Содержание

СЕТИ И КАРТЫ КОХОНЕНА

Горбаченко В. И.

Рассматриваются архитектура и алгоритмы обучения наиболее известных разновидностей сетей Кохонена: слои (сети) Кохонена, реализующие кластеризацию входных векторов, самоорганизующиеся карты Кохонена (*SOM — self-organizing maps*), реализующие визуализацию многомерных данных на плоскости, сети векторного квантования, обучаемые с учителем (*LVQ — learning vector quantization*), сети встречного распространения.

1. Сети Кохонена

1.1. Принципы построения сети Кохонена

Сети (слои) Кохонена (*Kohonen T.*) [1–3] относятся к *самоорганизующимся* нейронным сетям. Самоорганизующаяся сеть позволяет выявлять *кластеры* (группы) входных векторов, обладающих некоторыми общими свойствами.

Кластеризация — это разделение исследуемого множества объектов на группы "похожих" объектов, называемых *кластерами* [4–5]. Синонимами термина "кластер" (англ. *Cluster* — сгусток, пучок, группа) являются термины класс, таксон, сгущение. Задача кластеризации принципиально отличается от задачи классификации. Решением задачи *классификации* является отнесение каждого из объектов к одному из *заранее определенных* классов. В задаче кластеризации происходит отнесение объекта к одному из *заранее неопределенных* классов. Разбиение объектов по кластерам осуществляется *при одновременном формировании кластеров*.

Кластеризация позволяет сгруппировать сходные данные, что облегчает решение ряда задач Data Mining [4]:

- *Изучение данных, облегчение анализа*. Содержательный анализ полученных кластеров позволяет обнаружить закономерности. Например, можно выявить группы клиентов сети сотовой связи, для которых можно предложить новый тарифный план. Другие примеры — выявление групп покупателей торговой сети, сегментация рынка. Анализ содержания кластера позволяет применить к объектам различных кластеров разные методы анализа.
- *Прогнозирование*. Относя новый объект к одному из кластеров, можно прогнозировать поведение объекта, поскольку его поведение будет схожим с поведением объектов кластера.
- *Обнаружение аномалий*. Содержательный анализ кластеров помогает выявить аномалии. Обычно, это кластеры, в которые попадает мало объектов.

Важно отметить роль *содержательной интерпретации каждого кластера*. Каждому кластеру необходимо присвоить содержательное название, отражающее суть объектов кластера. Для этого необходимо выявить, признаки, объединяющие объекты в кластер. Это может потребовать статистического анализа свойств объекта кластера.

С помощью сетей Кохонена производится кластеризация объектов, описываемых количественными характеристиками.

Формально *задача кластеризации* описывается следующим образом [6]. Дано множество объектов $I = \{i_1, i_2, \ldots, i_n\}$, каждый из которых характеризуется вектором $\mathbf{x}_j$, $j = 1,2,\ldots,n$ атрибутов (параметров): $\mathbf{x}_j = \{x_{j1}, x_{j2}, \ldots, x_{jm}\}$. Требуется построить множество кластеров C и отображение F множества I на множество C, то есть $F : I \to C$. Задача кластеризации состоит в построении множества

$$C = \{c_1, c_2, \ldots, c_k, \ \ldots, c_g\},$$

где c_k — *кластер*, содержащий "похожие" объекты из множества I:

$$c_k = \left\{ i_j, i_p \middle| i_j \in I, i_p \in I \text{ и } d\left(i_j, i_p\right) < \sigma \right\}, \qquad (1)$$

σ — величина, определяющая меру близости для включения объектов в один кластер, $d\left(i_j, i_p\right)$ — мера близости между объектами, называемая *расстоянием*.

Если расстояние $d\left(i_j, i_p\right)$ меньше некоторого значения σ, то объекты считаются близкими и помещаются в один кластер. В противном случае считается, что объекты отличны друг от друга и их помещают в разные кластеры. Условие (1) известно как *гипотеза компактности*.

Кластеризация основана на использовании расстояния между векторами [3, 5]. Неотрицательное число $d\left(\mathbf{x}, \mathbf{y}\right)$ называется *расстоянием* (*метрикой*) между векторами $\mathbf{x}$ и $\mathbf{y}$, если выполняются следующие условия:

1. $d\left(\mathbf{x}, \mathbf{y}\right) \geq 0$ для всех $\mathbf{x}$ и $\mathbf{y}$.
2. $d\left(\mathbf{x}, \mathbf{y}\right) = 0$, тогда и только тогда, когда $\mathbf{x} = \mathbf{y}$.
3. $d\left(\mathbf{x}, \mathbf{y}\right) = d\left(\mathbf{y}, \mathbf{x}\right)$.
4. $d\left(\mathbf{x}, \mathbf{y}\right) \leq d\left(\mathbf{x}, \mathbf{k}\right) + d\left(\mathbf{k}, \mathbf{y}\right)$ — неравенство треугольника.

В сетях Кохонена обычно применяется *евклидово расстояние*

$$d_E\left(\mathbf{x}, \mathbf{y}\right) = \sqrt{\sum_{i=1}^{m} \left(x_i - y_i\right)^2} = \left\| \mathbf{x} - \mathbf{y} \right\|.$$

Евклидово расстояние между векторами $\mathbf{x}$ и $\mathbf{y}$ представляет собой евклидову норму разности векторов, или длину отрезка, соединяющего точки $\mathbf{x}$ и $\mathbf{y}$.

Евклидово расстояние является частным случаем *расстояния Минковского* (*H. Minkowski*)

$$d_p\left(\mathbf{x}, \mathbf{y}\right) = \left(\sum_{i=1}^{m} \left| x_i - y_i \right|^p \right)^{1/p} = \left\| \mathbf{x} - \mathbf{y} \right\|_p,$$

где $\left\| \mathbf{z} \right\|_p = \left(\sum_{i=1}^{m} \left| z_i \right|^p \right)^{1/p}$ — p-норма вектора $\mathbf{z}$.

Тогда 2-норма — это евклидова норма.

Другой частный случай — 1-норма, которая называется *манхэттенским расстоянием* (расстоянием городских кварталов)

$$d_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{m} |x_i - y_i|.$$

Манхеттенское расстояние — это расстояние, которое мы проходим, двигаясь параллельно осям координат, как в Манхеттнене или других городах с прямоугольной продольно-поперечной планировкой улиц. Известны и другие виды расстояний [5].

Сеть (*слой*) *Кохонена* (рис. 1) — это однослойная сеть, построенная из нейронов типа WTA (*Winner Takes All* — победитель получает все) — см. раздел 2.6.

