Лекция 8.

Тема: “**Сеть векторного квантования**”

9.1 Структура сети векторного квантования

Под векторным квантованием (ВК) подразумевается процесс преобразования некоторого вектора x из множества $A \in R^N$ в вектор w из множества $B \in R^M$, где M<N. Другими словами, множество векторов x размерности Nx1 разбивается на конечное число классов M, M<N, каждому из которых присваивается свой код (назначается опорный представитель) $w_i (i = \overline{1,M})$. Множество всех w_i образует *кодовое множество (кодовую книгу)* классификатора.

Векторное квантование осуществляется по методу "*ближайшего соседа*", причем под "ближайшим" понимается вектор, удовлетворяющий различным требованиям. Если в качестве такового выбирается вектор, находящийся от данного на минимальном евклидовом растоянии, т.е. на расстоянии минимизирующем стоимостной функционал

$$I(x, w_j) = \| x - w_j \|^2 = \sum_{i=1}^{N} (x_i - w_{ij})^2 , \qquad (9.1)$$

имеем классификатор называемый в литературе Voronoi-классификатор. Пример такой классификации, разбиения всего множества входных векторов на 7 классов (M=7), приведен на рис.9.1. Здесь крестиками обозначены входные векторы, а кружками – соответствующие опорные представители w_i $(i = \overline{1,7})$.

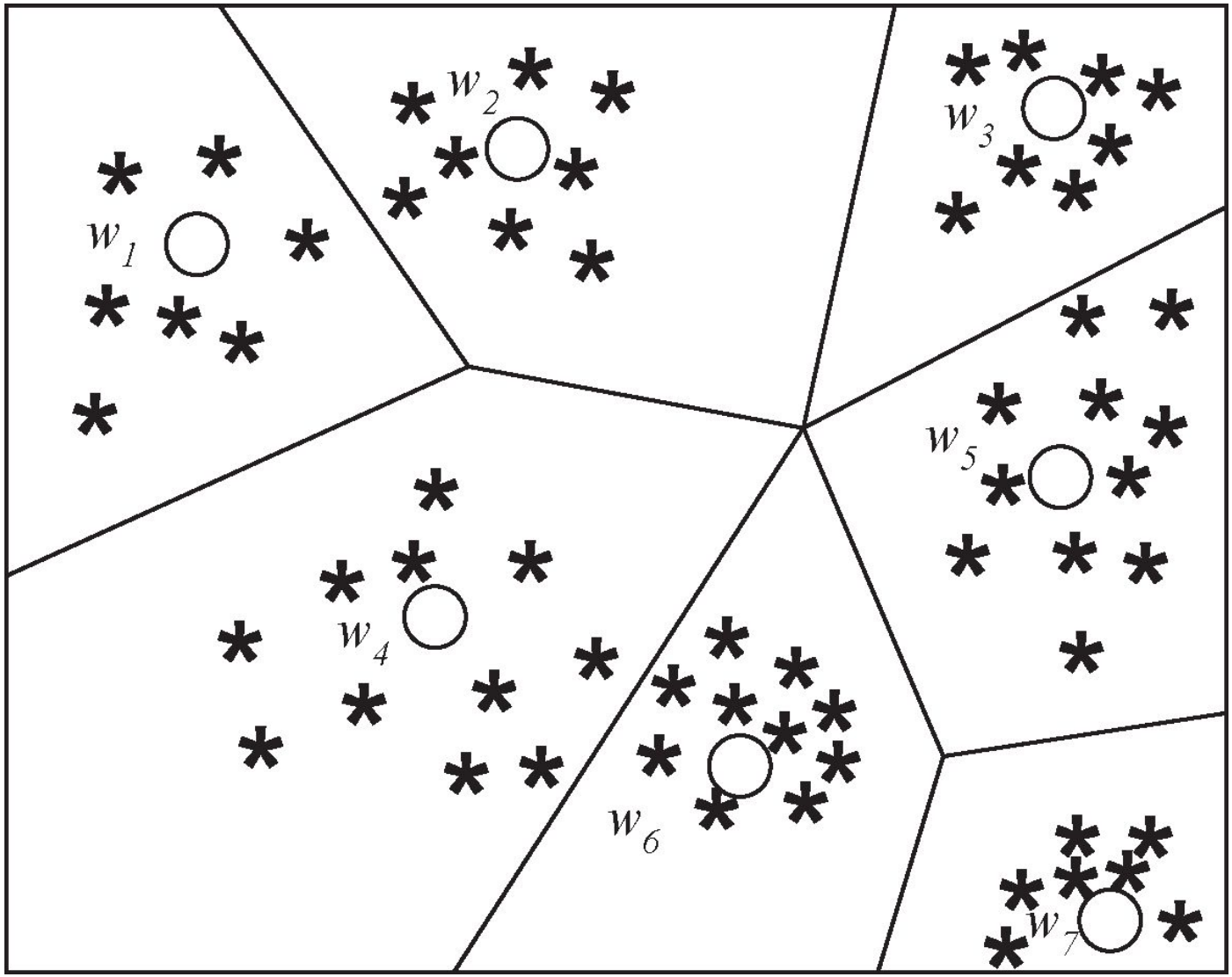
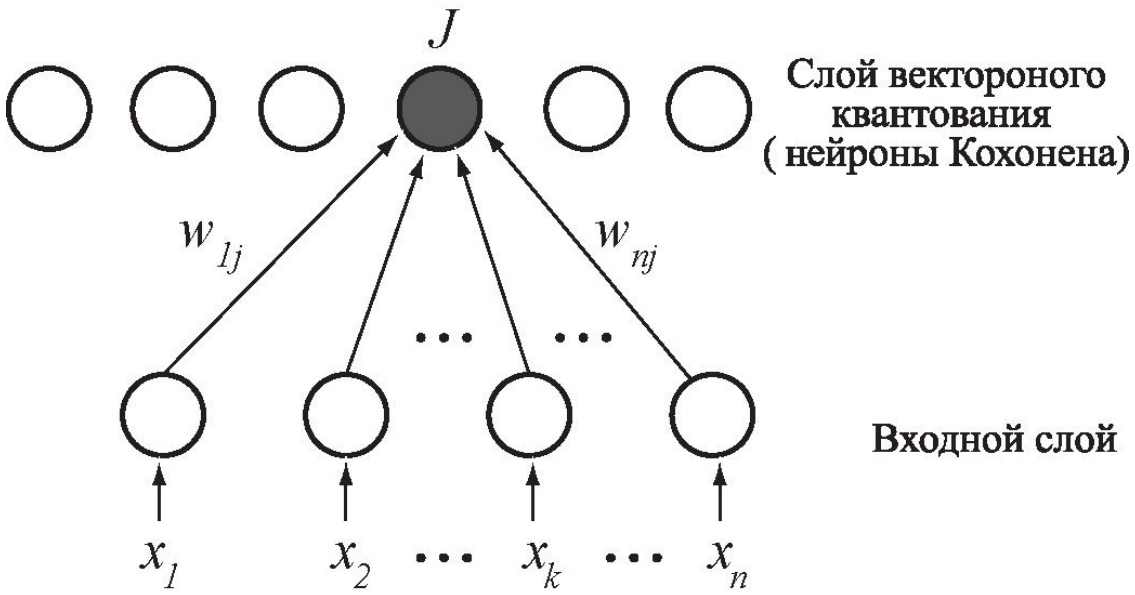


