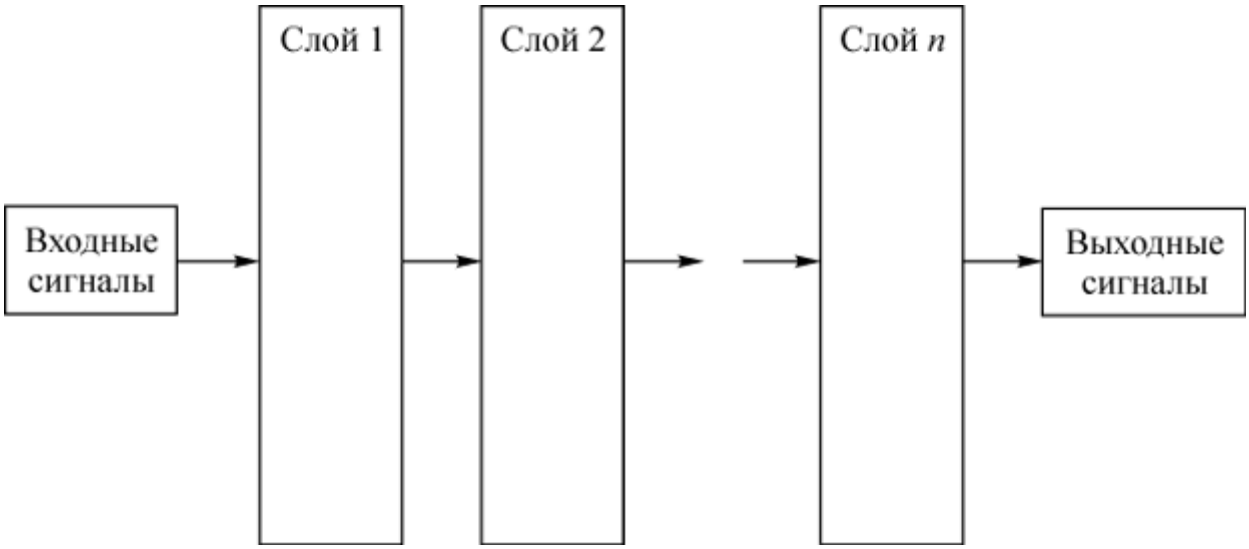
Контрольные вопросы ПЗ3(ПК-4):

1. Метод максимума правдоподобия.
2. Нейрофизиологическая аналогия.
3. Компоненты продукционных систем.
4. Стратегии решений организации поиска.
5. Представление простых фактов.
6. Примеры применения логики для представления знаний
7. Группирование кнопок опций.
8. Список со многими опциями.
9. Структура меню.
10. Различные роли элементов меню.
11. Изменение элементов меню.
12. События, связанные с мышью.
13. Черчение и рисование в системе wind□ws.
14. Построение инструментальной линейки.

Практическое занятие №4. Задача нелинейного разделения двух классов. Часть 2. Реализация булевых функций нейронными сетями. Выделение выпуклых областей.

Виды сетей

В многослойных (слоистых) сетях (рис.1) нейроны первого слоя получают входные сигналы, преобразуют их и передают нейронам второго слоя. Далее срабатывает второй слой и т.д., до n -ого, который выдает выходные сигналы для интерпретатора и пользователя. Если не оговорено противное, то каждый выходной сигнал i -го слоя подается на вход всех нейронов $i+1$ -го. Число нейронов в каждом слое может быть любым и никак заранее не связано с количеством нейронов в других слоях. Стандартный способ подачи входных сигналов: все нейроны первого слоя получают каждый *входной* сигнал. Наибольшее распространение получили трехслойные сети, в которых каждый слой имеет свое наименование: первый - *входной*, второй - скрытый, третий - выходной.



увеличить изображение

Рис. 1. Многослойная (слоистая) сеть

Монотонные слоистые сети - частный случай слоистых сетей с дополнительными условиями на связи и элементы. Каждый слой, кроме выходного, разбит на два блока - возбуждающий и тормозящий. Связи между слоями также подразделяются на два типа - возбуждающие (с положительными весами) и тормозящие (с отрицательными весами). Если от блока A к блоку C ведут только возбуждающие связи, то это означает, что любой выходной сигнал блока C является монотонной неубывающей функцией любого выходного сигнала блока A. Если же эти связи только тормозящие, то любой выходной сигнал блока C является монотонной невозрастающей функцией.

В полносвязной сети каждый *нейрон* передает свой выходной сигнал остальным нейронам, в том числе и самому себе. Выходными сигналами сети могут быть все или некоторые выходные сигналы нейронов после нескольких циклов функционирования сети. Все входные сигналы подаются всем нейронам. Для полносвязной сети *входной сумматор* нейрона фактически распадается на два: первый вычисляет линейную функцию от входных сигналов сети, второй - линейную функцию от выходных сигналов других нейронов, полученных на предыдущем шаге. Примером полносвязной сети является *сеть Хопфилда*.

Слоисто-циклические *(рекуррентные) сети* отличаются тем, что слои замкнуты в кольцо - последний передает свои выходные сигналы первому. Все слои равноправны и могут как получать

входные сигналы, так и выдавать выходные. Такие сети до получения ответа могут функционировать неограниченно долго, так же, как и полносвязные.

Слоисто-полносвязные сети состоят из слоев, каждый из которых, в свою *очередь*, представляет собой полносвязную *сеть*. При функционировании сигналы передаются от слоя к слою, и происходит обмен сигналами внутри слоя. В каждом слое процесс протекает следующим образом: прием сигналов с предыдущего слоя (или входных сигналов сети), обмен сигналами внутри слоя, передача последующему слою (или на *выход*). Подобные сети до получения ответа функционируют определенное число тактов, соответствующее количеству слоев, так же, как и слоистые сети.

Полносвязно-слоистые сети по структуре такие же, как и предыдущие, но функционируют по-другому. В них не разделяются фазы обмена внутри слоя и передачи следующему: на каждом такте нейроны всех слоев принимают сигналы от нейронов как своего, так и предыдущего, после чего передает сигналы как внутри слоя, так и последующему (или на *выход*). До получения ответа подобные сети могут функционировать неограниченно долго, так же, как и полносвязные.

Функционирование сетей

Сети периодического функционирования. Простейшие представления об этих сетях таковы.

В начальный момент состояния всех нейронов одинаковы, выходных сигналов нет. Подаются входные сигналы, определяющие *активность* сети (нулевой такт). Далее входные сигналы могут подаваться на каждом такте функционирования. На каждом такте могут сниматься выходные сигналы. После k тактов цикл функционирования заканчивается, и *сеть* возвращается в исходное состояние, готовая к новому циклу (акту). Между актами функционирования могут вставляться акты обучения. В общем случае, в результате *цикла* из k тактов *нейронная сеть* выдает в ответ на последовательность из k наборов входных сигналов последовательность k наборов выходных сигналов. Чаще используется упрощенный вариант: входные сигналы подаются только в самом начале, выходные снимаются в самом конце.

Для слоистых и слоисто-полносвязных сетей начальные слои по мере срабатывания освобождаются и могут заниматься новой задачей, пока последние слои заканчивают работу над предыдущей. Сети периодического функционирования по характеру использования напоминают ЭВМ: на вопрос следует ответ, причем воспроизводимый. Иначе обстоит дело с сетями непрерывного функционирования.

Непрерывное функционирование нейронной сети более соответствует имеющимся представлениям о поведении живых существ, чем периодическое. *Опыт* показывает, что, чередуя циклы функционирования и обучения, для таких сетей можно получить хорошие результаты адаптации. Для непрерывного функционирования необходимы сети с циклами: полносвязные, слоисто-циклические или полносвязно-слоистые.

Настройка нейронных сетей для решения задач

Тема данного раздела - формирование нейронных сетей для решения задач. Прежде чем приступить к поиску параметров сети, нужно поставить задачу, т.е. ответить на вопросы:

1. Какие сигналы сеть будет получать?
2. Как мы будем интерпретировать сигналы, поступающие от сети?

3. Как мы будем оценивать работу сети, если сеть обучается путем минимизации ошибок (т.е. что такое вектор ошибок и как вычисляется целевая функция — оценка функционирования сети)?

Ответы на данные вопросы воплощаются в спецустройствах или программах: в предобработчике, интерпретаторе ответов, оценке.

Итак, прежде чем формировать *сеть*, необходимо создать её окружение. В процессе обучения, кроме того, используются:

1. Обучающая выборка (система, работающая с исходными данными);
2. Учитель, модифицирующий параметры сети;
3. Контрастер (система, упрощающая нейронную сеть).

Предобработка данных

Нормировка и центрирование данных (предобработка) используются почти всегда (кроме тех случаев, когда данные представляют собой бинарные векторы с координатами 0,1 или ± 1, либо символьные последовательности). Цель этих преобразований - сделать так, чтобы каждая компонента вектора данных лежала в отрезке $[-1, 1]$ (или $[0, 1]$) или, по крайней мере, не слишком далеко выходила из этого отрезка, и её характерный разброс тоже был бы единичным.

Стандартные преобразования исходной выборки $x^p, p = 1, 2, \ldots, M$:

$x_i^p = [x_i^p - M(x_i^p)]/\sigma(x_i^p)$ или $x_i^p = [x_i^p - M(x_i^p)]/max|x_i^p - M(x_i^p)|,$

где x_i^p - i -я компонента вектора x^p,

$M(x_i^p) = (\sum x_i^p)/n$ - выборочная оценка математического ожидания x_i^p;

$\sigma(x_i^p) = \{\sum_i^p [x_i^p - M(x_i^p)]^2/n\}^{1/2}$ - выборочная оценка среднего квадратичного отклонения. Любое изменение выборки $\{x^p\}$ должно, согласно этим формулам, менять и нормировку. Нормировка и центрирование вписывают исходную выборку в куб со стороной 2, вершинами которого являются векторы с координатами $\pm 1.$

Интерпретация ответов сети

При интерпретации выходных сигналов сети необходимы аккуратность и порой изобретательность, ведь от этого истолкования зависят требования, которые мы предъявляем к работе НС. Удачная их формулировка может упростить обучение и повысить *точность* работы, неудачная — свести на нет предыдущие усилия.