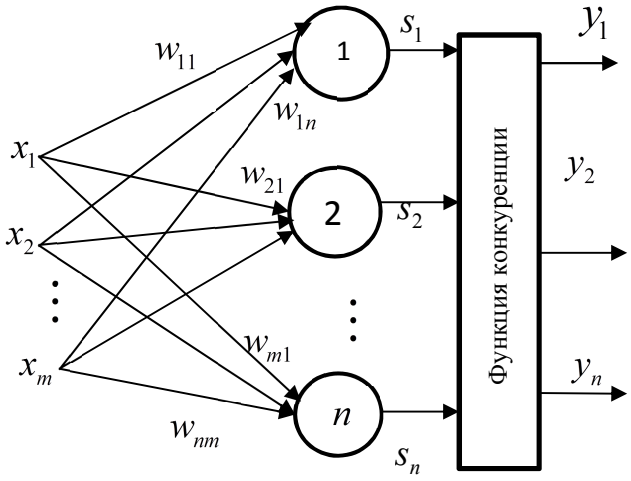

Рис. 1. Структура сети Кохонена

Каждый нейрон сети соединен со всеми компонентами m-мерного входного вектора $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$. Входной вектор — это описание одного из объектов, подлежащих кластеризации. Количество нейронов совпадает с количеством кластеров, которое должна выделить сеть. В качестве нейронов сети Кохонена применяются линейные взвешенные сумматоры

$$s_j = b_j + \sum_{i=1}^{m} w_{ij} x_i ,$$

где j — номер нейрона, i — номер входа, s_j — выход адаптивного сумматора, w_{ij} — вес i-го входа j-го нейрона, b_j — порог.

Каждый j-ый нейрон описывается вектором весов $\mathbf{w}_j = \left(w_{1j}, w_{2j}, \ldots, w_{mj}\right)$, где m – число компонентов входных векторов. С выходов адаптивных сумматоров сигнал поступает на функцию конкуренции, работающую по правилу "победитель получает всё". Функция конкуренции находит выход адаптивный сумматор с максимальным значением выхода. Пусть k — номер такого сумматора. Тогда на выходе сети формируется выходной сигнал $y_k = 1$, остальные выходные сигналы равны нулю. Если максимум достигается одновременно на выходах нескольких сумматоров, то выходной сигнал, равный единице, соответствует одному из них, например, первому.

Обучение сети Кохонена представляет собой подбор значений весов, минимизирующих ошибки от замены близких в смысле используемой метрики входных векторов вектором весов. Такой подход называется векторным квантованием [7] и применяется в задачах сжатия аудио- и видеосигналов. Идея векторного квантования состоит в компактном представлении многомерных входных векторов с помощью ограниченного набора *опорных векторов* меньшей размерности, образующих *кодовую таблицу*. В случае сети Кохонена входные векторы кодируются номерами нейронов-победителей (номерами кластеров). Таким образом, все векторы из некоторой области входного пространства заменяются одним и тем же опорным вектором, являющимся их ближайшим соседом. При использовании евклидова расстояния входное пространство разбивается на *многогранники* (мозаику) *Вороного* (Вороной Г. Ф.). Пример [3] многогранников Вороного для двумерного пространства приведен на рис. 2.

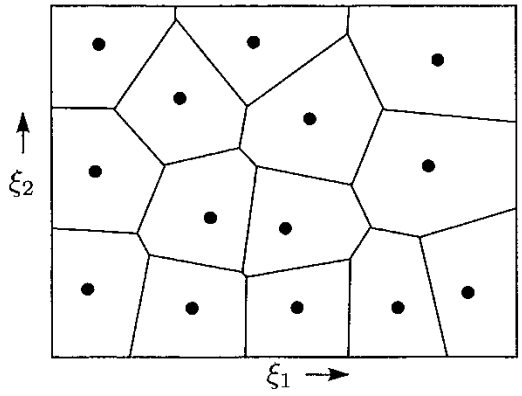

Рис. 2. Пример многогранников Вороного [8]

В многомерном пространстве мозаика Вороного образуется гиперплоскостями.

1.2. Обучение сети Кохонена

В сетях Кохонена используется обучение без учителя. Для обучения сети применяются *механизмы конкуренции*. При подаче на вход сети вектора $\mathbf{x}$ побеждает тот нейрон, вектор весов которого в наименьшей степени отличаются от входного вектора. Для нейрона-победителя выполняется соотношение

$$d\left(\mathbf{x},\mathbf{w}_j\right)=\min_{1\le i\le n} d\left(\mathbf{x},\mathbf{w}_i\right),$$

где n — количество нейронов, j — номер нейрона-победителя, $d\left(\mathbf{x},\mathbf{w}\right)$ — расстояние (в смысле выбранной метрики) между векторами $\mathbf{x}$ и $\mathbf{w}$.

Чаще всего в качестве меры расстояния используется е*вклидова мера*

$$d\left(\mathbf{x},\mathbf{w}_i\right)=\left\|\mathbf{x}-\mathbf{w}_i\right\|=\sqrt{\sum_{j=1}^{m}\left(x_j-w_{ji}\right)^2}\ .$$

Используются и другие меры расстояния (метрики).

Конкурирующая функция активации анализирует значения сумматоров и формирует выходы нейронов, равные 0 для всех нейронов, кроме одного "нейрона-победителя", имеющего на выходе максимальное значение. Таким образом, вектор выхода имеет единственный элемент, равный 1, который соответствует нейрону-победителю, а остальные равны 0. *Номер активного*

нейрона определяет ту группу (кластер), к которой наиболее близок входной вектор.

В сети Кохонена входные значения желательно (хотя и не обязательно) *нормировать*. Для этого следует воспользоваться одной из следующих формул:

$$x_{ні} = \frac{x_i}{\sqrt{\sum_{i=1}^{m} x_i^2}}, \quad x_{ні} = \frac{x_i}{|x_i|},$$

где $x_{ні}$ — нормированный компонент входного вектора.

Нормирование входных данных положительным образом сказывается на скорости обучения сети.

Перед процессом обучения производится *инициализация сети*, то есть первоначальное задание векторов весов. В простейшем случае задаются случайные значения весов. *Процесс обучения сети Кохонена* состоит из циклического повторения ряда шагов:

1. Подача исходных данных на входы. Обычно это случайная выборка одного из входных векторов.

2. Нахождение выхода каждого нейрона.

3. Определение "выигравшего" нейрона (веса которого в наименьшей степени отличаются от соответствующих компонентов входного вектора), или нейрона-победителя.

4. Корректировка весов "выигравшего" нейрона по *правилу Кохонена*

$$\mathbf{w}_i^{(k+1)} = \mathbf{w}_i^{(k)} + \eta_i^{(k)}\left[\mathbf{x} - \mathbf{w}_i^{(k)}\right], \qquad (2)$$

где $\mathbf{x}$ — входной вектор, k — номер цикла обучения, $\eta_i^{(k)}$ — коэффициент скорости обучения i-го нейрона в k-ом цикле обучения.

5. Переход на шаг 1, если обучение не завершено. Часто, например, в Neural Network Toolbox MATLAB, задается число циклов обучения. Можно проверять достижение малой величины функционала ошибки

$$E = \frac{1}{Q}\sum_{i=1}^{Q}\left\|\mathbf{x}_i - \mathbf{w}_{\mathbf{x}_i}\right\|^2, \qquad (3)$$

где $\mathbf{w}_{\mathbf{x}_i}$ — вектор весов нейрона-победителя при предъявлении входного вектора $\mathbf{x}_i$, Q — размер обучающей выборки.

