Министерство образования и науки Российской Федерации

Калужский филиал

федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

И.И. Кручинин
(к.т.н. доцент)

Лекция: «Нейросетевые классификаторы» по курсу «Введение в машинное обучение»

Калуга – 2018

Краткое содержание:

1. Искусственные нейронные сети.
2. Задача исключающего ИЛИ
3. Многослойные нейронные сети
4. Метод обратного распространения ошибок
5. Классификационные способности двухслойного персептрона
6. Трехслойный персептрон
7. Выбор градиентного метода оптимизации
8. Построение нейросетевого классификатора
9. Сети Кохонена. Модели конкурентного обучения
10. Самоорганизующие карты Кохонена
11. Гибридные сети встречного распространения
12. Многомерное шкалирование
13. Диаграмма Шепарда

Искусственные нейронные сети

Человеку и высшим животным буквально на каждом шагу приходится распознавать, принимать решения и обучаться. Нейросетевой подход возник из стремления понять, каким образом мозг решает столь сложные задачи, и реализовать эти принципы в автоматических устройствах. Пока искусственные нейронные сети (artificialneuralnetworks, ANN) являются лишь предельно упрощёнными аналогами естественных нейронных сетей. Нервные системы животных и человека гораздо сложнее тех устройств, которые можно создать с помощью современных технологий. Однако для успешного решения многих практических задач оказалось вполне достаточно «подсмотреть» лишь общие принципы функционирования нервной системы. Некоторые разновидности ANN представляют собой математические модели,

имеющие лишь отдалённое сходство с нейрофизиологией, что отнюдь не препятствует

их практическому применению.