Масштабирование является естественной операцией при обработке выходных сигналов. Стандартные (обезразмеренные) НС формируются так, чтобы их выходные сигналы лежали в интервалах $[-1, 1]$ (или $[0, 1]$). Если нам нужно получить сигнал в интервале $[a, b]$, то нужно преобразовать выходной сигнал $y \in [-1, 1]$:

$$y = (a + b)/2 + (b - a)y/2.$$

В задачах классификации наиболее распространено правило интерпретации "победитель забирает все": число нейронов равно числу классов, номер нейрона с максимальным сигналом интерпретируется как номер класса. К сожалению, если классов много, то этот наглядный метод является слишком расточительным, потребляет слишком много *выходных нейронов*.

Знаковая *интерпретация* требует только $k = log_2 m$ нейронов (m - *число классов*). Строится она так. Пусть $y_1, \ldots, y_k$ - совокупность выходных сигналов нейронов. Заменим в этой последовательности положительные числа единицами, а отрицательные - нулями. Полученную последовательность нулей и единиц рассматриваем как номер класса в двоичной записи.

Порядковая *интерпретация* является еще более емкой, чем знаковая. В ней с помощью k нейронов можно описать принадлежность к $k!$ классам (а не 2^k как для знаковой). Пусть $y_1, \ldots, y_k$ - выходные сигналы. Проведем их сортировку и обозначим через n_i номер i -го сигнала после сортировки (1 соответствует наименьшему сигналу, k - наибольшему). Перестановку $\sigma = (n_1, n_2, \ldots, n_k)$ рассмотрим как *слово*, кодирующее номер класса. Всего возможно $k!$ перестановок. Этим интерпретатором можно пользоваться, если характерная ошибка выходного сигнала меньше $1/k.$ Даже при $k = 10$ получаем реализуемые требования к точности $(< 1/10)$ и богатые возможности (10! классов).

Оценка способности сети решить задачу

В данном разделе рассматриваются только сети, все элементы которых непрерывно зависят от своих аргументов. Предполагается, что все входные данные предобработаны так, чтобы все входные и выходные сигналы сети лежали в диапазоне приемлемых входных сигналов $[a, b].$

Нейронная сеть вычисляет некоторую *вектор*-функцию F от входных сигналов. Эта *функция* зависит от параметров сети. Обучение сети состоит в подборе такого набора параметров сети, чтобы величина

$$\sum_{i,p} [F_i(x^p) - f_i^p]^2$$

была минимальной (в идеале равна нулю), здесь $\{f_i\}$ - множество аппроксимируемых функций. Для того, чтобы *нейронная сеть* могла хорошо приблизить заданную таблично функцию f, необходимо, чтобы реализуемая сетью *функция* F при изменении входных сигналов с x^i на x^j могла изменить *значение* с f^i на f^j. Очевидно, что наиболее трудным для сети должно быть приближение функции в точках, в которых при малом изменении входных сигналов происходит большое изменение значения функции. Таким образом, наибольшую сложность будет представлять приближение функции f в точках, в которых достигает максимума *выражение* $\|f^i - f^j\|/||x^i - x^j\|$. Для аналитически заданных функций величина

$$sup_{x,y}(\|f(x) - f(y)\|/\|x - y\|)$$

называется константой Липшица. Исходя из этих соображений, можно дать следующее *определение* сложности задачи.

Сложность аппроксимации *таблично заданной функции* f, которая в точках x^i принимает значения f^i, задается выборочной оценкой *константы* Липшица, вычисляемой по формуле:

$$\Lambda_t = \max_{i \neq j}(\|f(x^i) - f(x^j)\|/\|x^i - x^j\|) \qquad 1)$$

Оценка (1) является оценкой *константы* Липшица аппроксимируемой функции снизу.

Константа Липшица сети вычисляется по следующей формуле:

$$\Lambda_n = sup_{x,y}(\|F(x) - F(y)\|/\|x - y\|)$$

Для того, чтобы оценить способность сети заданной конфигурации решить задачу, необходимо оценить константу Липшица сети и сравнить ее с выборочной оценкой (1). В случае $\Lambda_n < \Lambda_t$ *сеть* принципиально не способна решить *задачу аппроксимации* функции f. Однако из $\Lambda_n \geqslant \Lambda_t$ еще не следует утверждение о способности сети аппроксимировать функцию f!

Константа Липшица сигмоидальной сети

Рассмотрим слоистую сигмоидальную *сеть* (*сеть* с сигмоидальными нейронами) со следующими свойствами:

1. Число входных сигналов - n_0.
2. Число нейронов в i -м слое - n_i.
3. Каждый нейрон первого слоя получает все входные сигналы, а каждый нейрон любого другого слоя получает сигналы всех нейронов предыдущего слоя.
4. Все нейроны всех слоев имеют одинаковые функции активации $f(x) = 1/(1 + e^{-\beta x})$.
5. Все синаптические веса ограничены по модулю единицей.
6. В сети m слоев.

В этом случае оценка *константы* Липшица сети равна:

$$\Lambda_n \leqslant \beta^m (n_0 n_m)^{1/2} \prod_{i=1}^{m-1} n_i \qquad 2)$$

Формула (2) подтверждает экспериментально установленный факт, что, чем круче характеристическая *функция* нейрона (т.е. чем больше β), тем более сложные функции (функции с большей константой Липшица) может аппроксимировать *сеть* с такими нейронами.

Контрольные вопросы ПЗ4(ПК-4):

1. Реализация булевых функций нейронными сетями.
2. Выделение выпуклых областей.
3. Особенности планирования целенаправленных действий.
4. Оценки сложности задачи планирования.
5. Назначение экспертных систем.
6. Структура экспертных систем.
7. Решение задачи коммивояжера сетью Хопфилда.
8. Машина Больцмана.
9. Функция консенсуса.
10. Создание диалоговой панели.
11. Укажите, какая диаграмма рассматривает систему как совокупность предметов
12. Какие основные понятия используются при создании функциональной диаграммы IDEF0?
13. Какие функции реализуются в ИС организационного управления?
14. Укажите составляющие этапы проектирования ИС.
15. Что отражает модель ЖЦ ИС?
16. Какая модель ЖЦ наиболее объективно отражает реальный процесс создания сложных систем?

Практическое занятие №5. Многослойные сети сигмоидального типа. Алгоритм обратного распространения ошибки. Методы инициализации весов.

Аннотация: Рассматриваются: многослойный персептрон, алгоритм обратного распространения ошибки, подбор коэффициента обучения (одномерная минимизация), методы инициализации весов сети.

Алгоритм обратного распространения ошибки

Возьмем двухслойную *сеть* (рис. 1) (*входной слой* не рассматривается). Веса нейронов первого (скрытого) слоя пометим верхним индексом (1), а выходного слоя - верхним индексом (2). Выходные сигналы скрытого слоя обозначим $v_j, j = 1, 2, \ldots, K$, а выходного слоя - $y_j, j = 1, 2, \ldots, M.$ Будем считать, что *функция активации нейронов* задана в сигмоидальной униполярной или биполярной форме. Для упрощения описания будем использовать расширенное обозначение входного вектора сети в виде $x = [x_0, x_1, \ldots, x_N]^T$, где $x_0 = 1$ соответствует порогу. С вектором x связаны два выходных вектора сети: *вектор* фактических выходных сигналов $y = [y_0, y_1, \ldots, y_M]^T$ и *вектор* ожидаемых выходных сигналов $d = [d_0, d_1, \ldots, d_M]^T$.

Цель обучения состоит в подборе таких значений весов $w_{ij}^{(1)}$ и $w_{ij}^{(2)}$ для всех слоев сети, чтобы при заданном входном векторе x получить на выходе значения сигналов y_i, которые с требуемой точностью будут совпадать с ожидаемыми значениями d_i для $i = 1, 2, \ldots, M.$ Выходной сигнал i -го нейрона скрытого слоя описывается функцией

$$v_i = f(\sum_{j=0}^{N} w_{ij}^{(1)} x_j).$$

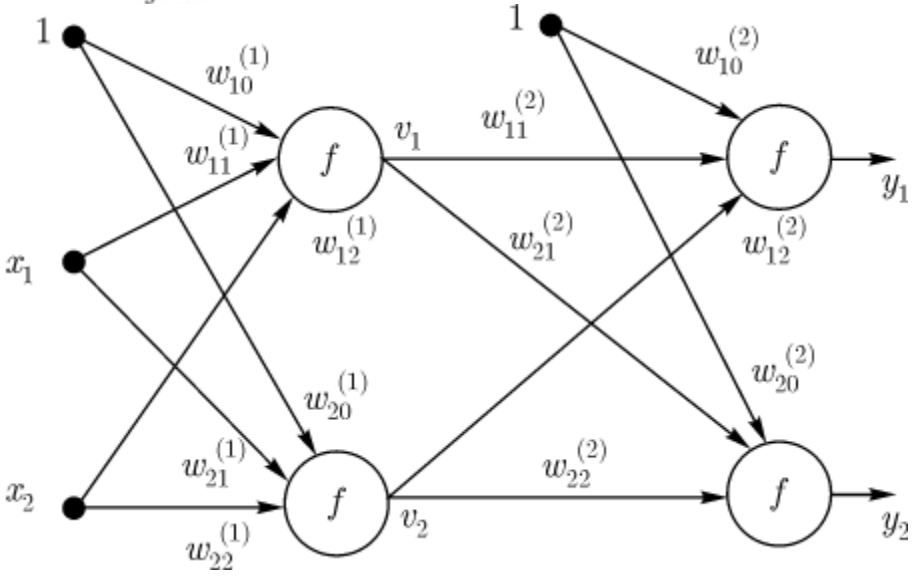


Рис. 1. Пример двухслойной нейронной сети

В выходном слое k -й *нейрон* вырабатывает выходной сигнал

$$y_k = f(\sum_{i=0}^{K} w_{ki}^{(2)} v_i) = f(\sum_{i=0}^{K} w_{ki}^{(2)} f(\sum_{j=0}^{N} w_{ij}^{(1)} x_j))$$

Из формулы следует, что на *значение* выходного сигнала влияют веса обоих слоев, тогда как сигналы, вырабатываемые в скрытом слое, не зависят от весов выходного слоя.