Таким образом, нейрон, чей вектор весов был ближе к входному вектору, обновляется, чтобы быть еще ближе. В результате этот нейрон, скорее всего, выиграет конкуренцию при подаче на вход близкого вектора и проиграет при подаче существенно отличающегося вектора. После многократной подачи обучающих векторов будет иметься нейрон, который выдает 1, когда вектор принадлежит кластеру, и 0, когда вектор не принадлежит кластеру. Таким образом, сеть учится классифицировать входные векторы.

При обучении сети Кохонена возникает проблема так называемых *"мертвых" нейронов*. Одно из ограничений всякого конкурирующего слоя состоит в том, что некоторые нейроны оказываются незадействованными. Это проявляется в том, что нейроны, имеющие начальные весовые векторы, значительно удаленные от векторов входа, никогда не выигрывают конкуренции, независимо от того, как долго продолжается обучение. В результате оказывается, что такие векторы не используются при обучении и соответствующие нейроны никогда не оказываются победителями. Такие "нейроны-неудачники" называют *"мертвыми" нейронами,* поскольку они не выполняют никакой полезной функции. Таким образом, входные данные будут интерпретироваться меньшим числом нейронов. Поэтому надо дать шанс победить всем нейронам. Для этого алгоритм обучения модифицируют таким образом, чтобы "мертвые" нейроны участвовали в обучении.

Например [1], алгоритм обучения модифицируют таким образом, чтобы нейрон-победитель терял активность. Одним из приемов учета активности нейронов является подсчет *потенциала* p_i каждого нейрона в процессе обучения. Первоначально нейронам присваивается потенциал $p_i(0) = \frac{1}{n}$, где

n — число нейронов (кластеров). В k-ом цикле обучения потенциал определяется по правилам:

$$p_i(k) = \begin{cases} p_i(k-1) + \dfrac{1}{n}, & i \neq j, \\ p_i(k-1) - p_{\min}, & i = j, \end{cases}$$

где j — номер нейрона-победителя.

Если значение потенциала $p_i(k)$ падает ниже уровня $p_{\min}$, то нейрон исключается из рассмотрения — "отдыхает". При $p_{\min} = 0$ нейроны не исключаются из борьбы. При $p_{\min} = 1$ нейроны побеждают по очереди, так как в каждый цикл обучения только один из них готов к борьбе. На практике хороший результат получается при $p_{\min} \approx 0.75$.

В *Neural Network Toolbox* для борьбы с "мертвыми" нейронами используется изменение смещения нейронов [9]. Соответствующее правило настройки, учитывающее нечувствительность мертвых нейронов, реализовано в виде функции **`learncon`** и заключается в следующем. В начале процедуры настройки всем нейронам конкурирующего слоя присваивается одинаковый параметр активности $c_0 = \dfrac{1}{N}$, где N — количество нейронов конкурирующего слоя, равное числу кластеров. В процессе настройки функция **`learncon`** корректирует этот параметр таким образом, чтобы его значения для активных нейронов становились больше, а для неактивных нейронов меньше. Соответствующая формула для вектора параметров активности выглядит следующим образом:

$$\mathbf{c}^{(k+1)} = (1 - r_l)\mathbf{c}^{(k)} + r_l \mathbf{s}^{(k)},$$

где r_l — параметр скорости настройки; k — номер цикла обучения; $\mathbf{s}^{(k)}$ — вектор выходов адаптивных сумматоров в k-ом цикле обучения.

Компоненты вектора смещения вычисляется по формуле

$$b_i^{(k+1)} = e^{\left(1 - \ln c_i^{(k+1)}\right)} - b_i^{(k)}.$$

Смещение для нейрона-победителя уменьшится, а смещения для остальных нейронов немного увеличатся. Параметр скорости настройки r_l по умолчанию равен 0.001. Увеличение смещений для неактивных нейронов позволяет расширить диапазон покрытия входных значений, и неактивный нейрон начинает формировать кластер. В конечном счете, он может начать притягивать новые входные векторы. Это дает два преимущества. Если нейрон не выигрывает конкуренции потому, что его вектор весов существенно отличается от векторов, поступающих на вход сети, то его смещение по мере обучения становится достаточно большим, и он становится конкурентоспособным. Когда это происходит, его вектор весов начинает приближаться к некоторой группе векторов входа. Как только нейрон начинает побеждать, его смещение начинает уменьшаться. Таким образом, задача активизации "мертвых" нейронов оказывается решенной. Второе преимущество, связанное с настройкой смещений, состоит в том, что они позволяют выровнять значения параметра активности и обеспечить притяжение приблизительно одинакового количества векторов входа. Таким образом, если один из кластеров притягивает большее число векторов входа, чем другой, то более заполненная область притянет дополнительное количество нейронов и будет поделена на меньшие по размерам кластеры.

2. Карты Кохонена

2.1. Принципы построения карт Кохонена

Карты Кохонена (*самоорганизующиеся карты,* или *SOM — self-organizing map*) [1–4, 10] предназначены для визуального представления *многомерных* свойств объектов на двумерной карте. Карты Кохонена производят отображение входных данных высокой размерности на элементы регулярного массива малой размерности (обычно, двумерного). Карты Кохонена похожи на сети Кохонена. Отличие состоит в том, что в карте нейроны, являющиеся центрами кластеров, упорядочены в некоторую

структуру (обычно двумерную сетку). В процессе обучения карты настраиваются веса не только нейрона-победитель, но и его соседей. В результате близкие по некоторой метрике входные векторы в сети Кохонена относятся к одному нейрону (центру кластера), а в карте Кохонена могут относиться к разным близко расположенным на сетке нейронам. Обычно нейроны располагаются в узлах двумерной сетки с прямоугольными или шестиугольными ячейками. Нейроны-соседи определяются расстоянием между нейронами на карте. На рис. 3 показаны шестиугольные и прямоугольные ячейки, в центрах которых располагаются нейроны. Шестиугольные ячейки более корректно отображают декартово расстояние между объектами на карте, т. к. для этих ячеек расстояние между центрами смежных ячеек одинаковы.

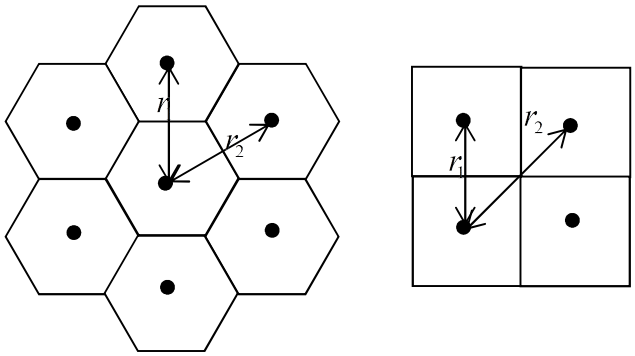

Рис. 3. Шестиугольные и прямоугольные ячейки

Каждой ячейке соответствует нейрон сети Кохонена. То есть в карте Кохонена число нейронов равно числу ячеек карты и больше числа нейронов сети Кохонена, равного числу кластеров. Число ячеек карты зависит от требуемой детальности изображения и подбирается экспериментально.