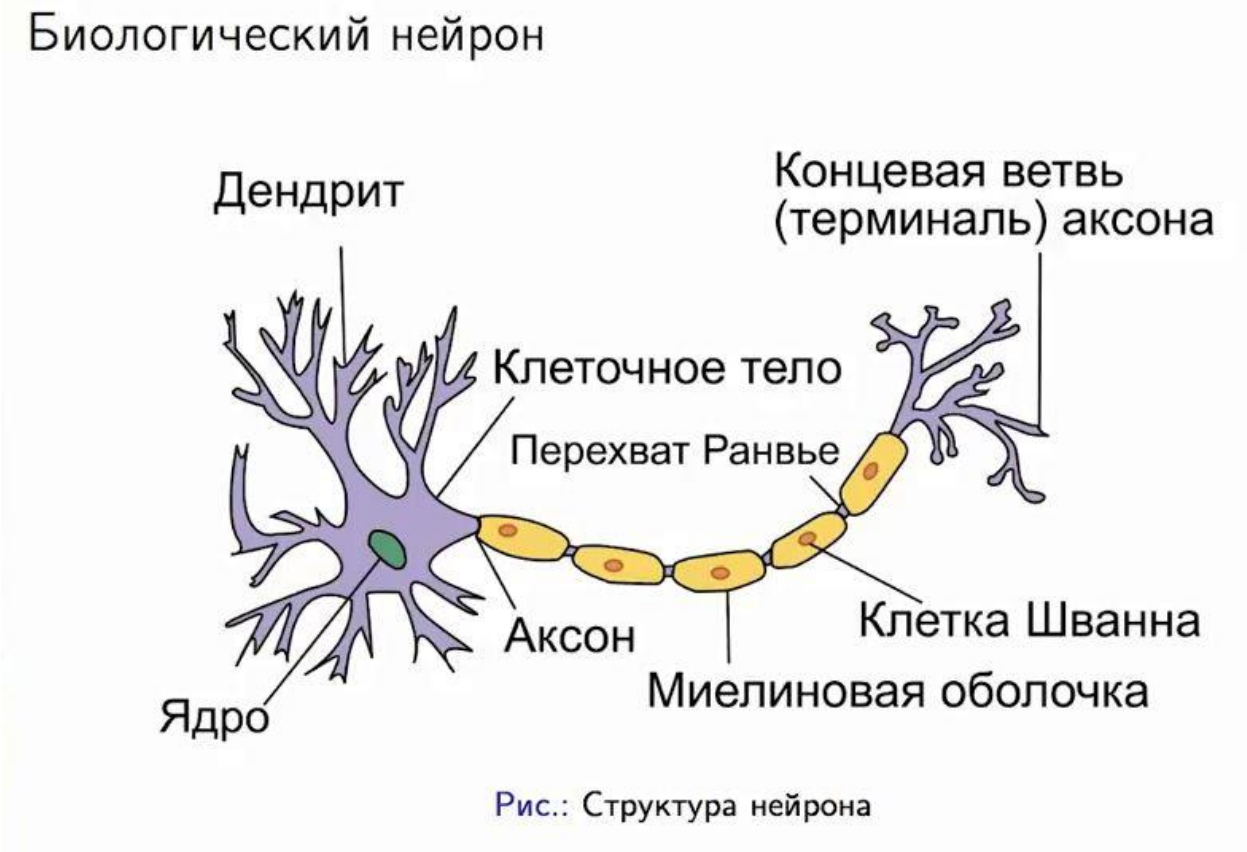


Рис.: Структура нейрона

Проблема полноты

Итак, отдельно взятый нейрон вида позволяет реализовать линейный классификатор или линейную регрессию. При решении практических задач линейность оказывается чрезмерно сильным ограничением. На ограниченность персептрона указывали Минский и Пайперт в своей знаменитой книге «Персептроны» . Следующий классический пример иллюстрирует невозможность нейронной реализации даже очень простых функций.

Задача «исключающего ИЛИ»

Легко построить нейроны, реализующие логические функции И, ИЛИ, НЕ от бинарных переменных x^1 и x^2, см. Рис. 18:

$$x^1 \vee x^2 = \left[x^1 + x^2 - \tfrac{1}{2} > 0\right];$$
$$x^1 \wedge x^2 = \left[x^1 + x^2 - \tfrac{3}{2} > 0\right];$$
$$\neg x^1 = \left[-x^1 + \tfrac{1}{2} > 0\right];$$

Однако функцию $x^1 \oplus x^2 = [x^1 =6\ x^2]$ — исключающее ИЛИ (exclusiveor, XOR) принципиально невозможно реализовать одним нейроном с двумя входами x^1 и x^2, поскольку множества нулей и единиц этой функции линейно неразделимы.

Возможны два пути решения этой проблемы, см. Рис 20.

Первый путь — пополнить состав признаков, подавая на вход нейрона нелинейные преобразования исходных признаков. В частности, если разрешить образовывать всевозможные произведения исходных признаков, то нейрон будет строить уже не линейную, а полиномиальную разделяющую поверхность. В случае исключающего ИЛИ достаточно добавить только один вход x^1x^2, чтобы в расширенном пространстве множества нулей и единиц оказались линейно разделимыми:

$$x^1 \oplus x^2 = \left[x^1 + x^2 - 2x^1x^2 - \tfrac{1}{2} > 0\right].$$

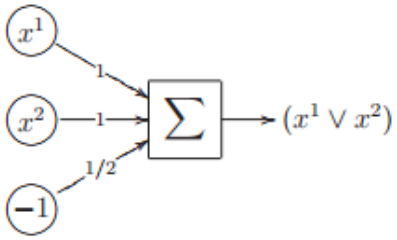


Рис. 18. Однослойный персептрон, реализующий операцию ИЛИ.

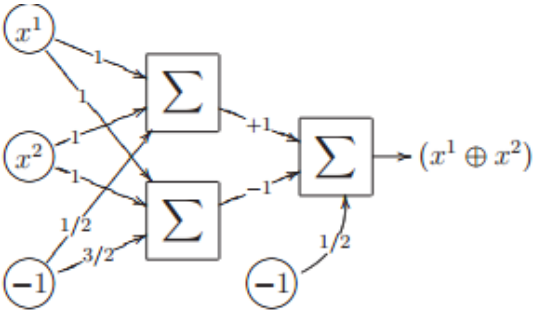


Рис. 19. Двухслойная сеть, реализующая операцию исключающего ИЛИ.

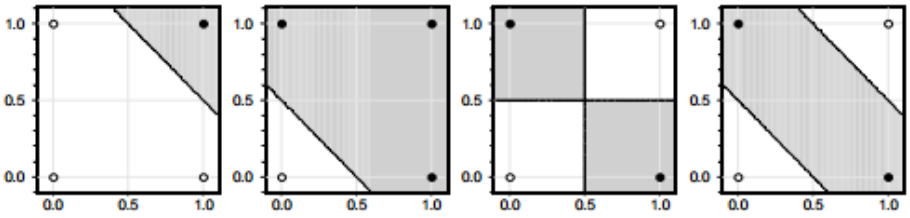


Рис. 20. Персептронная реализация элементарных логических функций. Слева направо: И, ИЛИ, XOR с помощью добавления признака x^1x^2, XOR с помощью двухслойной сети.