Основу алгоритма обратного распространения ошибки составляет целевая *функция*, формулируемая, как правило, в виде квадратичной суммы разностей между фактическими и ожидаемыми значениями выходных сигналов. Для обучающей выборки, состоящей из p примеров, *целевая функция* имеет вид

$$E(w) = [\sum_{j=1}^{p} \sum_{k=1}^{M} (y_k^{(j)} - d_k^{(j)})^2]/2$$

Минимизация целевой функции достигается уточнением вектора весов (обучением) по формуле

$$w(t+1) = w(t) + \Delta w,$$

где

$$\Delta w = \alpha s(w), \qquad (1)$$

α - коэффициент обучения, а $s(w)$ - направление в пространстве весов w. Выбор этого направления обычно основан на определении градиента целевой функции относительно весов всех слоев сети. Для весов выходного слоя задача имеет очевидное решение. Для других слоев используется *алгоритм обратного распространения ошибки*. Рассмотрим его на примере двухслойной сети. В этом случае при $p = 1$ *целевая функция* определяется выражением

$$E(w) = \sum_{k=1}^{M} [f(\sum_{i=0}^{K} w_{ki}^{(2)} v_i) - d_k]^2/2 = \sum_{k=1}^{M} [f(\sum_{i=0}^{K} w_{ki}^{(2)} f(\sum_{j=0}^{N} w_{ij}^{(1)} x_j)) - \qquad (2)$$

Компоненты градиента рассчитываются дифференцированием зависимости (2). В первую *очередь* определяются веса нейронов выходного слоя. Для выходных весов получаем:

$$\partial E / \partial w_{ij}^{(2)} = (y_i - d_i)[df(u_i^{(2)})/du_i^{(2)}]v_j,$$

где

$$u_i^{(2)} = \sum_{j=0}^{K} w_{ij}^{(2)} v_j$$

Если ввести обозначение

$$\partial_i^{(2)} = (y_i - d_i)[df(u_i^{(2)})/du_i^{(2)}],$$

то соответствующую компоненту градиента относительно весов выходного слоя можно представить в виде

$$\partial E/\partial w_{ij}^{(2)} = \partial_i^{(2)} v_j. \qquad 3)$$

Компоненты градиента относительно нейронов скрытого слоя определяются так же, но описываются более сложной зависимостью, следующей из существования функции, которая задана в виде

$$\partial E/\partial w_{ij}^{(1)} = \sum_{k=1}^{M} (y_k - d_k)[dy_k/dv_i][dv_i/dw_{ij}^{(1)}].$$

Отсюда получаем

$$\partial E/\partial w_{ij}^{(1)} = \sum_{k=1}^{M} (y_k - d_k)[df(u_k^{(2)})/du_k^{(2)}] w_{ki}^{(2)} [df(u_i^{(1)})/du_i^{(1)}] x_j,$$

Если ввести обозначение

$$\partial_i^{(1)} = \sum_{k=1}^{M} (y_k - d_k)[df(u_k^{(2)})/du_k^{(2)}] w_{ki}^{(2)} [df(u_i^{(1)})/du_i^{(1)}],$$

то получим *выражение*, определяющее компоненты градиента относительно весов нейронов входного слоя в виде

$$\partial E/w_{ij}^{(1)} = \partial_i^{(1)} x_j. \qquad 4)$$

В обоих случаях (формулы (3) и (4)) описания градиента имеют аналогичную структуру и представляются произведением двух сигналов: первый соответствует начальному узлу данной взвешенной связи, а второй — величине погрешности, перенесенной на узел, с которым эта *связь* установлена. *Определение* вектора градиента важно для последующего процесса уточнения весов. В классическом *алгоритме обратного распространения ошибки вектор* $s(w)$ в выражении (1) задает направление антиградиента (*метод наискорейшего спуска*), поэтому

$$\Delta w = -\alpha \nabla E(w).$$

В соответствии с алгоритмом обратного распространения ошибки в каждом цикле обучения выделяются следующие этапы:

1. *Анализ* нейронной сети в прямом направлении передачи информации при генерации входных сигналов, составляющих очередной *вектор* x. В результате такого анализа рассчитываются значения выходных сигналов нейронов скрытых слоев и выходного слоя, а также соответствующие *производные* $df(u_i^{(1)})/du_i^{(1)}, df(u_i^{(2)})/du_i^{(2)}, \ldots, df(u_i^{(m)})/du_i^{(m)}$ функций активации каждого слоя (m - количество слоев сети).

2. Создание *сети обратного распространения* ошибок путем изменения направлений передачи сигналов на обратные, замена функций активации их производными и подача на бывший *выход* (а в настоящий момент - вход) сети сигнала в виде разности между фактическим и ожидаемым значением. Для определенной таким образом сети необходимо рассчитать значения требуемых обратных разностей.

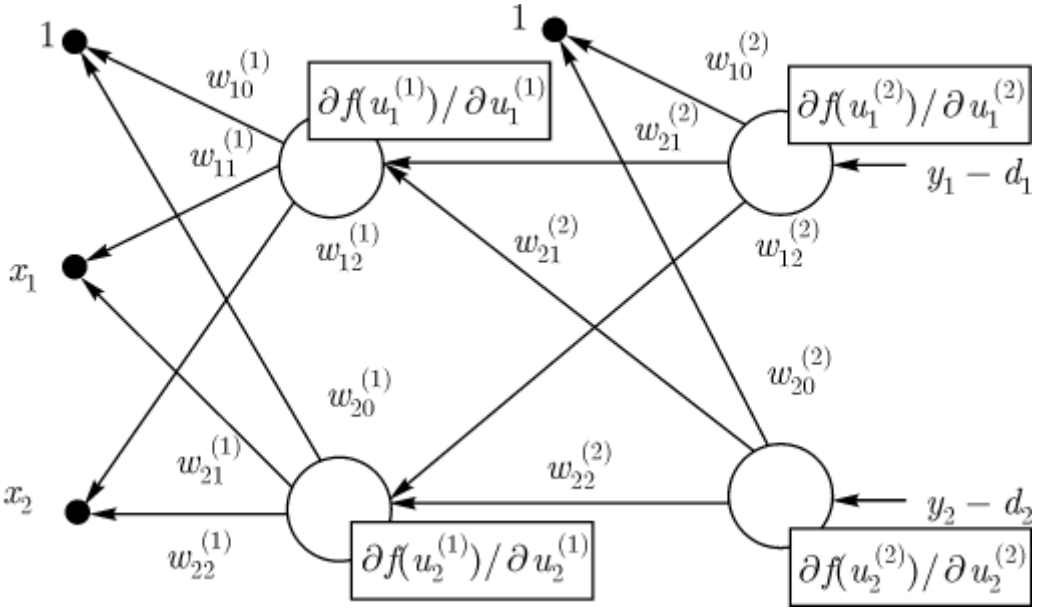


Рис. 2. Сеть обратного распространения ошибки

3. Уточнение весов (обучение сети) производится по предложенным выше формулам для оригинальной сети и для *сети обратного распространения* ошибки.

4. Описанный процесс следует повторить для всех обучающих примеров задачника, продолжая его вплоть до выполнения условия остановки алгоритма. Действие алгоритма завершается в момент, когда *норма* градиента упадет ниже априори заданного значения, характеризующего *точность* процесса обучения.

Руководствуясь рис. 2, можно легко определить все компоненты градиента целевой функции, т.е. все частные *производные* функции E по весам сети. Для этого, двигаясь от входов сети (бывших выходов), нужно перемножить все встречающиеся на пути величины (кроме весов $w_{ij}^{(k)}$, для которых рассчитывается *частная производная* $\partial E/w_{ij}^{(k)}$). Кроме того, там, где дуги сходятся к одной вершине, нужно выполнить *сложение* произведений, полученных на этих дугах.

Так, например, чтобы посчитать производную $\partial E/w_{12}^{(2)}$, нужно перемножить величины $y_2 - d_2, df(u_2^{(2)})/du_2^{(2)}, v_1$, а для вычисления производной $\partial E/w_{21}^{(1)}$ нужно посчитать произведения

$$\pi_1 = (y_1 - d_1) \times [\partial f(u_1^{(2)})/\partial u_1^{(2)}]w_{12}^{(2)}$$

и

$$\pi_2 = (y_2 - d_2)[\partial f(u_2^{(2)})/\partial u_2^{(2)}]w_{22}^{(2)}$$

и затем сложить эти произведения и результат умножить на $\partial f(u_2^{(2)})/\partial u_2^{(2)}$ и x_1.