Для каждой ячейки вычисляется одна из статистических характеристик выбранного компонента входных векторов, попавших в ячейку. В зависимости от величины этой характеристики ячейка окрашивается в определенный цвет (подробнее см. раздел 5.2.3).

Карты Кохонена только позволяют по раскраске ячеек карты выдвигать гипотезы о наличии кластерной структуры и числе кластеров, зависимостях между значениями отдельных переменных. Выдвинутые гипотезы должны проверяться и подтверждаться иными способами. Карты используются на стадии разведочного анализа данных, скорее для общего понимания задачи, чем для получения каких-либо точных результатов.

2.2. Обучение карты Кохонена

2.2.1. Алгоритм последовательного обучения

В *алгоритме последовательного* (*Iterative*) *обучения* обновление весов производится после каждого обучающего примера. Обучение нейронов производится аналогично обучению нейронов сети Кохонена. Отличие состоит в том, что кроме нейрона-победителя обучаются нейроны, входящие в *окрестность* (*neighborhood*), или *радиус обучения* (*radius of learning*) нейрона-победителя. Нейрон принадлежит окрестности нейрона-победителя, если расстояние между ним и нейроном-победителем *на карте* меньше определенной величины (в процессе обучения изменяются веса нейронов, но их положение на карте не изменяется). Такой алгоритм называют алгоритмом типа *WTM* (*Winner Takes Most* — победитель получает больше). В классическом алгоритме веса нейрона-победителя и всех нейронов, лежащих в пределах его окрестности, подвергаются обучению (адаптации) по несколько измененному *правилу Кохонена* (2). Веса нейронов, находящихся за пределами окрестности не изменяются. Размер окрестности и коэффициент скорости обучения являются функциями, значения которых уменьшаются с увеличением номера цикла обучения.

Изменение в правиле Кохонена состоит в том, что коэффициент скорости обучения $\eta_i^{(k)}$ разбивается на две части: функцию соседства $\eta_i(d,k)$ и функцию скорости обучения $a(k)$

$$\eta_i^{(k)} = \eta_i(d,k) \cdot a(k). \tag{4}$$

В качестве *функции соседства* применяется или константа

$$\eta_i(d,k) = \begin{cases} const, & d_i \le \sigma(k), \\ 0, & d_i > \sigma(k) \end{cases} \tag{5}$$

или Гауссова функция

$$\eta_i(d,k) = e^{-\frac{d_i}{2\sigma(k)}}.$$

При этом лучший результат получается при использовании Гауссовой функции расстояния. В (4) и (5) $d_i = \left\| \mathbf{r}_i - \mathbf{r}_{c_j} \right\|$ — расстояние между i-м нейроном и нейроном-победителем c_j, $\mathbf{r}_i$ и $\mathbf{r}_{c_j}$ — *координаты на сетке карты* i-го и победившего c_j-го нейронов, $\left\| \mathbf{r}_i - \mathbf{r}_{c_j} \right\|$ — расстояние между ячейками i и c_j на сетке карты. Функция $\sigma(k)$ является убывающей функцией от номера цикла обучения. Наиболее часто используется функция, линейно убывающая от номера цикла обучения.

Рассмотрим теперь *функцию скорости обучения* $a(k)$. Эта функция также представляет собой функцию, убывающую от номера цикла обучения. Наиболее часто используются два варианта этой функции: линейная и обратно пропорциональная от номера цикла обучения вида

$$a(k) = \frac{A}{k + B},$$

где A и B — константы. Применение этой функции приводит к тому, что все векторы из обучающей выборки вносят примерно равный вклад в результат обучения.

Обучение состоит из двух основных этапов: на первоначальном этапе — *этапе упорядочения векторов весовых коэффициентов в пространстве признаков* выбирается достаточно большое значение скорости обучения и радиуса обучение, что позволяет расположить векторы нейронов в соответствии с распределением примеров в выборке (изменяется положение

векторов нейронов в пространстве признаков, но не на карте). На заключительном этапе — *этапе настройки* производится точная подстройка весов, когда значения параметров скорости обучения много меньше начальных. Обучение продолжается до тех пор, пока погрешность сети (3) не станет достаточно малой.

2.2.2. Алгоритм пакетного обучения

В настоящее время для обучения карт Кохонена широко используется *алгоритм пакетного обучения карты Кохонена* (*Batch-Learning Self-Organizing Map*) [3, 11]. В этом алгоритме сначала предъявляются все примеры, а потом производится обновление весов. Алгоритм использует нормированные входные векторы. Алгоритм состоит из ряда проходов В k-ом проходе выполняются следующие шаги.

1. Подаются все N входных векторов, и вычисляется евклидово расстояние между каждым входным вектором $\mathbf{x}_j$ и векторами весов $\mathbf{w}_i$ всех нейронов. Определяется номер нейрона-победителя

$$c_j = \arg\min_i \left\{ \left\| \mathbf{w}_i - \mathbf{x}_j \right\| \right\}.$$

2. Обновляются все векторы весов как усредненные значения компонентов всех входных векторов

$$\mathbf{w}_i^{new} = \frac{\sum_{j=1}^{N} h_{c_j,i} \mathbf{x}_j}{\sum_{j=1}^{N} h_{c_j,i}},$$

где $h_{c_j,i}$ — функция окрестности

$$h_{c_j,i} = \exp\left(-\frac{\left\| \mathbf{r}_i - \mathbf{r}_{c_j} \right\|^2}{2\sigma^2(k)} \right),$$

$\mathbf{r}_i$ и $\mathbf{r}_{c_j}$ — координаты на сетке карты i-го и победившего c_j-го нейронов, $\left\|\mathbf{r}_i - \mathbf{r}_{c_j}\right\|$ — расстояние между ячейками i и c_j на сетке карты, параметр $\sigma(k)$ уменьшается с увеличением номера прохода

$$\sigma(k) = \sigma_0\left(1 - k/k_{\max}\right),$$

σ_0 — подбираемое начальное значение, $k_{\max}$ — максимальное число проходов.

Проходы повторяются, пока функционал (3) не станет достаточно малым.

2.2.3. Алгоритм нейронного газа

Алгоритм нейронного газа [1, 12], названный из-за подобия его динамики движению газа, обеспечивает бóльшую скорость сходимости, чем рассмотренные алгоритмы. В этом алгоритме на каждой k-ом цикле обучения *все нейроны* сортируются в порядке удаленности от нейрона-победителя. *Для каждого нейрона* определяется значение функции соседства. Для i-го нейрона эта функция имеет вид

$$\eta_i^{(k)} = e^{-\frac{m_i}{\lambda^{(k)}}}, \qquad (6)$$

где m_i — номер нейрона, полученный в результате сортировки (для нейрона-победителя этот номер равен нулю); λ — параметр (параметр ширины), аналогичный параметру в алгоритме последовательного обучения, уменьшающийся с номером итерации.