заключается в том, что подбор нужных нелинейных преобразований является нетривиальной задачей, которая для общего случая до сих пор остаётся нерешённой.

Второй путь — построить композицию из нескольких нейронов. Например, исключающее ИЛИ можно реализовать, подав выходы И-нейрона и ИЛИ-нейрона на вход ещё одному ИЛИ-нейрону, Рис 19:

$$x^1 \oplus x^2 = \left[(x^1 \vee x^2) - (x^1 \wedge x^2) - \tfrac{1}{2} > 0\right].$$

Дальнейшее обобщение этой идеи приводит к построению многослойных нейронных сетей, состоящих из огромного количества связанных нейронов и напоминающих естественные нейронные сети. Пример такой композиции показан на Рис. 21. Значения всех *n*признаков одновременно подаются на вход всех *H*нейронов первого слоя. Затем их выходные значения подаются на вход всех *M*нейронов следующего слоя. В данном случае этот слой является выходным — такая сеть

называется двухслойной.[1]В общем случае сеть может содержать произвольное число слоёв. Все слои, за исключением последнего, называются скрытыми (hiddenlayers).

Вычисление выходных значений сети может осуществляться с высокой степенью параллелизма, за число тактов, равное числу слоёв. Существуют эффективные аппаратные реализации нейронных сетей, в которых вычисления действительно происходят параллельно. Но пока на практике чаще используются программные реализации, выполненные на обычных последовательных компьютерах.