Таким образом, получим

$$\partial E/\partial w_{12}^{(1)} = (\pi_1 + \pi_2)[\partial f(u_2^{(1)})/\partial u_2^{(1)}]x_1 =$$
$$= [(y_1 - d_1)[\partial f(u_1^{(2)})/\partial u_1^{(2)}]w_{12}^{(2)} + (y_2 - d_2)[\partial f(u_2^{(2)})/\partial u_2^{(2)})]w_{22}^{(2)}] \times$$
$$\times [\partial f(u_2^{(1)})/\partial u_2^{(1)}]x_1 = x_1 \sum_{k=1}^{2} (y_k - d_k)[\partial f(u_k^{(2)})/\partial u_k^{(2)}]w_{k2}^{(2)}[\partial f(u_2^{(1)})/\partial u_2^{(1)}].$$

Одномерная оптимизация

Все пошаговые методы оптимизации состоят из двух важнейших частей:

- выбора направления,
- выбора шага в данном направлении (подбор коэффициента обучения).

Методы одномерной оптимизации дают эффективный способ для выбора шага.

В простейшем случае коэффициент обучения фиксируется на весь период оптимизации. Этот способ практически используется только совместно с методом наискорейшего спуска. Величина подбирается раздельно для каждого слоя сети по формуле

$$\alpha \leqslant min(1/n_i),$$

где n_i обозначает количество входов i -го нейрона в слое.

Более эффективный метод основан на адаптивном подборе коэффициента α с учетом фактической динамики величины целевой функции. Стратегия изменения значения α определяется путем сравнения суммарной погрешности ε на t -й итерации с ее предыдущим значением, причем рассчитывается по формуле

$$\varepsilon = [\sum_{i=1}^{M} (y_i - d_i)^2]^{1/2}.$$

Для ускорения процесса обучения следует стремиться к непрерывному увеличению α при одновременном контроле прироста погрешности по сравнению с ее значением на предыдущем шаге. Незначительный рост погрешности считается допустимым.

Если погрешности на t -1-й и t -й итерациях обозначить соответственно ε_{t-1} и ε_t, а *коэффициенты* обучения на этих же итерациях — α_{t-1} и α_t, то *значение* ε_{t+1} следует рассчитывать по формуле

$$\alpha_{t+1} = \alpha_t \rho_d, \text{ если } \varepsilon_t > k_w \varepsilon_{t-1},$$
$$\alpha_{t+1} = \alpha_t \rho_i, \text{ если } \varepsilon_t \leqslant k_w \varepsilon_{t-1}.$$

где k_w - коэффициент допустимого прироста погрешности, ρ_d - коэффициент уменьшения α, ρ_i - коэффициент увеличения α.

Наиболее эффективный, хотя и наиболее сложный, метод подбора коэффициентов обучения связан с направленной минимизацией целевой функции в выбранном направлении s_t. Необходимо так подобрать *значение* α_t, чтобы новое решение $w_{t+1} = w_t + \alpha_t s_t$ соответствовало минимуму целевой функции в данном направлении s_t.

Поиск минимума основан на полиномиальной аппроксимации целевой функции. Выберем для аппроксимации *многочлен* второго порядка

$$E(w) = P_2(\alpha) = a_2\alpha^2 + a_1\alpha + a_0,$$

где a_2, a_1 и a_0 — *коэффициенты*, определяемые в цикле оптимизации. Для расчета этих коэффициентов используем три произвольные точки w_1, w_2, w_3, лежащие в направлении s_t, т.е.

$$w_i = w + \alpha_i s_t, \quad i = 1, 2, 3.$$

Соответствующие этим точкам значения целевой функции $E(w)$ обозначим как

$$P_2(\alpha_i) = E_i = E(w_i), \quad i = 1, 2, 3. \qquad 5)$$

Коэффициенты a_2, a_1 и a_0 рассчитываются в соответствии с решением системы уравнений (5). Для определения минимума многочлена $P_2(\alpha)$ его производная $dP_2/d\alpha = 2a_2\alpha + a_1$ приравнивается к нулю, что позволяет получить $\alpha_{min} = -a_1/2a_2$. После подстановки выражений для E_1, E_2, E_3 в формулу для α_{min} получаем

$$\alpha_{min} = \alpha_2 - [(\alpha_2 - \alpha_1)^2(E_2 - E_3) -$$
$$-(\alpha_2-\alpha_3)^2(E_2-E_1)]/2[(\alpha_2-\alpha_1)(E_2-E_3)-(\alpha_2-\alpha_3)(E_2-E_1)].$$

Методы инициализации весов

Обучение нейронных сетей представляет собой трудоемкий процесс, далеко не всегда дающий ожидаемые результаты. Проблемы возникают из-за нелинейных функций активации, образующих многочисленные *локальные минимумы*, к которым может сводиться процесс обучения. Применение методов глобальной оптимизации уменьшает *вероятность* остановки процесса обучения в точке локального минимума, однако платой за это становится резкое увеличение трудоемкости и длительности обучения. Для правильного подбора управляющих параметров требуется большой *опыт*.

На результаты обучения огромное влияние оказывает подбор начальных значений весов сети. Выбор начальных значений, достаточно близких к оптимальным, значительно ускоряет процесс обучения. К сожалению, не существует универсального метода подбора весов, который бы гарантировал нахождение наилучшей начальной точки для любой решаемой задачи.

Неправильный выбор диапазона случайных значений весов может вызвать слишком раннее насыщение нейронов, в результате которого, несмотря на продолжающееся обучение, среднеквадратичная *погрешность* будет оставаться практически постоянной. Это означало бы попадание в седловую зону целевой функции вследствие слишком больших начальных значений весов. При этом взвешенная сумма входных сигналов нейрона может иметь *значение*, соответствующее глубокому насыщению сигмоидальной функции активации, и поляризация насыщения будет обратна ожидаемой. *Значение* возвратного сигнала, генерируемое в методе обратного распространения, пропорционально величине производной от функции активации и в точке насыщения близко нулю. Поэтому изменения значений весов, выводящие *нейрон* из состояния насыщения, происходят очень медленно. Процесс обучения надолго застревает в седловой зоне. *Нейрон*, остающийся в состоянии насыщения, не участвует в преобразовании данных, сокращая таким образом эффективное количество нейронов в сети. В итоге процесс обучения чрезвычайно замедляется, поэтому состояние насыщения отдельных нейронов может длиться практически непрерывно вплоть до исчерпания лимита итераций.

Удаление стартовой точки активации нейронов от зоны насыщения достигается путем ограничения диапазона случайных значений. Почти все оценки нижней и верхней границ диапазона допустимых значений лежат в интервале (0,1). Хорошие результаты дает равномерное распределение весов, нормализованное для каждого нейрона по амплитуде $w_{in} = 2/(n_{in})^{1/2}$, где n_{in} означает количество входов нейрона. Веса порогов для нейронов скрытых слоев должны принимать случайные значения из интервала $(-1/w_{in}, 1/w_{in})$, а для выходных нейронов - нулевые значения.

Контрольные вопросы ПЗ5(ПК-4):

1. Алгоритм обратного распространения ошибки.
2. Методы инициализации весов
3. Этапы разработки экспертных систем.
4. Интерфейс с конечным пользователем.
5. Представление знаний в экспертных системах.
6. Уровни представления и уровни детальности.
7. Организация знаний в рабочей системе.
8. Организация знаний в базе данных.
9. Методы поиска решений в экспертных системах.
10. Как называется классификация, объединяющая в себе системы обработки транзакций?
11. Системы поддержки принятия решений.
12. Информационно-справочные системы.

13. Офисные информационные системы
14. Как называются связи, когда одна и та же запись может входить в отношения со многими другими записями?

Практическое занятие №6. Методы глобальной оптимизации. Элементы глобальной оптимизации. Алгоритмы имитации отжига. Генетические алгоритмы. Метод виртуальных частиц.

Аннотация: Рассматриваются: особенности задачи оптимизации, возникающей при обучении нейронных сетей; алгоритмы выбора направления минимизации: алгоритм наискорейшего спуска, партан-методы, одношаговый квазиньютоновский метод и сопряженные градиенты.

Универсальный путь обучения

Существует универсальный *путь* обучения нейронных сетей - *минимизация* оценки как *неявно заданной функции* параметров сети. При реализации этого подхода предполагается, что:

- задана обучающая выборка, состоящая из векторов входных сигналов x^p ;
- известны требования к соответствующим выходным сигналам y^p, зафиксированные в функции оценки $E(y^p)$;
- оценка E по всей выборке или какой-либо ее части строится известным способом по значениям $E(y^p)$.

После подготовки (создание обучающей выборки, выбор функции оценки, предобработка входных данных и т.п.), предшествующей обучению, имеем способ вычисления некоторой функции E , *минимизация* которой как функции параметров настроит *сеть* для правильной работы.

Особенности задачи оптимизации, возникающей при обучении нейронных сетей

Задачи оптимизации нейронных сетей имеют ряд специфических ограничений. Они связаны с огромной размерностью задачи обучения. Число параметров может достигать 10^8 и более. В простейших программных имитаторах на персональных компьютерах подбирается 10^3 - 10^4 параметров. Из-за высокой размерности возникают два требования к алгоритму:

1. Ограничение по памяти. Пусть n - число параметров. Если алгоритм требует затрат памяти порядка n^2, то он вряд ли применим для обучения. Желательно иметь алгоритмы, которые требуют затрат памяти $kn, k = const$.
2. Возможность параллельного вычисления наиболее трудоемких этапов алгоритма, и желательно нейронной сетью.
3. Обученный нейрокомпьютер должен с приемлемой точностью решать все тестовые задачи. Поэтому задача обучения становится многокритериальной задачей оптимизации: нужно найти точку общего минимума большого числа функций. Обучение нейрокомпьютера исходит из гипотезы о существовании этой точки.
4. Обученный нейрокомпьютер должен иметь возможность приобретать новые навыки без утраты старых. Возможно более слабое требование: новые навыки могут сопровождаться *потерей точности* в старых, но потеря не должна быть существенной. Это означает, что в достаточно большой окрестности найденной точки общего минимума оценок их значения незначительно отличаются от минимальных. Итак, имеем четыре специфических ограничения, выделяющих обучение нейрокомпьютера из общих задач оптимизации:

- астрономическое число параметров;

- необходимость высокого параллелизма при обучении;
- многокритериальность решаемых задач;
- необходимость найти достаточно широкую область, в которой значения всех минимизируемых функций близки к минимальным.