Рис.9.1 – Разбиение множества входных образов на классы

В сети ВК реализован принцип *"победитель получает все" (winner takes all)*, когда за право представления любого входного образа (вектора) соревнуются все нейроны выходного слоя. Нейрон, весовой вектор которого w_c наиболее близок входному образу, является победителем.

Конкурирующие нейроны Кохонена имеют как собственные обратные связи, так и латеральные, причем первые служат для усиления собственного сигнала нейрона, а вторые – для подавления влияния сигналов, поступающих от других нейронов.

Каждый нейрон входного слоя соединен со всеми нейронами слоя векторного квантования (VQ – слоя). Сила связи определяется соответствующим весом. На рис. 9.2 $w_j = (w_{1j}, w_{2j}, .. w_{Nj})^T$ – вектор весов j-го нейрона Кохонена.



Динамика сети описывается следующими дифференциальными уравнениями:

$$\frac{dy_j}{dt} = \sum_i w_{ij} x_i + \sum_{K \in S_j} v_{k_j} y_k - h(y_j), (j = \overline{1,M}), \qquad (9.2)$$

где x_i - входные сигналы $(i = \overline{1,N})$; W_{ij} - вес связи между i-м и j-м нейронами; v_{k_j} - вес связи между k-м нейроном выходного и j-м нейроном входного слоёв; s_j - множество нейронов, связанных с j-м нейроном; $h(y_j)$ - нелинейный член, учитывающий эффекты насыщения и смещения.

В уравнении (9.2) настраиваемыми являются весовые коэффициенты w_{ij}, веса же обратных связей v_{k_j} задаются и в процессе работы сети не изменяются.

9.2 Неконтролируемое обучение сети ВК

Существует несколько подходов к обучению сети ВК. В наиболее простом, первоначальном, варианте при обучении происходило изменение весов только нейрона – победителя в соответствии с правилом

$$\frac{dw_{ij}}{dt} = \begin{cases} \alpha(x_i - w_{ij}), \text{если } y_j = 1; \\ 0 \qquad\quad \text{в противном случае.} \end{cases} \tag{9.3}$$

Здесь α – коэффициент обучения.