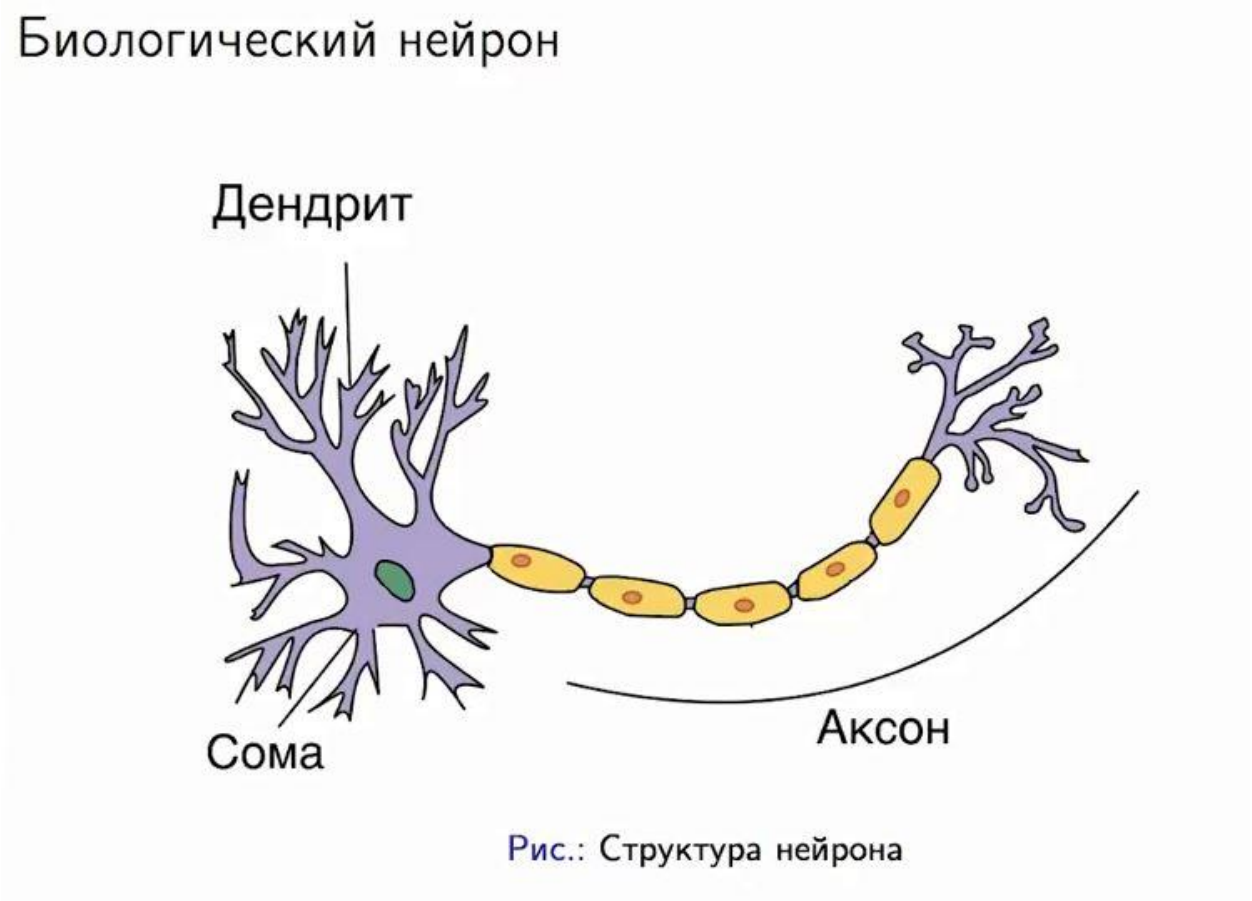


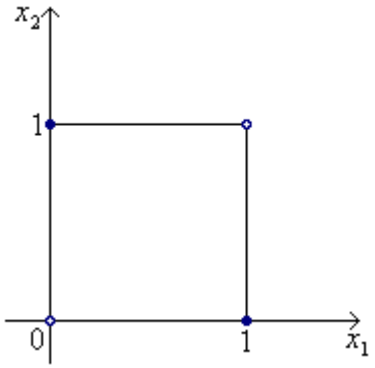
Рис.: Структура нейрона

Рассмотрим булеву функцию $xor(x_1, x_2)$ как некий *классификатор*. *Вектор* признаков имеет вид $x = (x_1, x_2)$. В данном случае имеется четыре прецедента и два класса. Напомним таблицу значений функции $xor(x_1, x_2)$.

№ прецедента	x_1	x_2	$xor(x_1, x_2)$	Класс

[1] Существует терминологическая путаница с подсчётом числа слоёв. Иногда такую сеть (видимо, глядя на картинку) называют трёхслойной, считая входы $x^0, x^1, \ldots, x^n$ особым, «распределительным» слоем. По делу, в счёт должны идти только слои, состоящие из суммирующих нейронов.

1	0	0	0	Ω_1
2	0	1	1	Ω_0
3	1	0	1	Ω_0
4	1	1	0	Ω_1



Как видно из рисунка тут нельзя построить разделяющую прямую, поскольку *выпуклые оболочки* точек, относящихся к первому классу и ко второму классу, пересекаются. Следовательно, и линейный *классификатор* построить нельзя. Попытаемся построить необходимый нелинейный *классификатор* как суперпозицию несколько линейных.

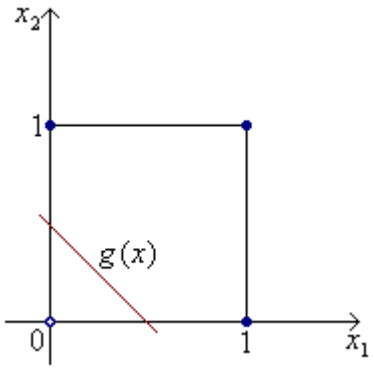
Рассмотрим две вспомогательные булевы функции $or(x_1, x_2)$ и $and(x_1, x_2)$. Напомним таблицы значений этих функций:

№ прецедента	x_1	x_2	$and(x_1, x_2)$	$or(x_1, x_2)$
1	0	0	0	0
2	0	1	0	1
3	1	0	0	1

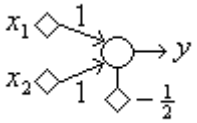
4	1	1	1	1

Построение линейного классификатора функции $or(x_1, x_2)$. Очевидно, что разделяющей *прямой* является линия:

$$x_1 + x_2 = \frac{1}{2}$$

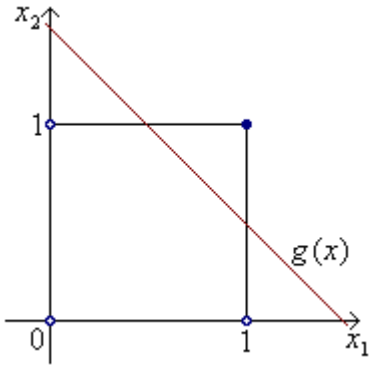


Соответствующий *перспептрон* имеет вид:

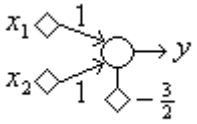


Построение линейного классификатора функции $and(x_1, x_2)$. Здесь также можно построить разделяющую прямую:

$$x_1 + x_2 = \frac{3}{2}$$



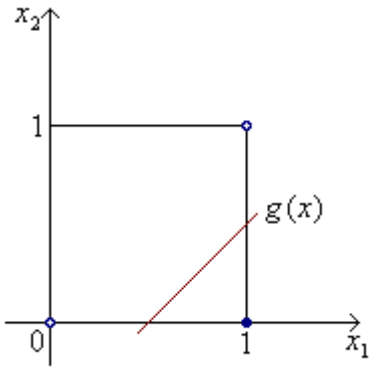
Соответствующий *персептрон* имеет вид:



Построение нелинейного классификатора функции $xor(x_1, x_2)$. Пусть на выходе персептрона для функции $or(x_1, x_2)$ - y_1, а на выходе персептрона для функции $and(x_1, x_2)$ – y_2. Посмотрим, какие значения принимает *вектор* (y_1, y_2).

Исходныевектора		OR	AND	XOR	
x_1	x_2	y_1	y_2		Класс
0	0	0	0	1	Ω_1
0	1	1	0	0	Ω_0
1	0	1	0	0	Ω_0

1	1	1	1	1	Ω_1



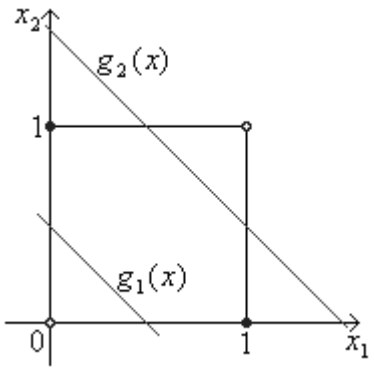
Обозначив классы как показано в таблице, получаем разделяющую прямую, изображенную на рисунке и соответствующий линейный *классификатор*:

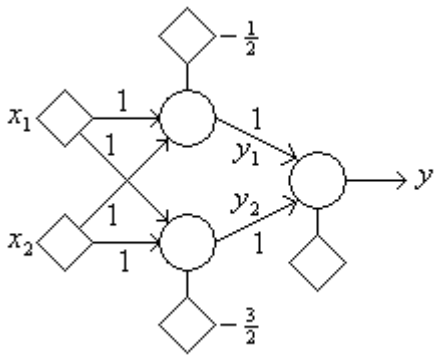
$$y_1 - y_2 = \frac{1}{2}$$

Учитывая вышеизложенное, получаем нелинейный *классификатор*, который задается через два линейных классификатора, как показано на рисунке слева:

$$x_1 + x_2 = \frac{1}{2} \text{ и } x_1 + x_2 = \frac{3}{2}$$

Соответствующий двухслойный *персептрон* изображен на рисунке справа.





Биологический нейрон

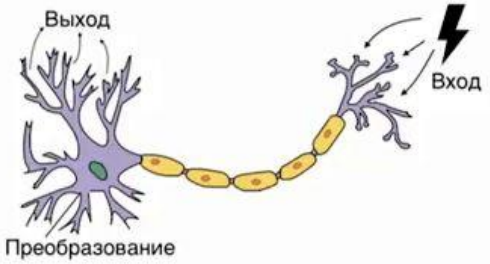


Рис.: Структура нейрона

Выходной сигнал посылается при достижении определенного уровня входного сигнала.
Модель:

$$y = \begin{cases} 1, & \sum_{i=1}^{N} x_i > b \\ 0, & \text{иначе} \end{cases} = \mathbb{I}[\sum_{i=1}^{N} x_i > b]$$

Вычислительные возможности нейронных сетей.