Учет ограничений при обучении

Для параметров сети возможны ограничения простейшего вида:

$$w_{i\min} \leqslant w_i \leqslant w_{i\max}.$$

Они вводятся из различных соображений: чтобы избежать слишком крутых или, наоборот, слишком пологих характеристик нейронов, чтобы предотвратить появления слишком больших коэффициентов усиления сигнала на синапсах и т.п.

Учесть ограничения можно, например, *методом штрафных функций* либо методом проекций:

- Использование *метода штрафных функций* означает, что в оценку E добавляется штрафы за выход параметров из области ограничений. В~градиент E вводятся производные штрафных функций.
- Проективный метод означает, что если в сети предлагается изменение параметров $w_i := W_i$ и W_i для некоторых i выходит за ограничения, то следует положить

$$w_i := \begin{cases} W_i, & \text{если} \quad w_{i\min} \leqslant W_i \leqslant w_{i\max} \\ w_{i\max}, & \text{если} \quad W_i > w_{i\max} \\ w_{i\min}, & \text{если} \quad W_i < w_{i\min} \end{cases}$$

Практика показывает, что проективный метод не приводит к затруднениям. Обращение со штрафными функциями менее успешно. Далее будем считать, что ограничения учтены одним из методов, и будем говорить об обучении как о безусловной минимизации.

Выбор направления минимизации

Пусть задано начальное *значение* вектора параметров w^0 и вычислена *функция* оценки $E = E(w^0)$. Процедура одномерной оптимизации дает приближенное положение минимума $e(x) = E(w^0 + xs)$ (вообще говоря, локального).

Наиболее очевидный выбор направления s для одномерной оптимизации - направление антиградиента E :

$$s = -\nabla E.$$

Выберем на каждом шаге это направление, затем проведем одномерную оптимизацию, потом снова вычислим *градиент* E и т.д. Это - метод наискорейшего спуска, который иногда работает хорошо. Но неиспользование информации о кривизне функции оценки (целевой функции) и резкое замедление минимизации в окрестности точки оптимального решения, когда *градиент* принимает очень малые значения, часто делают *алгоритм наискорейшего спуска* низкоэффективным.

Другой способ - случайный выбор направления s для одномерной оптимизации. Он требует большого числа шагов, но зато предельно прост — ему необходимо только прямое функционирование сети с вычислением оценки.

Партан-методы

Для исправления недостатков наискорейшего спуска разработаны итерационный и модифицированный партан-методы.

Итерационный партан-метод (k -партан) строится следующим образом. В начальной точке w^0 вычисляется *градиент* оценки E и делается шаг наискорейшего спуска - для этого используется одномерная *оптимизация*. Далее снова вычисляется *градиент* E и выполняется спуск (т.е. перемещение в направлении антиградиента), и описанный процесс повторяется k раз. После k шагов наискорейшего спуска получаем точку w^k и проводим одномерную оптимизацию из w^0 в направлении $s = w^k - w^0$ с начальным шагом $\alpha = 1$. После этого цикл повторяется.

Модифицированный партан-метод требует запоминания дополнительных параметров. Он строится следующим образом. Из w^0 делается два шага наискорейшего спуска. Получаем w^1 и w^2. Далее выполняем одномерную оптимизацию в направлении $w^2 - w^0$. Получаем w^3. Далее выполняется наискорейший спуск из w^3. Получаем w^4. Выполняем одномерную оптимизацию из w^2 в направлении $w^4 - w^2$. Получаем w^5 и~т.д. Таким образом, четные w^{2k} получаем наискорейшим спуском из w^{2k-1}, нечетные w^{2k+1} - одномерной оптимизацией из w^{2k-2} в направлении $s = w^{2k} - w^{2k-2}$ (начальный шаг $\alpha = 1$). Как показала практика, модифицированный партан-метод в задачах обучения работает лучше, чем k -партан.

Одношаговый квазиньютоновский метод и сопряженные градиенты

В тех случаях, когда является положительно определенной *матрица* D_2 вторых производных оценки E , наилучшим считается ньютоновское направление

$$s = -D_2^{-1}\nabla E.$$

С использованием этой формулы квадратичные формы минимизируются за один шаг, однако, применять эту формулу трудно по следующим причинам:

1. Время. Поиск всех вторых производных функции E и обращение матрицы D_2 требует больших вычислительных затрат.
2. Память. Для решения задач большой размерности N требуется хранить N^2 элементов матрицы D_2^{-1} — это слишком много.
3. Матрица D_2 не всегда является положительно определенной.

Для преодоления этих трудностей разработана масса методов. Идея квазиньютоновских методов с ограниченной памятью состоит в том, что поправка к направлению наискорейшего спуска отыскивается как результат действия матрицы малого ранга. Сама *матрица* не хранится, а

её действие на векторы строится с помощью скалярных произведений на несколько специально подобранных векторов.

Простейший и весьма эффективный метод основан на $BFGS$ формуле (Брайден-Флетчер-Гольдфард-Шанно) и использует результаты предыдущего шага. Обозначим:

s_k - направление спуска на k -шаге;

α_k - величина k шага (k -й шаг - сдвиг на $\alpha_k s_k$);

g_k - *градиент функции* оценки в начальной точке k -го шага;

$y_k = g_k - g_{k-1}$ - изменение градиента в результате k -го шага.

$BFGS$ - формула для направления спуска на $k+1$ -м шаге имеет вид:

$$s_{k+1} = -g_{k+1} + [(s_k, g_{k+1})y_k + (y_k, g_{k+1})s_k]/(y_k, s_k) -$$
$$-h_k s_k (s_k, g_{k+1})/(y_k, s_k) - s_k (y_k, y_k) \cdot (s_k, g_{k+1})/(y_k, s_k)^2,$$

где (x, y) - *скалярное произведение* векторов x и y .

Если одномерную оптимизацию в поиске шага проводить достаточно точно, то новый *градиент* g_{k+1} будет практически ортогонален предыдущему направлению спуска, т.е. $(s_k, g_{k+1}) = 0$. При этом формула для s_{k+1} упрощается:

$$s_{k+1} = -g_{k+1} + s_k (y_k, g_{k+1})/(y_k, s_k).$$

Это - формула метода сопряженных градиентов (МСГ), которому требуется достаточно аккуратная одномерная *оптимизация* при выборе шага.

В описанных методах предполагается, что начальное направление спуска $s_0 = -g_0$. После некоторой последовательности из k шагов целесообразно возвращаться к наискорейшему спуску - проводить *рестарт*. Он используется в тех случаях, когда очередное s_{k+1} - плохое направление спуска, т.е. движение вдоль него приводит к слишком маленькому шагу либо вообще не дает улучшения.

Контрольные вопросы ПЗ6(ПК-4):

1. Элементы глобальной оптимизации.
2. Алгоритмы имитации отжига.
3. Генетические алгоритмы.
4. Метод виртуальных частиц.
5. Методы работы со знаниями.
6. Основные определения.
7. Подготовительный и основной этапы.
8. Системы приобретения знаний от экспертов.
9. Формализация качественных знаний.

10. Приведите пример формализации качественных знаний.
11. Гибридный алгоритм обучения нечетких сетей.
12. Что такое генерализация?
13. Что такое полиморфизм?
14. В каком случае говорят о зависимости между классами?
15. Как соотносятся между собой понятия агрегации и композиции?

Практическое занятие №7. Радиальные нейронные сети.
Математические основы радиальных сетей. Радиальная нейронная сеть.

Элементы глобальной оптимизации

Все представленные ранее методы обучения нейронных сетей являются локальными. Они ведут к одному из локальных минимумов целевой функции, лежащему в окрестности точки начала обучения. Только в ситуации, когда *значение* глобального минимума известно, удается оценить, находится ли найденный локальный *минимум* в достаточной близости от искомого решения. Если локальное решение признается неудовлетворительным, следует повторить процесс обучения при других начальных значениях весов и с другими управляющими параметрами. Можно либо проигнорировать полученное решение и начать обучение при новых (как правило, случайных) значениях весов, либо изменить случайным образом найденное локальное решение (встряхивание весов) и продолжить обучение сети.

При случайном приращении весов переход в новую точку связан с определенной вероятностью того, что возобновление процесса обучения выведет *поиск* из "сферы притяжения" локального минимума.

При решении реальных задач в общем случае даже приблизительная оценка глобального минимума оказывается неизвестной. *По* этой причине возникает необходимость применения методов глобальной оптимизации. Рассмотрим три из разработанных подходов к глобальной оптимизации: метод имитации отжига, *генетические алгоритмы* и метод виртуальных частиц.

Алгоритмы имитации отжига

Метод имитации отжига основан на идее, заимствованной из статистической механики. Он отражает поведение расплавленного материала при отвердевании с применением процедуры отжига (управляемого охлаждения) при температуре, последовательно понижаемой до нуля.

В процессе медленного управляемого охлаждения, называемого отжигом, кристаллизация расплава сопровождается глобальным уменьшением его энергии, однако допускаются ситуации, в которых она может на какое-то время возрастать (в частности, при подогреве расплава для предотвращения слишком быстрого его остывания). Благодаря допустимости кратковременного повышения энергетического уровня, возможен *выход* из ловушек локальных минимумов энергии, которые возникают при реализации процесса. Только понижение температуры до абсолютного нуля делает невозможным какое-либо самостоятельное повышение энергетического уровня расплава.