Если сумма весов для каждого нейрона является величиной постоянной и $\| w_i \|^2 \approx 1$, а входные векторы нормализованы $\| x \|^2 = 1$, то, как показал Кохонен, время определения нейрона – победителя может быть сокращено. При этом нейрон – победитель с определяется путём минимизации евклидовой нормы

$$\| x - w_c \|^2 = \min_j (\| x - w_j \|^2), \tag{9.4}$$

или

$$c = \arg\min_j (\| x - w_j \|^2). \tag{9.5}$$

При выполнении указанных выше условий относительно норм входных и весовых векторов

$$\| x - w_j \|^2 = 2 - x^T w_j, \tag{9.6}$$

величина

$$y_j = x^T w_j = \sum_{i=1}^{N} x_i w_{ij} \tag{9.7}$$

может использоваться в качестве критерия подобия образов x и w_j.

В дискретном случае при предъявлении на каждом такте обучения нового входного образа происходит коррекция весов нейрона – победителя по формуле

$$w_c(k+1) = w_c(k) + \alpha(x(k) - w_c(k)), \qquad (9.8)$$

$$w_i(k+1) = w_i(k) \text{ при } i \neq c.$$

Как и в (9.3), коэффициент обучения α может быть выбран постоянным или уменьшающимся в процессе обучения. Как следует из (9.8), вектор весов нейрона – победителя w_c смещается в направлении вектора входного образа x. Процесс коррекции весов этого нейрона представлен на рис. 9.3

При $\alpha(k) \neq 1$ вектор $w(k)$ корректируется на величину $\alpha(k)(x(k) - w(k))$ и занимает новое положение $w(k+1)$. Видно, что обучение состоит во вращении весового вектора в направлении вектора входов $x(k)$ без существенного изменения его длины.