Корректировка вектора весов i-го нейрона производится по формуле

$$\mathbf{w}_i^{(k+1)} = \mathbf{w}_i^{(k)} + \eta_i^{(k)} a_i^{(k)}\left[\mathbf{x}^{(k)} - \mathbf{w}_i^{(k)}\right],$$

где $\eta_i^{(k)}$ — функция соседства, определяемая по формуле (6); $a_i^{(k)}$ — функция скорости обучения.

Вектор весов нейрона-победителя уточняется по алгоритму WTA. Но в отличие от алгоритма WTA уточняются веса *всех* нейронов.

Параметры $\lambda^{(k)}$ и $a_i^{(k)}$ должны уменьшаться с ростом номера итерации

$$\lambda^{(k)} = \lambda_{\max}\left(\frac{\lambda_{\min}}{\lambda_{\max}}\right)^{k/k_{\max}}, \quad a_i^{(k)} = a_i^{(0)}\left(\frac{a_{\min}}{a_i^{(0)}}\right)^{k/k_{\max}},$$

где k — номер текущей итерации, $k_{\max}$ — максимальное заданное количество итераций, $\lambda_{\min}$ и $\lambda_{\max}$ — принятые минимальное и максимальное значения параметра λ, $a_i^{(0)}$ — начальное значение скорости обучения, $a_{\min}$ — заданное минимальное значение скорости обучения, соответствующее $k_{\max}$.

Для уменьшения объема вычислений при $m_i > K$, где K — задаются, полагают $\eta_i^{(k)} = 0$. То есть настраиваются только K нейронов.

Все рассмотренные алгоритмы требовали задания количества нейронов. *Алгоритм растущего* (*расширяющегося*) *нейронного газа* (*Growing Neural Gas*) [13] позволяет не только производить кластеризацию входных данных, но и вычисляет количество нейронов в процессе обучения сети.

2.2.4. Инициализация карты Кохонена

Перед построением карты Кохонена необходимо задать конфигурацию сетки нейронов (обычно, шестиугольная) и количество нейронов карты. Количество нейронов определяет степень детализации карты. При этом карта с бóльшим количеством нейронов требует бóльшего времени обучения.

Далее производится *инициализация* — присвоение начальных значений весам нейронов. В простейшем случае веса можно инициализировать малыми случайными числами. Известно [3], что с точки зрения сокращения времени обучения сети, полезно любое упорядочение начального состояния карты. Лучшее результаты дает использование в качестве начальных значений весов значений случайно выбранных примеров из обучающей выборки. Полезна так называемая *линейная инициализация* [3],

обеспечивающая упорядоченное начальное состояние весов карты. Для этого, аналогично разделу 3.2.4.4, входные векторы преобразуются в центрированные, то есть в векторы с нулевым математическим ожиданием, строится матрица $\dot{\mathbf{X}} = \left[\dot{\mathbf{x}}_1\ \dot{\mathbf{x}}_2 \ldots \dot{\mathbf{x}}_n\right]$, составленная из центрированных входных векторов, и вычисляется ковариационная матрица (3.13)

$$\mathbf{K} = \frac{1}{n-1}\dot{\mathbf{X}}\dot{\mathbf{X}}^{\mathrm{T}}.$$

Находятся два наибольших собственных значения матрицы $\mathbf{K}$ (эти значения положительные, так как матрица положительно определена). Строится прямоугольник с прямоугольной или шестиугольной правильной решеткой, размеры которого равны двум наибольшим собственным значениям матрицы $\mathbf{K}$. Начальные значения весов нейронов строятся как линейные комбинации двух собственных векторов матрицы $\mathbf{K}$, соответствующих наибольшим собственным значениям. Веса линейных комбинаций берутся равными координатам нейрона в построенном прямоугольнике. Математически это означает, что начальные значения весов берутся из подпространства, натянутого (см., например, [14]) на два главных собственных вектора матрицы $\mathbf{K}$.

Начальные значения компонентов векторов весов будут упорядочены и хорошо приближать веса карты. Поэтому можно начинать обучение непосредственно с этапа настройки. Но линейная инициализация может порождать "пустые" нейроны, веса которых не являются близкими никаким входным векторам. Еще лучшие начальные значения весов дает использование нелинейного метода главных компонент [15].

Инициализация карты Кохонена может быть произведена с использованием одного из алгоритмов кластеризации [16], например, *k-means* [4–6]: с помощью алгоритма кластеризации формируется столько кластеров, сколько нейронов содержит карта. Далее производится точная настройка карты.

Сложные методы инициализации сокращают время обучения сети, но увеличивают время инициализации. Поэтому необходимо учитывать суммарные затраты времени на создание карты.

2.3. Графическое представление карт Кохонена

Объекты, векторы признаков которых близки, попадают в одну ячейку или в ячейки, расположенные на карте вблизи. Следовательно, *двумерная* карта Кохонена отражает на плоскости близость *многомерных* векторов признаков. Обычно требуется анализировать, по каким конкретно параметрам проявляется сходство объектов. Для этого используется *раскраска карт Кохонена*. Для этого *строится столько карт, сколько параметров* (компонентов входных векторов) *анализируется*. Каждая карта соответствует одному параметру объекта. Ячейки карты раскрашиваются в разные цвета (или оттенки серого цвета) в зависимости от значения весов нейронов, соответствующих каждой ячейке. Выделяются диапазоны значений весов. Каждому диапазону ставится в соответствие цвет (или оттенок серого), и ячейки карты "раскрашиваются" соответствующими цветами. Пример графика весов, часто называемого *компонентной плоскостью — Component Planes,* показан на рис. 4 (график получен в MATLAB Neural Network Toolbox). На графике более темные цвета представляют большие веса. Одинаковыми цветами отмечаются близкие веса. На рис. 4 видно, что исследуемые объекты резко отличаются по первому и второму входам и близки по третьему и четвертому входам.

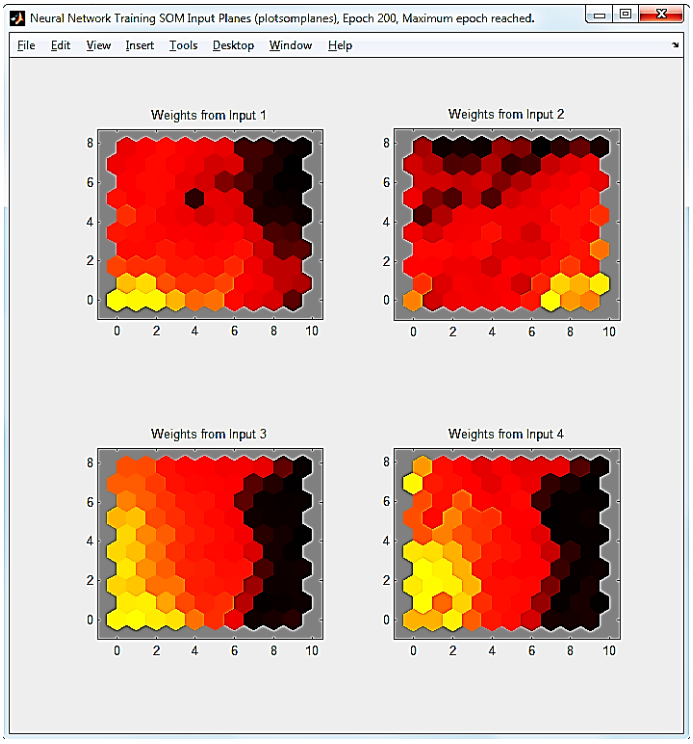

Рис. 4. Компонентные плоскости

Для анализа полезно знать, сколько векторов входных данных связано с каждой ячейкой (нейроном) карты. Для этого в MATLAB Neural Network Toolbox строится *График попаданий примеров в кластеры* (рис. 5).