Метод имитации отжига представляет собой алгоритмический аналог физического процесса управляемого охлаждения. Классический *алгоритм* имитации отжига можно описать следующим образом:

1. Запустить процесс из начальной точки w при заданной начальной температуре $T = T_{max}$.
2. Пока $T > 0$, повторить L раз следующие действия:
   - выбрать новое решение w' из окрестности w;
   - рассчитать изменение целевой функции $\Delta = E(w') - E(w)$;
   - если $\Delta \leqslant 0$, принять $w = w'$; в противном случае (при $\Delta > 0$) принять, что $w = w'$ с вероятностью $exp(-\Delta/T)$ путем генерации случайного

числа R из интервала $(0,1)$ с последующим сравнением его со значением $exp(-\Delta/T)$. Если $exp(-\Delta/T) > R$, принять новое решение $w = w'$; в противном случае проигнорировать его.

3. Уменьшить температуру $(T := rT)$ с использованием коэффициента r, выбираемого из интервала $(0,1)$, и вернуться к п. 2.

4. После снижения температуры до нуля провести обучение сети любым из *детерминированных методов* локальной оптимизации вплоть до достижения минимума целевой функции.

Наибольшего ускорения имитации отжига можно достичь путем замены случайных начальных значений весов w тщательно подобранными значениями с использованием любых доступных способов предварительной обработки исходных данных.

Метод имитации отжига оказывается особенно удачным для полимодальных комбинаторных проблем с очень большим количеством возможных решений, например, для *машины Больцмана*, в которой каждое *состояние системы* считается допустимым. При решении наиболее распространенных задач обучения *многослойных нейронных сетей* наилучшие результаты в общем случае достигаются применением стохастически управляемого метода повторных *рестартов* совместно с детерминированными алгоритмами локальной оптимизации.

Генетические алгоритмы

Генетические алгоритмы имитируют процессы наследования свойств живыми организмами и генерируют последовательности новых векторов w, содержащие оптимизированные переменные: $w = [w_1, w_2, \ldots, w_n]^T$. При этом выполняются *операции* трех видов: *селекция*, скрещивание и *мутация*.

На начальной стадии выполнения генетического алгоритма случайным образом инициализируется определенная популяция хромосом (векторов w). Размер популяции, как правило, пропорционален количеству оптимизируемых параметров. Слишком малая популяция хромосом приводит к замыканию в неглубоких локальных минимумах. Слишком большое их количество чрезмерно удлиняет вычислительную процедуру и также может не привести к точке глобального минимума.

Селекция хромосом для спаривания (необходимого для создания нового поколения) может основываться на разных принципах. Одним из наиболее распространенных считается принцип элитарности, в соответствии с которым наиболее приспособленные (в смысле целевой функции) *хромосомы* сохраняются, а наихудшие отбраковываются и заменяются вновь созданным потомством, полученным в результате скрещивания пар родителей.

Существует огромное множество методов скрещивания, начиная с полностью случайного. При взвешенно-случайном скрещивании учитывается *информация* о текущем значении целевой функции. Отбор может происходить *по* принципу рулетки; при этом *площадь* сегмента колеса рулетки, сопоставленного конкретной хромосоме, пропорциональна величине ее функции приспособленности $F(w) = -E(w)$, где $E(w)$ - ее *целевая функция*.

Процесс скрещивания основан на рассечении пары хромосом на две части с последующим обменом этих частей в хромосомах родителей (рис. 1). *Место* рассечения также выбирается случайным образом. Количество новых потомков равно количеству отбракованных в результате

селекции (размер популяции остается неизменным). Признается допустимым перенос в очередное поколение некоторых случайно выбранных хромосом вообще без скрещивания.

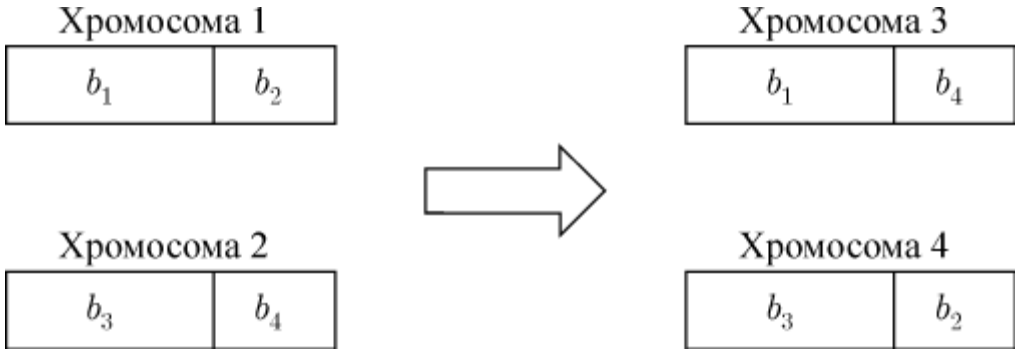


Рис. 1. Процесс скрещивания

Последняя генетическая операция - это *мутация*. При двоичном кодировании *мутация* состоит в инверсии случайно выбранных битов. При кодировании векторов десятичными числами *мутация* заключается в замене значения какого-либо элемента вектора другим случайно выбранным значением. *Мутация* обеспечивает защиту как от слишком быстрого завершения алгоритма (в случае выравнивания значений всех хромосом и целевой функции), так и от представления в какой-либо конкретной позиции всех хромосом одного и того же значения. Однако необходимо иметь в виду, что случайные *мутации* приводят к повреждению уже частично приспособленных векторов. Обычно *мутации* подвергается не более 1 - 5% *бит* всей популяции хромосом. Элемент, подвергаемый *мутации*, отбирается случайным образом.

Доказано, что каждое последующее поколение, сформированное селекцией, скрещиванием и мутацией, имеет статистически лучшие средние показатели приспособленности (меньшие значения целевой функции).

В качестве окончательного решения принимается наиболее приспособленная *хромосома*, имеющая *минимальное значение целевой функции*. Генетический процесс завершается либо в момент генерации удовлетворяющего нас решения, либо при выполнении максимально допустимого количества итераций. При реализации генетического процесса отслеживается, как правило, не только *минимальное значение целевой функции*, но и среднее *значение по* всей популяции хромосом, а также их вариации. Решение об остановке алгоритма может приниматься и в случае отсутствия прогресса минимизации, определяемого *по* изменениям названных характеристик.

Метод виртуальных частиц

Метод виртуальных (случайных) частиц может надстраиваться почти над любым методом оптимизации. Он создан для:

1. повышения устойчивости обученных сетей;
2. вывода сетей из возникающих при обучении локальных минимумов оценки.

Основная идея метода - использование случайных сдвигов аргумента и суммирование полученных значений функции для усреднения. Ожидается, что в результате уменьшится влияние рельефа минимизируемой функции на процесс минимизации и откроется более *прямой путь* к её глобальному минимуму.

Метод случайных частиц состоит в том, что к оптимизируемой точке (частице) добавляется несколько других, траектории которых получаются из траектории данной частицы сдвигом на

случайный *вектор*. Эти "виртуальные" частицы время от времени уничтожаются и рождаются новые. Спуск (*минимизация*) строится так, чтобы уменьшилось *значение* суммы значений оптимизируемой функции в указанных точках.

Рассмотрим один из вариантов алгоритма виртуальных частиц. Пусть требуется найти *минимум* функции $H_{pg}(w)$.Параметры сети разбиваются на группы структурно эквивалентных. Для каждой группы задается свой *интервал* случайных сдвигов. Определяется число виртуальных частиц n и генерируется $n-1$ случайных векторов $r_1, r_2, \ldots, r_{n-1}$. Их *координаты* независимо и равномерно распределены в заданных интервалах.

Начальное положение основной частицы - w^0. Начальное положение i -ой виртуальной частицы $w^0 + r_i, i = 1, \ldots, n-1$. Случайный *вектор* для n -й виртуальной частицы строится так:

$$r_n = (r_1 + r_2 + \ldots + r_{n-1}), \quad (1)$$

и её положение задается вектором $w^0 + r_n$. Всем частицам, кроме n -й, присваивается *вес* W, $0 < W < 1$, n -я получает *вес* $W_n = W/n^{1/2}$. Далее минимизируется *функция*

$$H_W(w) = H_{pg}(w) + W(H_{pg}(w + r_1) + \ldots + H_{pg}(w + r_{n-1})) + W_n H_{pg}(w + r_n).$$

Алгоритм локальной оптимизации может быть выбран любой - от наискорейшего спуска и партан-методов до *метода сопряженных градиентов*. Выбор r_n в виде (1) и $W_n = W/n^{1/2}$ определяется двумя обстоятельствами:

1. для каждой координаты вектора r_n дисперсия будет совпадать с дисперсией координат векторов $r_i, i = 1, \ldots, n-1$;
2. для квадратичных $H_{pg}(w)$ точки минимума $H_{pg}(w)$ и $H_W(w)$ совпадут.

В методе виртуальных частиц возникает важный вопрос: когда уничтожать имеющиеся виртуальные частицы и порождать новые?

Есть три варианта:

1. Функция $H_W(w)$ минимизируется до тех пор, пока скорость обучения не упадет ниже критической. После этого вновь производят случайные сдвиги частицы, и обучение продолжается.
2. Порождение новых частиц производится после каждого цикла базового алгоритма оптимизации - при *рестартах*. Например, после каждого шага метода наискорейшего спуска, после партан-шага итерационного партан-метода и т.п.
3. При каждом вычислении оценок и градиентов.

Первый способ наиболее консервативен. Он долго сохраняет все достоинства и недостатки предшествующего спуска, хотя направление движения может существенно измениться при порождении новых виртуальных частиц.

Третий способ вносит случайный процесс внутрь базового алгоритма, в результате возможны колебания даже при одномерной оптимизации. Его преимущество - экономия памяти.