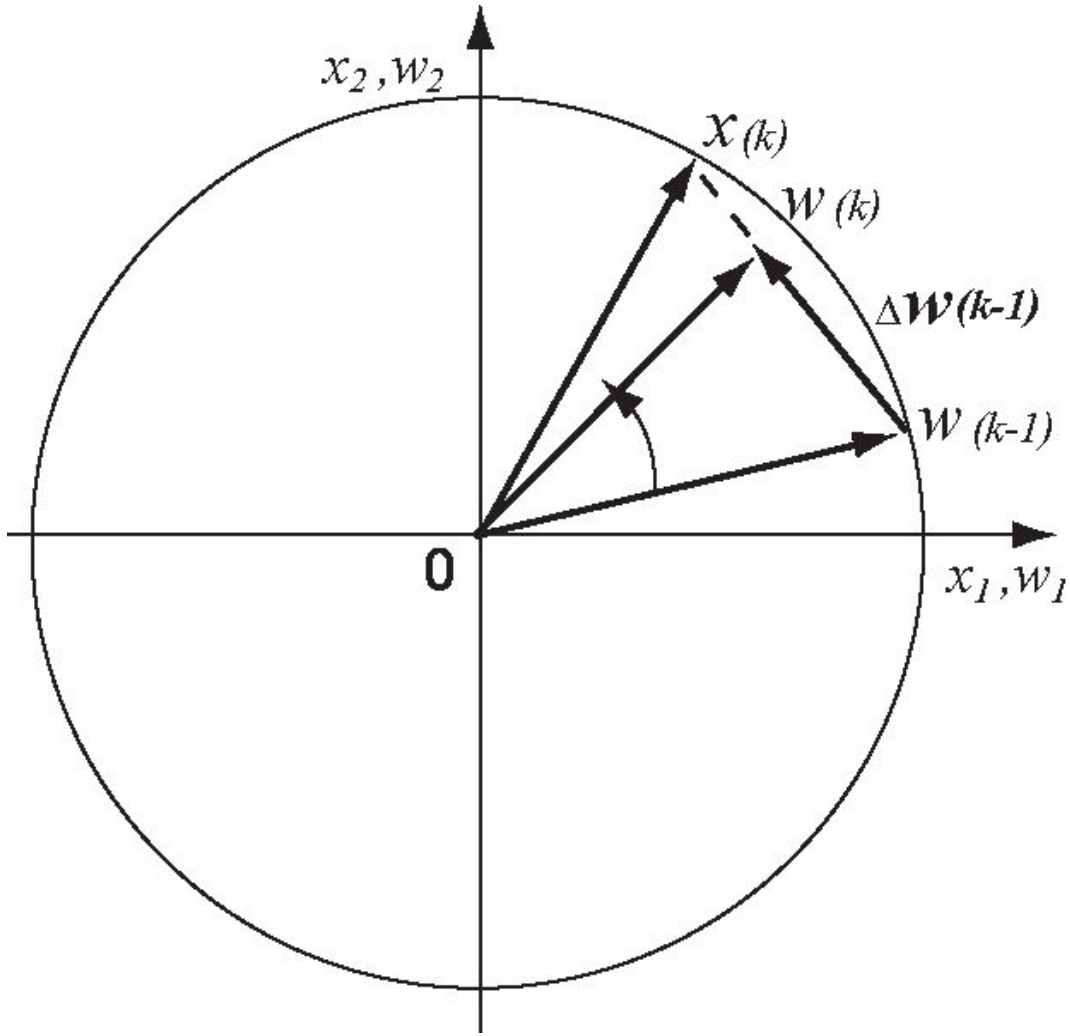


Рис.9.3 – Коррекция весовых коэффициентов нейрона – победителя

Основным недостатком данного вида обучения является то, что если начальные распределения векторов весов и входных образов не являются примерно одинаковыми, то может возникнуть ситуация, когда некоторые из нейронов никогда не станут победителями, т.е. их векторы весов не будут изменяться. Для исключения подобной ситуации в алгоритм обучения вводят некоторый механизм "памяти", который штрафует часто корректируемые веса нейронов – победителей. В качестве такого алгоритма обучения с "памятью" может быть использован, например, следующий [ ]:

$$w_i(k+1) = w_i(k) + \alpha(x(k) - w_i(k))z_i, \qquad (9.9)$$

где

$$z_i = \begin{cases} 1, \text{если} \| x - w_i \|^2 - B_i < \| x - w_j \|^2 \text{ для всех } j \neq i, \\ 0 \quad \text{в противном случае;} \end{cases}$$

$$B_i = c\left(\frac{1}{n} - p_i\right) \text{ - коэффициент штрафа;}$$

p_i - временной интервал, на котором i-й нейрон является победителем

$$p_i(k+1) = p_i(k) + b(y_i(k) - p_i(k));$$

$b \in (0.1]$ – постоянный параметр;

c – постоянный параметр, влияющий на величину штрафа.

9.3 Контролируемое обучение сети ВК

Основным отличием *контролируемого обучения* сети ВК (*Learning vector quantization, LVQ*) от рассмотренного выше является использование для каждого входного образа x желаемого соответствующего выходного сигнала. Данный вид обучения реализуется различными способами.

9.3.1. LVQ1

Как и в описанном выше методе, победителем в сети является тот нейрон, вектор весов которого W_c наиболее близок входному образу x, т.е. нейрон *c* определяется как

$$c = \arg\min_j(\| x - w_j \|).$$

Значения всех весов w_j, минимизирующие ошибку классификации, вычисляются LVQ1 методом асимптотически. При этом коррекция весов происходит по правилу

$$w_c(k+1) = \begin{cases} w_c(k) + \alpha(k)[x(k) - w_c(k)], & \text{если } w_c \text{ и } x \text{ относятся} \\ & \text{к одному классу;} \\ w_c(t) - \alpha(k)[x(t) - w_c(k)] & \text{в противном случае;} \end{cases} \qquad (9.10)$$

$$w_c(k+1) = w_j(k) \text{ для всех } j \neq c,$$

где $\alpha(k) \in (0,1]$. Параметр $\alpha(t)$ может оставаться постоянным или монотонно уменьшаться.

Таким образом, вектор весов нейрона – победителя w_c, который ближе всего расположен к предъявляемому входному вектору, смещается в направлении последнего, если входной вектор относится к одному с ним классу, и удаляется от него в противном случае. Веса остальных нейронов не изменяются. Изменение весов нейрона Кохонена показано на рис.9.4. На рис.9.4 а) приведен случай, когда векторы x(k) и $w_c(k)$ относятся к одному классу, на рис. 9.4 б) – к разным.