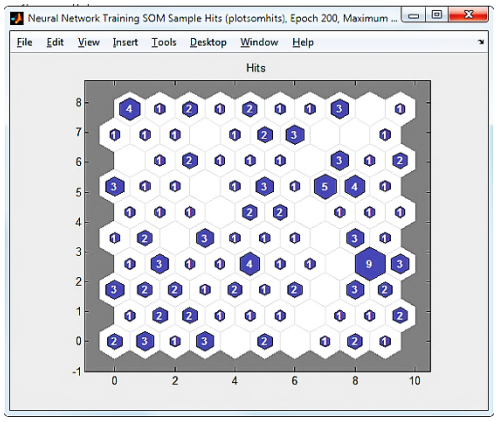

Рис. 5. График попаданий примеров в кластеры

Для анализа также полезно оценивать расстояния между векторами весов соседних нейронов. Для этого используются *унифицированная матрица расстояний — U-matrix* и *отображение Семмона* (*J. W. Sammon*) [17]. Элементы у*нифицированной матрицы расстояний — U-matrix* [18]

показывают расстояние между весами нейрона и весами его ближайших соседей. Большое значение говорит о том, что данный нейрон сильно отличается от окружающих и относится к другому кластеру. На рис. 6 показана матрица расстояний, построенная в MATLAB Neural Network Toolbox. Шестиугольники представляют нейроны. Линии показывают связи между соседними нейронами. Цвета в участках, содержащих связи, показывают расстояния между нейронами. Более темные цвета представляют бóльшие расстояния. Полоса темных сегментов пересекает карту от нижнего региона до правого верхнего региона карты. Карта сгруппировала объекты на два кластера.

Отображение (проекция) Семмона [19] является нелинейным отображением набора входных векторов на плоскость с приближенным сохранением относительного расстояния между векторами. Обычно соседние нейроны соединяются линиями.

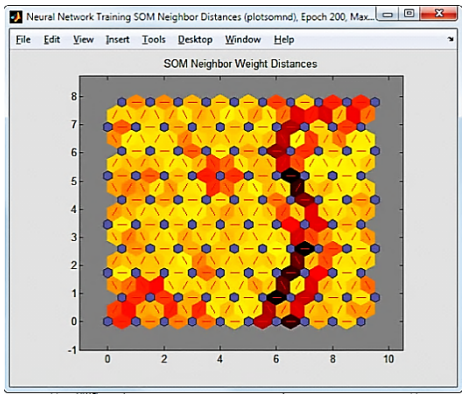

Рис. 6. Унифицированная матрица расстояний

В MATLAB Neural Network Toolbox строится *График положения весов* (рис. 7), который показывает точками координаты входных векторов, определяемые по первым двум компонентам. Аналогично точками другого цвета отображаются координаты первых двух компонентов векторов весов нейронов. Соседние нейроны соединяются линиями. Обратите внимание, что близкими являются нейроны в пространстве признаков, а не расположенные в соседних ячейках.

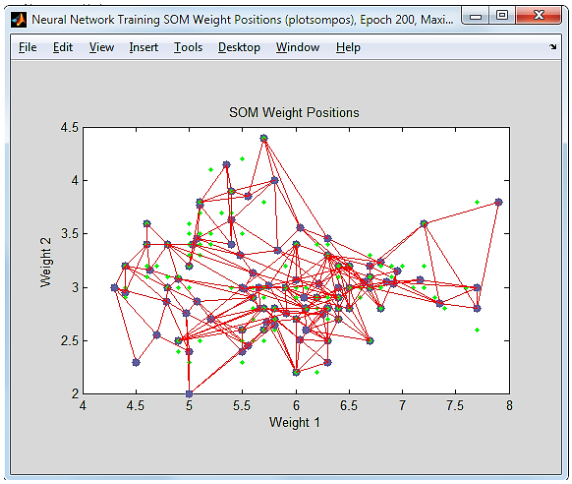

Рис. 7. Положение весов

3. Сети векторного квантования, обучающиеся с учителем (LVQ-сети)

LVQ-сети основаны на обучающемся векторном квантовании (*LVQ* — *Learning Vector Quantization*) [3] и представляют собой слой Кохонена, *обучающийся с учителем*. Для построения *LVQ*-сети задается количество кластеров (нейронов) n, количество классов m ($n \geq m$) и принадлежность каждого кластера определенному классу. Например, анализы пациентов разделяются на 5 кластеров, 2 из которых соответствуют здоровым людям, а 3 — больным. Разделить кластеры по классам можно в тех же пропорциях, что и распределение примеров соответствующих классов в обучающей выборке. Соответствие между номерами кластеров и номерами классов может быть произвольным. Для простоты интерпретации результатов работы сети целесообразно пронумеровать кластеры последовательно. В нашем примере первые два кластера соответствуют классу здоровых людей, а последующие 3 — классу больных. В процессе обучения *LVQ*-сети веса нейронов настраиваются с учетом принадлежности обучающих примеров и кластеров одному классу. Обученная *LVQ*-сеть производит кластеризацию входных векторов с учетом классов. Например, мы узнаем, что анализы конкретного пациента принадлежат одному из кластеров, соответствующих

больным людям. Особенности анализов, попавших в этот кластер, можно узнать только в результате рассмотрения этих анализов.

Известно несколько алгоритмов обучения *LVQ*-сетей [3]. Простейший алгоритм, известный как *LVQ1*, на k-цикле обучения имеет следующий вид.

1. Для очередного вектора $\mathbf{x}$ обучающей выборки находится нейрон с номером c, для которого евклидово расстояние между $\mathbf{x}$ и вектором его весов $\mathbf{w}_c^{(k)}$ минимально.

2. Корректируется вектор весов нейрона-победителя:

– если $\mathbf{w}_c^{(k)}$ и $\mathbf{x}$ принадлежат одному классу:

$$\mathbf{w}_c^{(k+1)} = \mathbf{w}_c^{(k)} + \alpha^{(k)}\left[\mathbf{x} - \mathbf{w}_c^{(k)}\right],$$

– – если $\mathbf{w}_c^{(k)}$ и $\mathbf{x}$ принадлежат различным классам классу:

$$\mathbf{w}_c^{(k+1)} = \mathbf{w}_c^{(k)} - \alpha^{(k)}\left[\mathbf{x} - \mathbf{w}_c^{(k)}\right],$$

где $\alpha^{(k)}$ — коэффициент скорости обучения.

Веса других нейронов не изменяются.