Наиболее перспективным представляется второй способ. Он, с одной стороны, не разрушают базового алгоритма, а с другой - за счет многократного порождения виртуальных частиц позволяет приблизиться к глобальному множеству. Метод виртуальных частиц имеет все достоинства методов глобальной оптимизации, не использующих случайные возмущения, но лишен многих их недостатков.

Хорошие результаты обучения приносит *объединение* алгоритмов глобальной оптимизации с *детерминированными методами* локальной оптимизации. На первом этапе обучения сети применяется выбранный *алгоритм* глобальной оптимизации, а после достижения целевой функцией определенного уровня включается детерминированная *оптимизация* с использованием какого-либо локального алгоритма.

Четыре типа устойчивости

Навыки обучения нейрокомпьютера должны быть устойчивы к возмущению различных типов. Разработчики нейрокомпьютеров выделяют четыре типа устойчивости:

1. к случайным возмущениям входных сигналов;
2. к флуктуациям параметров сети;
3. к разрушению части элементов сети;
4. к обучению новым примерам.

В конкретных ситуациях необходимо доопределять возмущения, *по* отношению к которым нужно вырабатывать *устойчивость*. Например, при распознавании визуальных образцов можно выделить несколько разновидностей возмущений входного сигнала: прибавление случайного сигнала (шум фона), затенение части исходного изображения, искажение изображения некоторыми преобразованиями.

Для выработки устойчивости первых трех типов полезны генераторы случайных искажений. Для устойчивости 1-го типа *генератор* искажений производит возмущение входных сигналов и тем самым преобразует обучающей пример. Для устойчивости 2-го типа *генератор* искажений меняет случайным образом параметры сети в заданных пределах, а для устойчивости 3-го типа - удаляет случайно выбранную часть сети, состоящую из заданного количества элементов (нейронов, синапсов).

В существенной конкретизации нуждается четвертый тип устойчивости, т.к. трудно представить себе *устойчивость* к обучению любому новому примеру. Если принять гипотезу, что обучение новым примерам будет действовать на старые навыки так же, как случайный сдвиг параметров, то получается, что выработка устойчивости 2-го типа является средством для обучения устойчивости 4-го типа. Другое средство - выработка устойчивости к обучению отдельным примерам, уже входящим в задачник. Это свойство устойчивости 1-го типа состоит в том, что обучение до минимума оценки *по* любому (одному) из обучающих примеров не разрушает навыка решения остальных. Возмущение здесь состоит в изменении процесса обучения.

Для выработки устойчивости 1-го типа примеры предъявляются сети не все сразу, а *по* одному, и *сеть* учится каждому из них до предела. Для выработки важнейшей устойчивости 4-го типа такая периодически производимая "порча" процесса обучения может быть полезной. *Опыт* показывает, что обучение позволяет выработать *устойчивость* к весьма сильным возмущением. Так, в задачах распознавания визуальных образов уровень шума на выходе мог в несколько раз превосходить общую интенсивность сигнала, случайный сдвиг параметров - достигать 0.5-0.7 их предельного значения, разрушение - 30-50\% элементов. И, тем не менее, обученная *сеть* делает не более 10\% ошибок!

Контрольные вопросы ПЗ7(ПК-4):

1. Математические основы радиальных сетей.
2. Радиальная нейронная сеть.
3. Системы понимания естественного языка.
4. Предпосылки возникновения систем понимания естественного языка.
5. Понимание в диалоге.
6. Примеры системы обработки естественного языка.
7. Методы озвучивания речи.
8. Наиболее распространенные системы синтеза речи.
9. Речевой вывод информации.
10. Автоматический компьютерный синтез речи по тексту.
11. Методы синтеза речи.
12. Обобщенная функциональная структура синтезатора.
13. Как называется совокупность действий со строго определенными правилами выполнения
14. Как называется единая система данных, организованная по определенным правилам, которые предусматривают общие принципы описания, хранения и обработки данных
15. Как называется формализованная система сведений о некоторой предметной области, содержащая данные о свойствах объектов, закономерностях процессов и правила использования в задаваемых ситуациях этих данных для принятия новых решений
16. Как называется вся совокупность полезной информации и процедур, которые можно к ней применить, чтобы произвести новую информацию о предметной области

Практическое занятие №8. Рекуррентные сети на базе персептрона. Персептронная сеть с обратной связью. Рекуррентная сеть Эльмана. Сеть RTRN. Адаптивная резонансная теория (АРТ). Сеть АРТ-1. Архитектура и работа. Слой сравнения. Слой распознавания.

Аннотация: ***Рассматриваются математические основы радиальных сетей и методы их обучения. Производится сравнение радиальных и сигмоидальных нейронных сетей.***

Многослойные нейронные сети, представленные в предыдущих разделах, выполняют аппроксимацию функции нескольких переменных путем преобразования *множества* входных переменных $x \in R^N$ в множество выходных переменных $y \in R^M$. игмоидальная *функция* активации по своему характеру осуществляет аппроксимацию глобального типа. В результате ее *нейрон*, который был однажды "включен" (после превышения суммарным сигналом определенного порогового значения), остается в этом состоянии при любом значении сигнала, превышающем данный порог. Поэтому преобразование значения функции в произвольной точке пространства выполняется объединенными усилиями многих нейронов, что и объясняет название глобальная *аппроксимация*.

Другой способ отображения входного *множества* в выходное заключается в преобразовании путем адаптации нескольких одиночных аппроксимирующих функций к ожидаемым значениям, причем эта *адаптация* проводится только в ограниченной области многомерного пространства. При таком подходе *отображение* всего *множества* данных представляет собой сумму локальных преобразований. С учетом роли, которую играют *скрытые нейроны*, преобразования составляют множество *базисных функций* локального типа. Выполнение одиночных функций (при ненулевых значениях) регистрируется только в ограниченной области пространства данных - отсюда и название локальная *аппроксимация*.

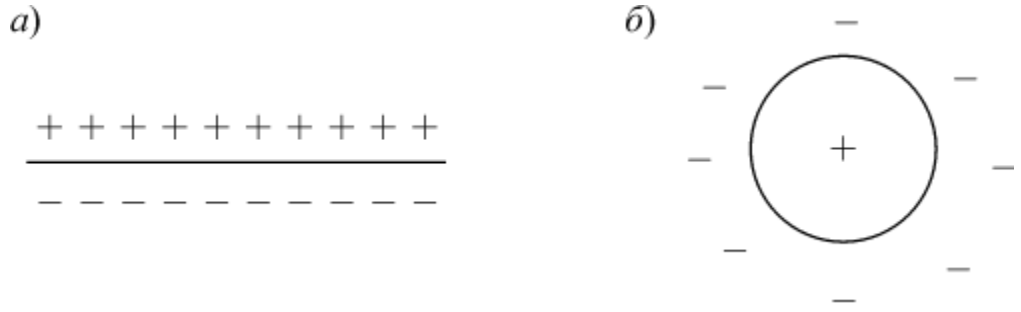


Рис. 1. Иллюстрация способов разделения пространства данных: а) сигмоидальным нейроном; б) радиальным нейроном

Особое семейство образуют сети с радиальной базисной функцией, в которых нейроны реализуют функции, радиально изменяющиеся вокруг выбранного центра и принимающие ненулевые значения только в окрестности этого центра. Подобные функции, определяемые в виде $\varphi(x) = \varphi(\|x - c\|)$, будем называть радиальными *базисными функциями*. В таких сетях роль нейрона заключается в отображении радиального пространства вокруг одиночной заданной точки (центра) либо вокруг группы таких точек, образующих *кластер*. *Суперпозиция* сигналов, поступающих от всех таких нейронов, которая выполняется выходным нейроном, позволяет получить *отображение* всего многомерного пространства.

Сети радиального типа представляют собой естественное *дополнение* сигмоидальных сетей. Сигмоидальный *нейрон* представляется в многомерном пространстве *гиперплоскостью*, разделяющей это *пространство* на две категории (два класса), в которых выполняется одно из

двух условий: либо $(w, x) > 0$, либо $(w, x) < 0$. Такой подход продемонстрирован на рис. 1а.

В свою *очередь*, радиальный *нейрон* представляет собой гиперсферу, которая осуществляет шаровое разделение пространства вокруг центральной точки (рис. 1б). Именно с этой точки зрения он является естественным дополнением сигмоидального нейрона, поскольку в случае круговой симметрии данных позволяет заметно уменьшить количество нейронов, необходимых для разделения различных классов. Поскольку нейроны могут выполнять различные функции, в радиальных сетях отсутствует необходимость использования большого количества скрытых слоев. Структура типичной радиальной сети включает *входной* слой, на который подаются сигналы, описываемые входным вектором x, скрытый слой с нейронами радиального типа и выходной слой, состоящий, как правило, из одного или нескольких линейных нейронов. *Функция выходного нейрона* сводится исключительно к взвешенному суммированию сигналов, генерируемых *скрытыми нейронами*.

Математические основы радиальных сетей

Математическую основу функционирования радиальных сетей составляет теорема Т. Ковера о распознаваемости образов, в соответствии с которой нелинейные проекции образов в некоторое многомерное *пространство* могут быть линейно разделены с большей вероятностью, чем при их проекции в *пространство* с меньшей размерностью.

Если *вектор* радиальных функций в N - мерном входном пространстве обозначить $\varphi(x)$, то это *пространство* является нелинейно φ - разделяемым на два пространственных класса X^+ и X^- тогда, когда существует такой *вектор* весов w, что

$$w^T\varphi(x) > 0, x \in X^+,$$
$$w^T\varphi(x) < 0, x \in X^-.$$

Граница между этими классами определяется уравнением $w^T\varphi(x) = 0$.