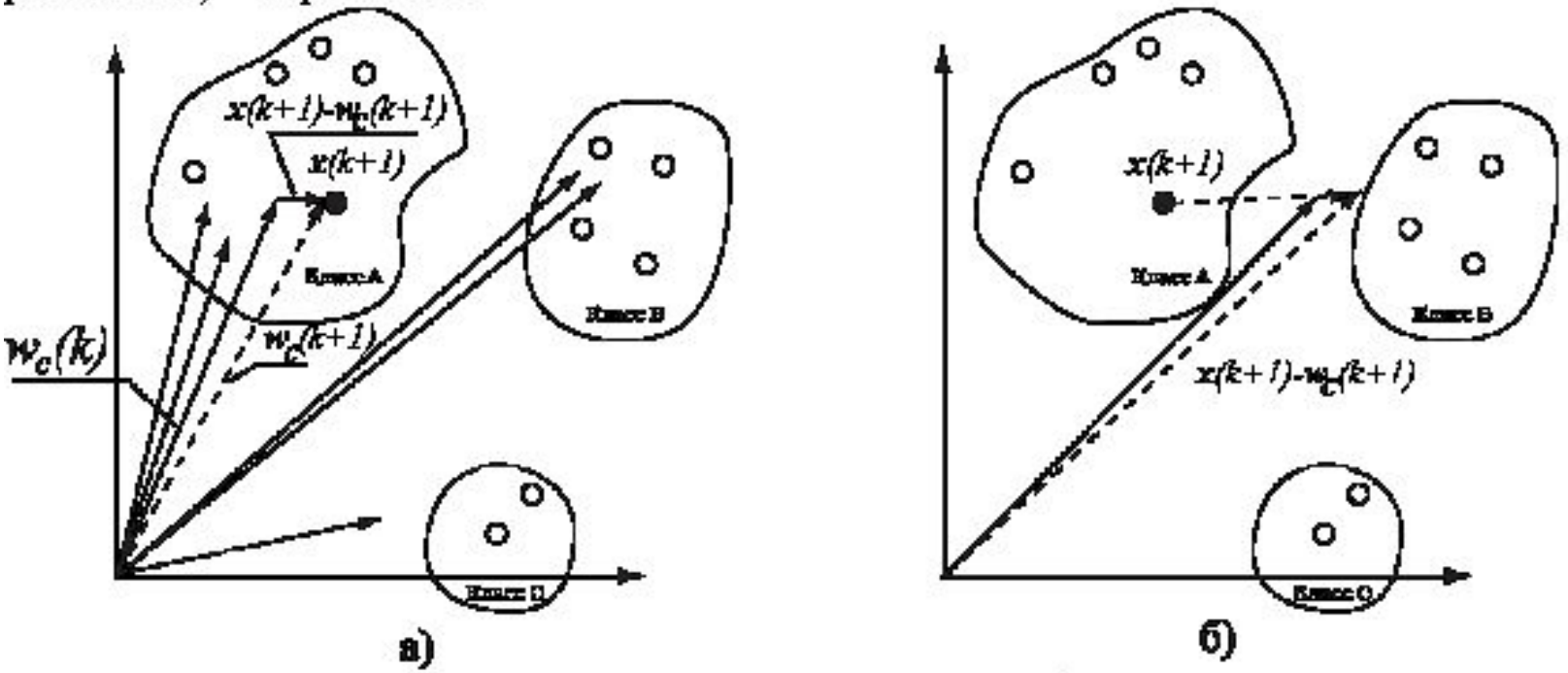


Рис. 9.4 – Изменение весов нейронов Кохонена.

9.3.2 LVQ 2.1

В LVQ 2.1 классификация происходит аналогично LVQ1, т.е. путем минимизации выбранной метрики. Отличие состоит в способе коррекции весов.

В методе LVQ2.1 определяется вектор w_i и следующий w_j и осуществляется адаптивная коррекция весов, если

- векторы w_i и w_j относятся к разным классам;
- вектор x принадлежит либо тому классу, к которому относится w_i, либо тому, которому принадлежит w_j;
- вектор x находится в некотором "окне" относительно перпендикуляра между этими двумя классами.

Если обозначить евклидово расстояние между векторами x и w_i как d_i, а между x и w_j - как d_j, то вектор x находится в "окне" относительной ширины V, если

$$\min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > s, \text{ где } s = \frac{1 - V}{1 + V}.$$

Кохонен предложил выбирать $V = 0{,}2 \div 0{,}3$. Следует отметить, что это "окно" – не прямоугольное.