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

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

Если входной вектор классифицируется сетью правильно, то соответствующий вектор весов сдвигается в сторону входного вектора. Если входной вектор классифицируется неправильно, то соответствующий вектор весов сдвигается в противоположную сторону от входного вектора.

Коэффициент скорости обучения $0 < \alpha^{(k)} < 1$ должен монотонно уменьшаться с ростом номера цикла обучения. Но даже начальное значение $\alpha^{(k)}$ должно быть достаточно малым, например, меньше 0,1.

В *алгоритме LVQ2.1* по правилам алгоритма *LVQ1* одновременно корректируются два вектора весов $\mathbf{w}_i$ и $\mathbf{w}_j$, близких к входному вектору $\mathbf{x}$.

Причем один из векторов принадлежит правильному классу, а второй — неправильному. Критерием близости векторов $\mathbf{w}_i$ и $\mathbf{w}_j$ к входному вектору $\mathbf{x}$ является попадание $\mathbf{x}$ в "окно" относительной ширины s. Если d_i и d_j — евклидовы расстояния от $\mathbf{x}$ до $\mathbf{w}_i$ и $\mathbf{w}_j$, то $\mathbf{x}$ попадает в "окно", если

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

Рекомендуется брать ширину "окна" в диапазоне от 0,2 до 0,3.

Известны [3] и другие алгоритмы обучения *LVQ*-сетей. В качестве начальных значений векторов весов можно использовать случайно выбранные векторы обучающих данных, принадлежащие соответствующим классам. При обучении *LVQ*-сетей возможно явление переобучения. Как и в случае многослойных персептронов, явление переобучения обнаруживается путем чередования обучения и тестирования на различных множествах примеров.

Интерпретация результатов работы *LVQ*-сети при большом количестве кластеров требует определенных усилий. Причем зачастую важен только класс, к которому относится входной вектор. В [20] рассматривается реализация *LVQ*-сети из двух слоев: первый слой является слоем Кохонена и производит кластеризация входных векторов на заданное число n кластеров. Второй линейный слой объединяет эти кластеры в m классов ($n \geq m$). В результате сеть производит классификацию входных векторов. Такой подход реализован в Neural Networks Toolbox системы MATLAB [9]. Входной слой обучается одним из алгоритмов обучения *LVQ*-сетей (в Neural Networks Toolbox реализованы алгоритмы *LVQ1* и *LVQ2.1*). Линейный слой не обучается, а формируется в зависимости от известного соответствия между номерами кластеров и классов. Первый слой выдает вектор из n элементов, из которых только один равен единице, остальные равны нулю. Номер единичного элемента в векторе равен номеру кластера, к которому отнесен входной вектор. Матрица весов выходного линейного слоя имеет m строк.

Строки матрицы соответствуют классам. Количество столбцов равно n. Столбцы соответствуют кластерам. В каждом столбце только один элемент равен единице. Этот элемент показывает, к какому классу относится кластер. Произведение матрицы весов линейного слоя на выходной вектор первого слоя формирует вектор из n элементов, номер единственного единичного элемента которого равен номер идентифицированного класса.

4. Сети встречного распространения

Сеть встречного распространения (*Counterpropagation Network*) [21–22] предложена Р. Хехт-Нильсеном (R. Heht-Nielson) [23] и представляет собой двухслойную сеть, первым слоем которой является слой Кохонена, а вторым — слой С. Гроссберга (S. Grossberg) [24]. Сети встречного распространения уступают по точности многослойным персептронам, но быстро обучаются и обладают рядом полезных свойств. Сети встречного распространения используют нормированные векторы.

Нейроны слоя Гроссберга представляют собой так называемые *выходные звезды Гроссберга* (*Outstar*). Выходная звезда Гроссберга (рис. 8) имеет скалярный вход и векторный выход.

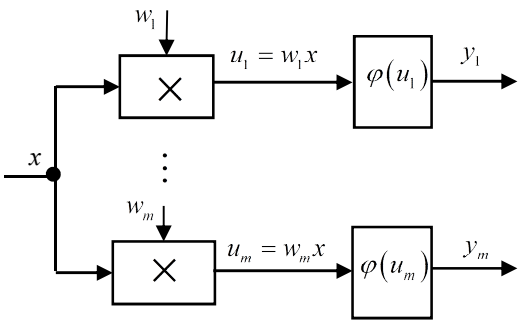

Рис. 8. Выходная звезда Гроссберга

Выходная звезда Гроссберга осуществляет преобразование [22]

$$y_i = \varphi(w_i x), \quad i = 1, 2, \ldots, m$$

с функцией активации (все векторы нормированы)

$$\varphi(u_i)=\begin{cases} u_i, & \text{если } -1\le u_i\le 1,\\ 1, & \text{если } 1<u_i,\\ -1, & \text{если } u_i<-1. \end{cases}$$

Обучение сети встречного распространения состоит из двух шагов. На первом шаге по рассмотренным ранее алгоритмам обучается слой Кохонена. Обученный слой Кохонена выдает вектор, у которого только один компонент отличен от нуля. На втором шаге производится обучение с учителем слоя Гроссберга. Коррекция весов слоя Гроссберга производится по формуле

$$w_{ij}^{(k+1)}=w_{ij}^{(k)}+\beta\left(y_j-w_{ij}^{(k)}\right)y_{ki},$$

где k — номер цикла обучения, w_{ij} — i-й вес нейрона j, y_{ki} — выход i-го нейрона слоя Кохонена (только для одного нейрона Кохонена он отличен от нуля), y_j — j-й компонент вектора желаемых выходов, β — коэффициент скорости обучения (рекомендуется [21] первоначально брать β равным ~0,1, а затем постепенно уменьшать в процессе обучения).

Так как вектор на выходе слоя Кохонена имеет только один единичный компонент, соответствующий нейрону-победителю, то фактически происходит уточнение только тех весов выходного слоя, которые связывают нейроны слоя Гроссберга с нейроном-победителем слоя Кохонена.

Если на обученную сеть подать вектор, не принадлежащий обучающей выборке, то сначала в слое Кохонена будет установлена его принадлежность определенному кластеру, после чего сигнал нейрона-победителя слоя Кохонена поступит на выходную звезду, на выходах которой сформируется выходной вектор, элементы которого равны координатам центроида (центра) обнаруженного кластера.

Рассмотренная сеть является *однонаправленной* сетью прямого распространения. *Двунаправленные сети встречного распространения* позволяют не только по входному вектору находить выходной вектор, но и по известному выходному вектору находить соответствующий входной вектор. Этими свойствами и обусловлен термин "встречное

распространение". Структура двунаправленной сети встречного распространения показана на рис. 9 [25].