Доказано, что каждое множество образов, случайным образом размещенных в многомерном пространстве, является φ - разделяемым с вероятностью 1 при условии соответственно большой размерности этого пространства. На практике это означает, что применение достаточно большого количества скрытых нейронов, реализующих радиальные функции $\varphi(x)$, гарантирует решение задачи классификации при построении всего лишь двухслойной сети: скрытый слой должен реализовать *вектор* $\varphi(x)$, а *выходной слой* может состоять из единственного линейного нейрона, который выполняет суммирование выходных сигналов от скрытых нейронов с весовыми коэффициентами, заданными вектором w.

Простейшая *нейронная сеть* радиального типа функционирует по принципу многомерной интерполяции, состоящей в отображении p различных входных векторов $x_i, i = 1, 2, \ldots, p$ из входного N -мерного пространства во множество из p чисел $d_i, i = 1, 2, \ldots, p$. Для реализации этого процесса необходимо

использовать p скрытых нейронов радиального типа и задать такую функцию отображения $F(x)$, для которой выполняется условие интерполяции

$$F(x_i) = d_i.$$

Использование p скрытых нейронов, соединяемых связями с весами с выходными линейными нейронами, означает формирование выходных сигналов сети путем суммирования взвешенных значений соответствующих базисных функций. Рассмотрим радиальную *сеть* с одним выходом и p обучающими парами (x_i, d_i). Примем, что *координаты* каждого из p центров узлов сети определяются одним из векторов x_i, т.е. $c_i = x_i$. В этом случае взаимосвязь между входными и выходными сигналами сети может быть определена системой уравнений, линейных относительно весов, которая в матричной форме имеет вид:

$$\varphi \cdot w = d, \qquad 1)$$

где $\varphi_{ji} = (\|x_j - x_i\|)$ определяет радиальную функцию с центром в точке x_i с вынужденным вектором x_j, $w = [w_1, w_2, \ldots, w_p]^T$ и $d = [d_1, d_2, \ldots, d_p]^T$.

Доказано, что для ряда радиальных функций в случае

$$x_1 \neq x_2 \neq \ldots x_p$$

квадратная интерполяционная *матрица* φ является невырожденной и при этом неотрицательно определенной. Поэтому существует решение уравнения (1) в виде

$$W = \varphi^{-1} d, \qquad 2)$$

что позволяет получить *вектор* весов *выходного нейрона* сети.

Теоретическое решение проблемы, представленное выражением (2), не может считаться абсолютно истинным по причине серьезного ограничения общих свойств сети, вытекающих из сделанных вначале допущений. При очень большом количестве обучающих выборок и равном ему количестве радиальных функций проблема с математической точки зрения становится бесконечной (плохо структурированной), поскольку количество уравнений начинает превышать число степеней свободы физического процесса, моделируемого уравнением (1). Это означает, что результатом такого чрезмерного количества весовых коэффициентов станет *адаптация* модели к разного рода шумам или нерегулярностям, сопровождающим обучающие выборки. Как следствие, интерполирующая эти данные гиперповерхность не будет гладкой, а обобщающие возможности останутся очень слабыми.

Чтобы их усилить, следует уменьшить количество радиальных функций и получить из избыточного объема данных дополнительную информацию для *регуляризации* задачи и улучшения ее обусловленности.

Радиальная нейронная сеть

Использование в разложении p *базисных функций*, где p - это количество обучающих выборок, недопустимо также и с практической точки зрения, поскольку обычно количество этих выборок очень велико, и в результате вычислительная сложность обучающего алгоритма

становится чрезмерной. Решение системы уравнений (1) размерностью $p \times p$ при больших значениях p становится затруднительным. Так же, как и для многослойных сетей, необходимо редуцировать количество весов, что в этом случае сводится к уменьшению количества *базисных функций*. Поэтому отыскивается субоптимальное решение в пространстве меньшей размерности, которое с достаточной точностью аппроксимирует точное решение. Если ограничиться K *базисными функциями*, то аппроксимирующее решение можно представить в виде

$$F(x) = f_1 + f_2 +, \ldots, +f_K, \qquad 3)$$

где $f_i = w_i \varphi(\|x - c_i\|), K < p$, а $c_i, i = 1, 2, \ldots, K$ - множество центров, которые необходимо определить. В особом случае, если принять $K = p$, можно получить точное решение $c_i = x_i$.

Чаще всего в качестве радиальной функции применяется *функция* Гаусса. При размещении ее центра в точке c_i она может быть определена в сокращенной форме как

$$\varphi(x) = \varphi(\|x - c_i\|) = \exp(-\|x - c_i\|^2 / 2\sigma_i^2). \qquad 4)$$

В этом выражении σ_i - *параметр*, от значения которого зависит ширина функции.

Полученное решение, представляющее аппроксимирующую функцию в многомерном пространстве в виде взвешенной суммы локальных базисных радиальных функций (*выражение* (3)), может быть интерпретировано радиальной нейронной сетью, представленной на рис. 2 (для упрощения эта *сеть* имеет только один *выход*), в которой i определяется зависимостью (4). Это *сеть* с двухслойной структурой, в которой только скрытый слой выполняет нелинейное *отображение*, реализуемое нейронами с базисными радиальными функциями. Выходной *нейрон*, как правило, линеен, а его роль сводится к взвешенному суммированию сигналов, поступающих от нейронов скрытого слоя. *Вес* w_0, как и при использовании сигмоидальных функций, представляет поляризацию (порог), вводящую показатель постоянного смещения функции.

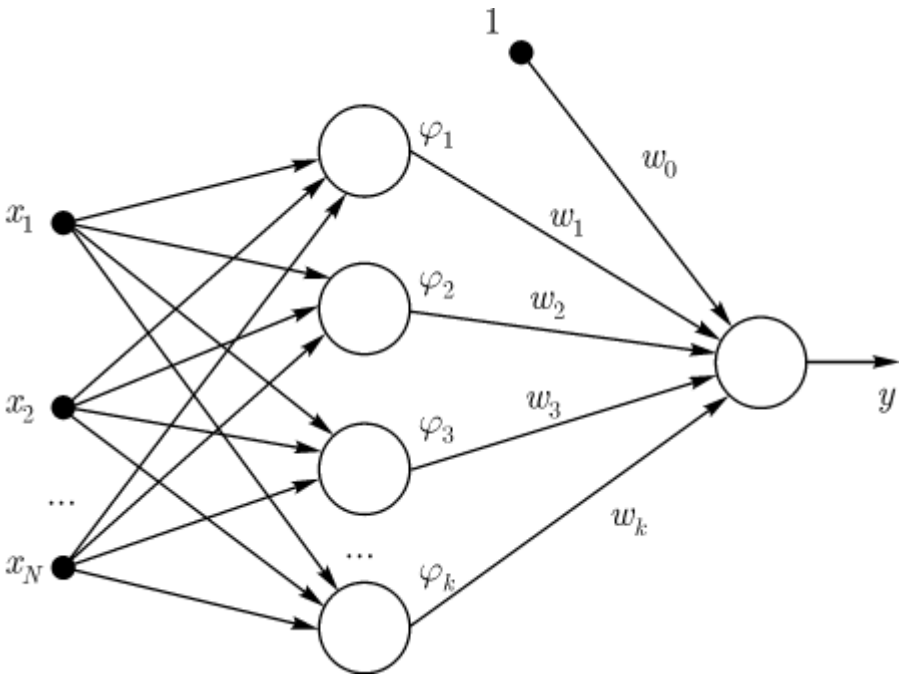


Рис. 2. Обобщенная структура радиальной сети

Полученная *архитектура* радиальных сетей имеет структуру, аналогичную многослойной структуре сигмоидальных сетей с одним скрытым слоем. Роль скрытых нейронов в ней играют базисные радиальные функции, отличающиеся своей формой от сигмоидальных функций. Несмотря на отмеченное сходство, сети этих типов принципиально отличаются друг от друга. Радиальная *сеть* имеет фиксированную структуру с одним скрытым слоем и линейными выходными нейронами, тогда как сигмоидальная *сеть* может содержать различное количество слоев, а выходные нейроны бывают как линейными, так и нелинейными. У используемых радиальных функций может быть весьма разнообразная структура. Нелинейная радиальная *функция* каждого скрытого нейрона имеет свои значения параметров c_i и σ_i, тогда как в сигмоидальной сети применяются, как правило, стандартные функции активации с одним и тем же для всех нейронов параметром β. Аргументом радиальной функции является эвклидово *расстояние* образца x от центра c_i, а в сигмоидальной сети - это скалярное *произведение* векторов $w^T x$.

Контрольные вопросы ПЗ8(ПК-4):

1. Персептронная сеть с обратной связью.
2. Рекуррентная сеть Эльмана.
3. Сеть R□RN.
4. Системы машинного зрения.
5. Основные принципы или целостность восприятия.
6. Распознавание символов.
7. Структурно-пятенный эталон.
8. Уроки машинного чтения от □□gnitive □echn□1□gies.
9. Распознавание рукописных текстов.
10. Состояние и тенденции развития искусственного интеллекта.
11. Успехи систем искусственного интеллекта и их причины
12. Какие из этапов создания ИС входят в стадию технического проектирования?
13. Какие из показателей отражаются в схеме маршрута движения документа?
14. Какие основные понятия используются при создании даиаграмм потоков данных?

15. Укажите основные компоненты диаграммы потоков данных
16. Сеть АРТ-1. Архитектура и работа.
17. Слой сравнения.
18. Слой распознавания.
19. Системы понимания естественного языка.
20. Предпосылки возникновения систем понимания естественного языка.
21. Понимание в диалоге.
22. Примеры системы обработки естественного языка.
23. Методы озвучивания речи.
24. Наиболее распространенные системы синтеза речи.
25. Речевой вывод информации.
26. Автоматический компьютерный синтез речи по тексту.
27. Методы синтеза речи. Обобщенная функциональная структура синтезатора.
28. Модуль лингвистической обработки.
29. Лингвистический анализ.
30. Формирование просодических характеристик.
31. □интезатор русской речи.