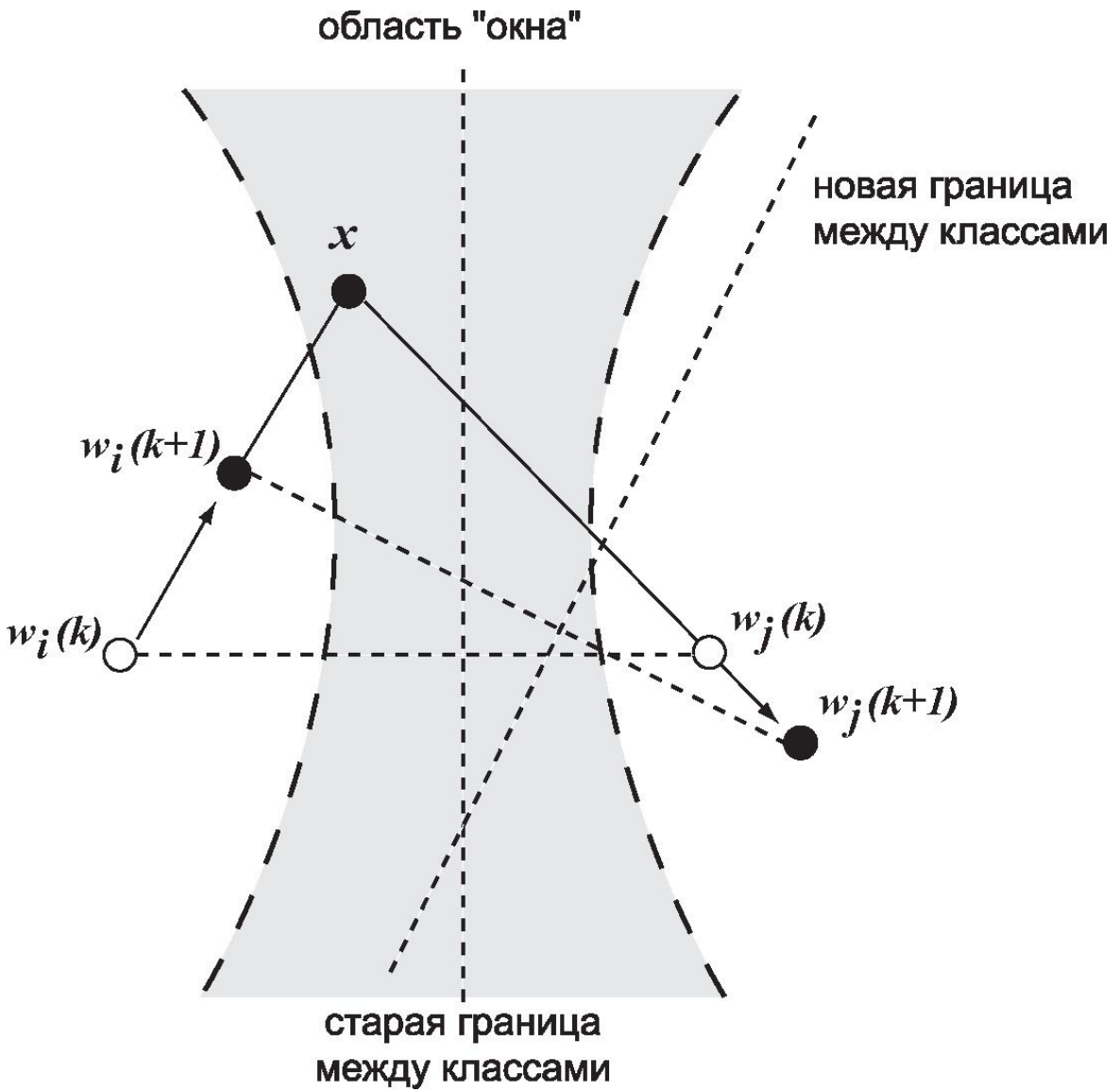


Рис.9.5 – Принцип работы LVQ 2.1.

Алгоритм коррекции весов имеет следующий вид:

$$w_i(k+1) = w_i(k) + \alpha(k)[x(k) - w_i(k)];$$
$$w_j(k+1) = w_j(k) - \alpha(k)[x(k) - w_j(k)]. \qquad (9.11)$$

Как видно из рисунка, в результате работы алгоритма (9.11) изменяется положение границы между классами.

9.3.3 LVQ 3

Данный метод является модификацией LVQ 2.1, позволяющей изменить не только веса векторов w_i и w_j, но и границу между классами, если оба вектора принадлежат тому же классу, что и x.

Если w_i и w_j относятся к разным классам, а w_i - к тому же классу, что и x, то коррекция весовых векторов происходит как в LVQ 2.1, т.е. по правилу (9.11).

Если же w_i и w_j относятся к одному классу, то адаптация весов осуществляется по алгоритму

$$w_i(k+1) = w_i(k) + e\alpha(k)[x(k) - w_i(k)];$$
$$w_j(k+1) = w_j(k) + e\alpha(k)[x(k) - w_j(k)], \qquad (9.12)$$

где $e \in [0{,}1 \div 0{,}5]$ некоторый заданный параметр (оптимальное значение e зависит от ширины “окна” и для узкого “окна” лучше выбирать малое e).

9.3.4 OLVQ 1

Недостаток LVQ 1 – медленная скорость сходимости, устранен в оптимизированном LVQ1, OLVQ1 (*Optimized Learning Vector Quantization*), в соответствии с которым адаптация вектора весов сети происходит по правилу

$$w_c(k+1) = \begin{cases} w_c(k) + \alpha_c(k)[x(k) - w_c(k)], & \text{если } w_C \text{ и } x \text{ принадлежат} \\ & \text{одному классу;} \\ w_c(k) - \alpha_c(k)[x(k) - w_c(k)] & \text{в противном случае;} \end{cases} \quad (9.13)$$

$$w_j(k+1) = w_j(k) \qquad \text{для всех } j \neq c.$$

Оптимизация LVQ1 осуществляется путем выбора оптимальных значений коэффициента обучения $\alpha_c^{опт}(k)$. Определение оптимального $\alpha_c^{опт}(k)$ происходит следующим образом.