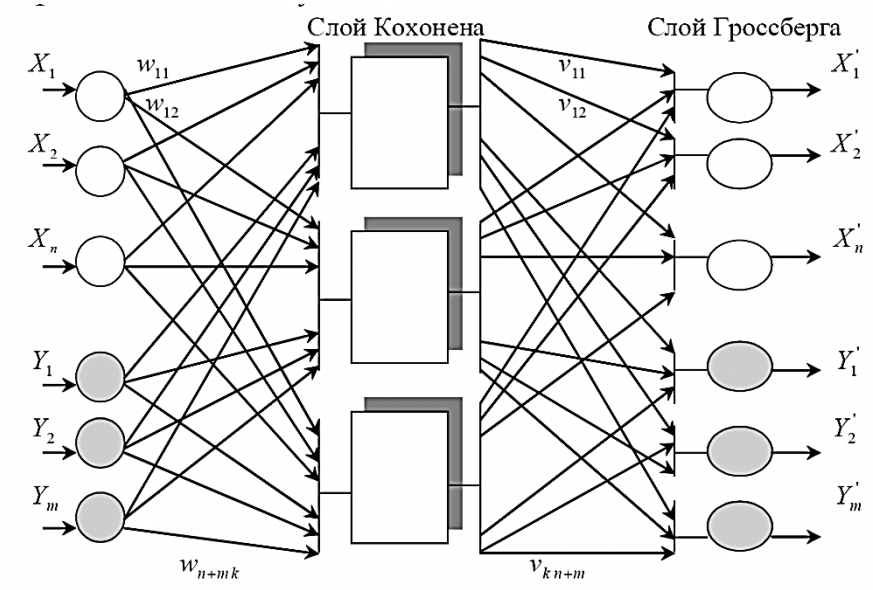

Рис. 9. Двунаправленная сеть встречного распространения

В процессе обучения векторы $\mathbf{x}$ и $\mathbf{y}$ используются и как входные векторы сети, и как целевые выходные векторы. Векторы $\mathbf{x}$ используется для обучения выходов $\mathbf{x}'$, а векторы $\mathbf{y}$ — для обучения выходов $\mathbf{y}'$ выходного слоя. Обучение сети производится точно так же, как и обучение однонаправленной сети встречного распространения. Формально только увеличилась размерность векторов. Если на вход обученной сети подать вектор $\mathbf{x}$, не использовавшийся при обучении, и соответствующий выходной вектор $\mathbf{y}$, то на выходе будут получены аппроксимации $\mathbf{x}'$ и $\mathbf{y}'$ этих векторов. Но такая задача вряд ли представляет практический интерес: известны входной и выходной векторы некоторого отображения. Если на вход подать только вектор $\mathbf{x}$, то на выходе будут получены аппроксимации $\mathbf{x}'$ и $\mathbf{y}'$. То есть будет найдено отображение $\mathbf{x}$ в $\mathbf{y}$. Если нам известен вектор $\mathbf{y}$, то подав только его на вход, получим $\mathbf{x}'$ и $\mathbf{y}'$. Получение $\mathbf{x}'$ означает реализацию обратного отображения $\mathbf{y}$ в $\mathbf{x}$.

Библиографический список

1. Осовский С. Нейронные сети для обработки информации. — М.: Финансы и статистика, 2002. — 244 с.

2. Хайкин С. Нейронные сети: полный курс. — М.: Вильямс, 2006. — 1104 с.

3. Кохонен Т. Самоорганизующиеся карты. — М.: БИНОМ. Лаборатория знаний, 2008. — 655 с.

4. Паклин Н. Б., Орешков В. И. Бизнес-аналитика: от данных к знаниям. — СПб.: Питер, 2013. — 704 с.

5. Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. — М.: ДМК Пресс, 2015. — 400 с.

6. Анализ данных и процессов / А. А. Барсегян, М. С. Куприянов, И. И. Холод, М. Д. Тесс, С. И. Елизаров. — СПб.: БХВ-Петербург, 2009. — 512 с.

7. Gersho A., Gray R. M. Vector Quantization and Signal Compression. — Springer, 1992. — 732 p.

8. Препарата Ф., Шеймос М. Вычислительная геометрия: Введение. — М.: Мир, 1989. — 478 с.

9. Beale M. H., Hagan M. T., Demuth H. B. Neural Network Toolbox. User's Guide. — Natick: Math Works, Inc., 2015. — 406 p.

10. Самоорганизующиеся карты — математический аппарат. [Электронный ресурс]. URL: http://basegroup.ru/community/articles/som (дата обращения: 20.07.2015).

11. Matsushita H., Nishio Y. Batch-Learning Self-Organizing Map with Weighted Connections avoiding false-neighbor effects // WCCI 2010 IEEE World Congress on Computational Intelligence July, 18-23, 2010 - CCIB, Barcelona, Spain. — P. 1075–1080.

12. Martinez M., Berkovich S., Schulten K. "Neural-gas" network for vector quantization and its application to time series prediction // IEEE Trans. on Neural Networks. – 1993. – 4. – P.558 – 569.

13. Sledge I. J., Keller J. M. Growing neural gas for temporal clustering // 19th International Conference on Pattern Recognition (ICPR'08), December 8-11, 2008, Tampa, Florida, USA. IEEE Computer Society 2008. — P. 1–4.

14. Горбаченко В. И. Вычислительная линейная алгебра с примерами на MATLAB. — СПБ.: БХВ-Петербург, 2011. — 320 с.

15. Akinduko A. A., Mirkes E. Initialization of Self-Organizing Maps: Principal Components Versus Random Initialization. A Case Study. [Электронный ресурс]. URL:http://www.math.le.ac.uk/people/ag153/homepage/PCA_SOM/AkindukoMirkes.pdf (дата обращения: 22.07.2015).

16. Attik M., Bougrain L., Alexandre F. Self-organizing Map Initialization // Artificial Neural Networks: Biological Inspirations – ICANN 2005: Lecture Notes in Computer Science. — 2005, Vol. 3696. — P. 357–362.

17. Bransdon C., Singleton A. D. Geocomputation: A Practical Primer. — SAGE Publications, 2015. — 392 p.

18. Ultsch A., Siemon H. P. Kohonen's Self Organizing Feature Maps for Exploratory Data Analysis // Proceedings of the International Neural Network Conference (INNC-90), Paris, France, July 9–13, 1990. — P. 305–308.

19. Sammon J., W. A nonlinear mapping for data structure analysis // IEEE Transactions on Computers, 1969, Vol. 18. — Issue 5. — P. 401–409.

20. Neural Network Design / M. N. Hagan, H. B. Demuth, M. H. Beale, O. De Jesús. — Martin Hagan, 2014. — 800 p.

21. Уоссермен Ф. Нейрокомпьютерная техника: Теория и практика. — М.: Мир, 1992. — 240 с.

22. Бодянский Е. В., Руденко О. Г. Искусственные нейронные сети: архитектура, обучение, применение. — Харьков: ТЕЛЕТЕХ, 2004. — 372 с.

23. Hecht-Nielsen R. Counterpropagation networks //Applied Optics, 1987, Vol. 26. — Issue 23. — 4979–4984.

24. Grossberg S. 1969. Some networks that can learn, remember and reproduce any number of complicated space-time patterns, I. // Journal of Mathematics and Mechanics. — 1969, Vol. 19. — No. 1. — P. 53–91.

25. Снитюк В. Е. Прогнозирование. Модели, методы, алгоритмы. — Киев: "Маклаут", 2008. — 364 с.