Запишем (9.13) в виде

$$w_c(k+1) = w_c(k) + S(k)\alpha_c(k)\left[x(k) - w_c(k)\right], \quad (9.14)$$

где

$$S(k) = \begin{cases} +1, & \text{если } w \text{ и } x \text{ принадлежат одному классу;} \\ -1, & \text{если } w \text{ и } x \text{ принадлежат разным классам.} \end{cases}$$

Следует отметить, что $x(k)$ несет информацию о вновь поступившем образе, а информация о всех предыдущих входных сигналах содержится в $w_c(k)$, поскольку

$$\begin{aligned} w_c(k+1) &= \left[1 - S(k)\alpha_c(k)\right]w_c(k) + S(k)\alpha_c(k)x(k) = \\ &= \left[1 - S(k)\alpha_c(k-1)\right]\left[1 - S(k)\alpha_c(k)\right]w_c(k-1) + \\ &+ S(k)\alpha_c(k)\left[1 - S(k)\alpha_c(k-1)\right]x(k-1) + S(k)\alpha_c(k)x(k). \end{aligned} \quad (9.15)$$

Величины коэффициентов обучения $\alpha(k), \alpha(k-1), \ldots$ отражают влияние соответствующих входных образов на корректируемый вектор w_c. Из (9.15) видно, что влияние на $w_c(k+1)$ образа x(k) определяется весом $\alpha_c(k)$, образа x(k-1) – весом $\alpha_c(k-1)\left[1 - S(k)\alpha_c(k)\right]$ и т.д. Распределение весовых векторов w является статистически оптимальным, если при $k \to \infty$ влияние всех обучающих образов на изменение вектора w_c одинаково. Формально это можно записать так

$$\alpha_c(k) = \left[1 - S(k)\alpha_c(k)\right]\alpha_c(k-1). \qquad (9.16)$$

Так как на двух соседних тактах суммирующее влияние соответствующих входных векторов на весовой вектор одинаково и это справедливо для всех моментов времени k, то по индукции можно показать, что при $k \to \infty$ влияние всех входных векторов на w одинаково. Из уравнения (9.16) имеем:

$$\alpha_c(k) = \frac{\alpha_c(k-1)}{1 + \alpha_c(k-1)S(k)}. \qquad (9.17)$$

Хотя $\alpha_c(k)$ принципиально может возрастать вследствие того, что на некоторых тактах S(k)= -1, значение этого коэффициента не должно быть больше единицы.

Обычно в качестве начального значения выбирают $\alpha_c(0) = 0{,}3$.

Так как данный метод является наиболее быстрым, Кохонен рекомендует начинать обучение сети именно с его помощью, а затем переходить на какой-либо из LVQ.

Пример 9.1. Рассмотрим работу метода LVQ1 на примере обучения сети, состоящей из двух нейронов и классифицирующей восемь следующих векторов, которые относятся к двум различным классам:

Вектор	Класс
$x_1=[0\ 0\ 1\ 1]^T$	1
$x_2=[1\ 1\ 0\ 1]^T$	2
$x_3=[0\ 1\ 1\ 1]^T$	1
$x_4=[1\ 0\ 0\ 1]^T$	1
$x_5=[1\ 1\ 0\ 0]^T$	2
$x_6=[0\ 1\ 1\ 0]^T$	2
$x_7=[1\ 0\ 1\ 1]^T$	1
$x_8=[1\ 1\ 1\ 1]^T$	2

В качестве начальных значений векторов весов w_1 и w_2 ассоцирующих с классами 1 и 2, примем соответственно векторы x_1 и x_2. Обучение сети будем осуществлять по алгоритму (9.10) с α=0.1.

Рассмотрим первый цикл обучения.

При поступлении вектора x_3 (с учетом того, что $w_1(0)= x_1$ и $w_2(0)= x_2$) имеем

$$\|x_3 - w_1\|^2 = 1 = \min,$$

$$\|x_3 - w_2\|^2 = 2.$$

Так как победителем оказался первый нейрон, то настраиваем его веса, принимая во внимание, что x_3 относится к классу 1, т.е.

$$w_1(1) = w_1(0) + 0.1[x_3 - w_1(0)] = (0 \quad 0 \quad 1 \quad 1)^T + 0.1[(0 \quad 1 \quad 1 \quad 1)^T - (0 \quad 0 \quad 1 \quad 1)^T] =$$

$$= (0 \quad 0.1 \quad 1 \quad 1)^T.$$

При поступлении вектора x_4 имеем

$$\|x_4 - w_1(1)\|^2 = 2.01,$$

$$\|x_4 - w_2(0)\|^2 = 1 = \min.$$

Поэтому корректируем веса $w_2(0)$ с учетом того, что $w_2(0)$ и x_4 относятся к разным классам

$$w_2(1) = w_2(0) - 0.1[x_4 - w_2(0)] = (1 \quad 1 \quad 0 \quad 1)^T - 0.1[(1 \quad 0 \quad 0 \quad 1)^T - (1 \quad 1 \quad 0 \quad 1)^T] =$$

$$= (1 \quad 1.1 \quad 0 \quad 1)^T.$$

Аналогично получаем

для x_5:

$$\|x_5 - w_1(1)\|^2 = 3.01,$$

$$\|x_5 - w_2(1)\|^2 = 1.01 = \min$$

$$w_2(2) = w_2(1) + 0.1[x_5 - w_2(1)] = (1 \quad 1.1 \quad 0 \quad 1)^T + 0.1[(1 \quad 1 \quad 0 \quad 0)^T -$$

$$- (1 \quad 1.1 \quad 0 \quad 1)^T] = (1 \quad 1.09 \quad 0 \quad 0.9)^T;$$

для x_6:

$$\|x_6 - w_1(1)\|^2 = 1.01 = \min,$$

$$\|x_6 - w_2(2)\|^2 = 2.8181;$$

$$w_1(2) = w_1(1) - 0.1[x_6 - w_1(1)] = (0 \quad 0.1 \quad 1 \quad 1)^T - 0.1[(0 \quad 1 \quad 1 \quad 0)^T -$$

$$- (0 \quad 0.1 \quad 1 \quad 1)^T] = (0 \quad 0.01 \quad 1 \quad 1.1)^T;$$

для x_7:

$$\|x_7 - w_1(2)\|^2 = 1.0101 = \min,$$

$$\|x_7 - w_2(2)\|^2 = 2.1981;$$

$$w_1(3) = w_1(2) + 0.1[x_7 - w_1(2)] = (0 \quad 0.01 \quad 1 \quad 1.1)^T + 0.1[(1 \quad 0 \quad 1 \quad 1)^T -$$

$$- (0 \quad 0.01 \quad 1 \quad 1.1)^T] = (0.1 \quad 0.009 \quad 1 \quad 1.09)^T;$$

для x_8:

$$\|x_8 - w_1(3)\|^2 = 2.99,$$

$$\|x_8 - w_2(2)\|^2 = 1.01 = \min;$$

$$w_2(3) = w_2(2) + 0.1[x_8 - w_2(2)] = (1 \quad 1.09 \quad 0 \quad 0.9)^T + 0.1[(1 \quad 1 \quad 1 \quad 1)^T -$$

$$- (1 \quad 1.09 \quad 0 \quad 0.9)^T] = (1 \quad 1.081 \quad 0.1 \quad 0.91)^T.$$

Первый цикл обучения окончен. Полученные значения векторов весов

$$w_1 = (0.1 \quad 0.009 \quad 1 \quad 1.09)^T;$$

$$w_2 = (1 \quad 1.081 \quad 0.1 \quad 0.91)^T$$

приводят к тому, что сеть относит x_4 ко второму классу, а x_6 – к первому. Продолжая обучение сети, получаем, что правильная классификация будет достигнута после 12 цикла. При этом

$$w_1 = (0.2251 \quad 0.0816 \quad 1 \quad 1.1419)^T;$$

$$w_2 = (0.9269 \quad 1.2046 \quad 0.2980 \quad 0.7130)^T.$$

После 500 циклов обучения с $\alpha = 0.1$ получим следующие значения искомых векторов весов:

$$w_1 = (0.4364 \quad 0.2116 \quad 0.8087 \quad 1.0326)^T;$$

$$w_2 = (0.7901 \quad 1.0488 \quad 0.455 \quad 0.5481)^T.$$

Несложно проверить, что при данных весах все предъявляемые сети векторы будут классифицированы правильно.

Пример 9.2. Рассмотрим работу алгоритма OLVQ1 на тестовой задаче из примера 9.1. В качестве начального значения параметра $\alpha_C(0)$ принималось $\alpha_C(0)= 0.3$. После первого цикла обучения с шагом 0.1 было получено

$$w_1 = (0.3000 \quad 0.0969 \quad 1.0000 \quad 1.1615)^T;$$

$$w_2 = (1.0000 \quad 1.1200 \quad 0.3000 \quad 0.7000)^T;$$

$\alpha = 0.2308$.

Это привело к тому, что неправильно распознался вектор x_4 (при использовании метода LVQ1 после первого цикла обучения неправильно распознались векторы x_4 и x_6).

После второго цикла обучения все вектора были правильно классифицированы. При этом

$$w_1 = (0.3639 \quad 0.1051 \quad 1.0000 \quad 1.2233)^T;$$

$$w_2 = (1.0000 \quad 1.1857 \quad 0.3588 \quad 0.6412)^T;$$

$\alpha = 0.1579$.

Таким образом, применение метода OLVQ1 привело к получению решения за два цикла обучения (метод LVQ1 обеспечил получение решения после 12 циклов).