Возникает вопрос: любую ли функцию можно представить (хотя бы приближённо) с помощью нейронной сети? Следующие факты позволяют ответить на этот вопрос утвердительно.

1. Любая булева функция представима в виде двухслойной сети. Это тривиальное следствие нейронной представимости функций И, ИЛИ, НЕ и представимости произвольной булевой функции в виде дизъюнктивной нормальной формы.
2. Из простых геометрических соображений вытекает, что двухслойная сетьс пороговыми функциями активации позволяет выделить произвольный выпуклый многогранник в *n*-мерном пространстве признаков. Трёхслойная сеть позволяет вычислить любую конечную линейную комбинацию характеристических функций выпуклых многогранников, следовательно, аппроксимировать любые области с непрерывной границей, включая невыпуклые и даже неодносвязные, а также аппроксимировать любые непрерывные функции.
3. В 1900 году Гильберт предложил список из 23 нерешённых задач, которые,по его мнению, должны были стать вызовом для математиков XX века. Тринадцатая проблема заключалась в следующем: возможно ли произвольную непрерывную функцию *n*аргументов представить в виде суперпозиции функций меньшего числа аргументов. ОтветбылданА.Н.Колмогоровым в [14].

Теорема 6.1 (Колмогоров, 1957). Любая непрерывная функция *n*аргументов на единичном кубе $[0,1]^n$представима в виде суперпозиции непрерывных функций одного

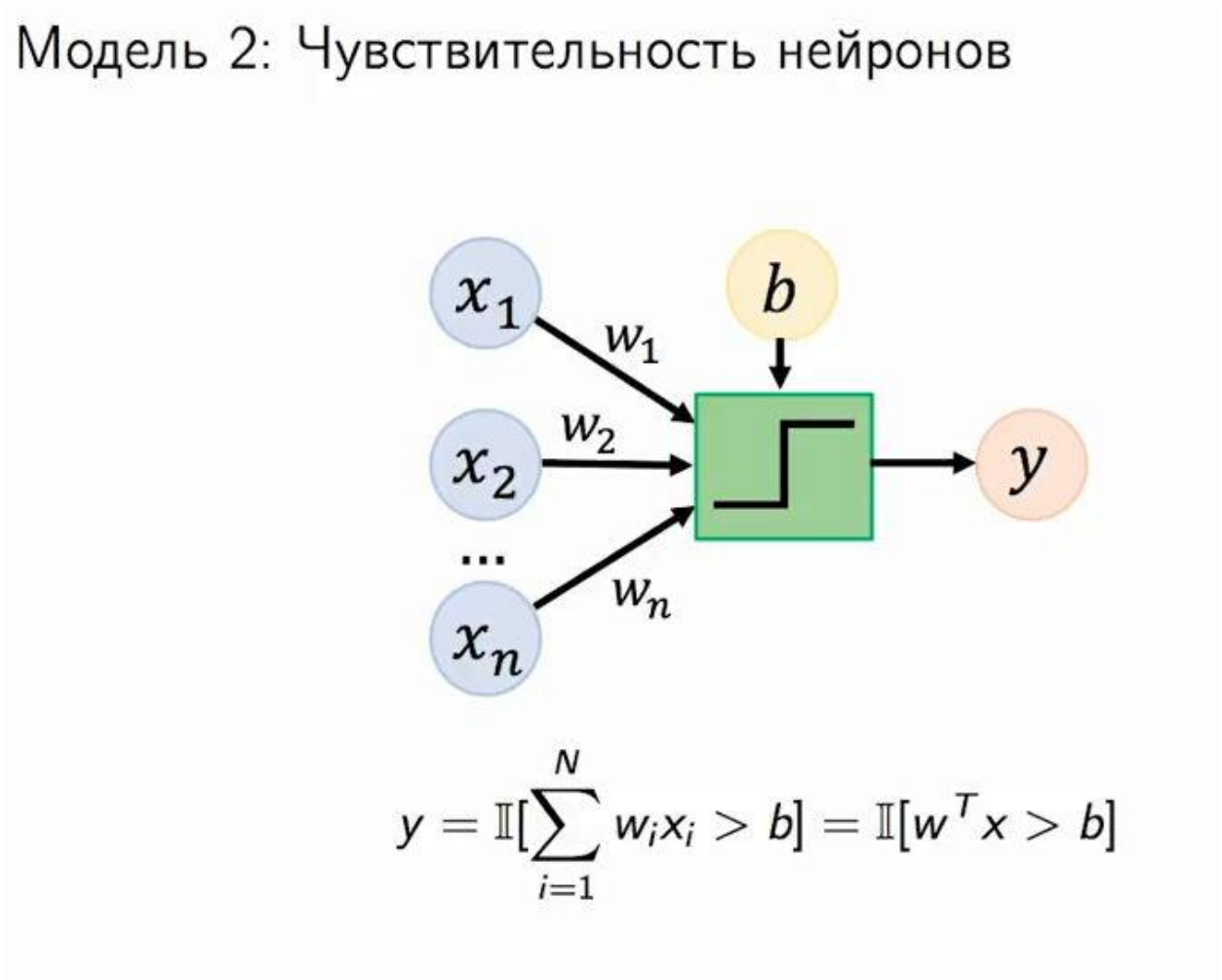
аргумента и операции сложения:

$$f(x^1, x^2, \ldots, x^n) = \sum_{k=1}^{2n+1} h_k \left(\sum_{i=1}^{n} \varphi_{ik}(x^i) \right),$$

где h_k, ϕ_{ik}— непрерывные функции, причём ϕ_{ik}не зависят от выбора *f*.

Нетрудно видеть, что записанное здесь выражение имеет структуру нейронной сети с одним скрытым слоем из $2n+1$ нейронов. Таким образом, двух слоёв уже достаточно, чтобы вычислять произвольные непрерывные функции, и не приближённо, а точно. К сожалению, представление Колмогорова не является персептроном: функции ϕ_{ik}не линейны, а функции h_kзависят от *f*, и в общем случае не являются дифференцируемыми.

4. Известна классическая теорема Вейерштрасса о том, что любую непрерывную функцию *n*переменных можно равномерно приблизить полиномом с любой степенью точности. Более общая теорема Стоуна утверждает, что любую непрерывную функцию на произвольном компакте *X*можно приблизить не только многочленом от исходных переменных, но и многочленом от любого конечного набора функций *F*, разделяющих точки.



Опр. 6.1. Набор функций F называется разделяющим точки множества X, если для любых различных $x, x' \in X$ существует функция $f \in F$ такая, что $f(x) \neq f(x')$.

Теорема 6.2 (Стоун, 1948). Пусть X — компактное пространство, $C(X)$ — алгебра непрерывных на X вещественных функций, F — кольцо в $C(X)$, содержащее константу ($1 \in F$) и разделяющее точки множества X. Тогда F плотно в $C(X)$.

На самом деле справедливо ещё более общее утверждение. Оказывается, вместо многочленов (суперпозиции операций сложения и умножения) можно пользоваться суперпозициями сложения и какой-нибудь (практически произвольной) непрерывной нелинейной функции одного аргумента [21]. Этот результат имеет прямое отношение к нейронным сетям, поскольку они строятся из операций сложения, умножения и нелинейных функций активации.

Опр. 6.2. Набор функций $F \subseteq C(X)$ называется замкнутым относительно функции ϕ: $\mathrm{R} \to \mathrm{R}$, если для любого $f \in F$ выполнено $\phi(f) \in F$.

Теорема 6.3 (Горбань, 1998). Пусть X — компактное пространство, $C(X)$ — алгебра непрерывных на X вещественных функций, F — линейное подпространство в $C(X)$, замкнутое относительно нелинейной непрерывной функции ϕ, содержащее константу ($1 \in F$) и разделяющее точки множества X. Тогда F плотно в $C(X)$.

Это интерпретируется как утверждение об универсальных аппроксимационных возможностях произвольной нелинейности: с помощью линейных операций и единственного нелинейного элемента ϕ можно получить устройство, вычисляющее любую непрерывную функцию с любой желаемой точностью. Однако данная теорема ничего не говорит о количестве слоёв нейронной сети (уровней вложенности суперпозиции) и о количестве нейронов, необходимых для аппроксимации произвольной функции.

Таким образом, нейронные сети являются универсальными аппроксиматорами функций. Возможности сети возрастают с увеличением числа слоёв и числа нейронов в них. Двух-трёх слоёв, как правило, достаточно для решения подавляющего большинства практических задач классификации, регрессии и прогнозирования.

Многослойные нейронные сети

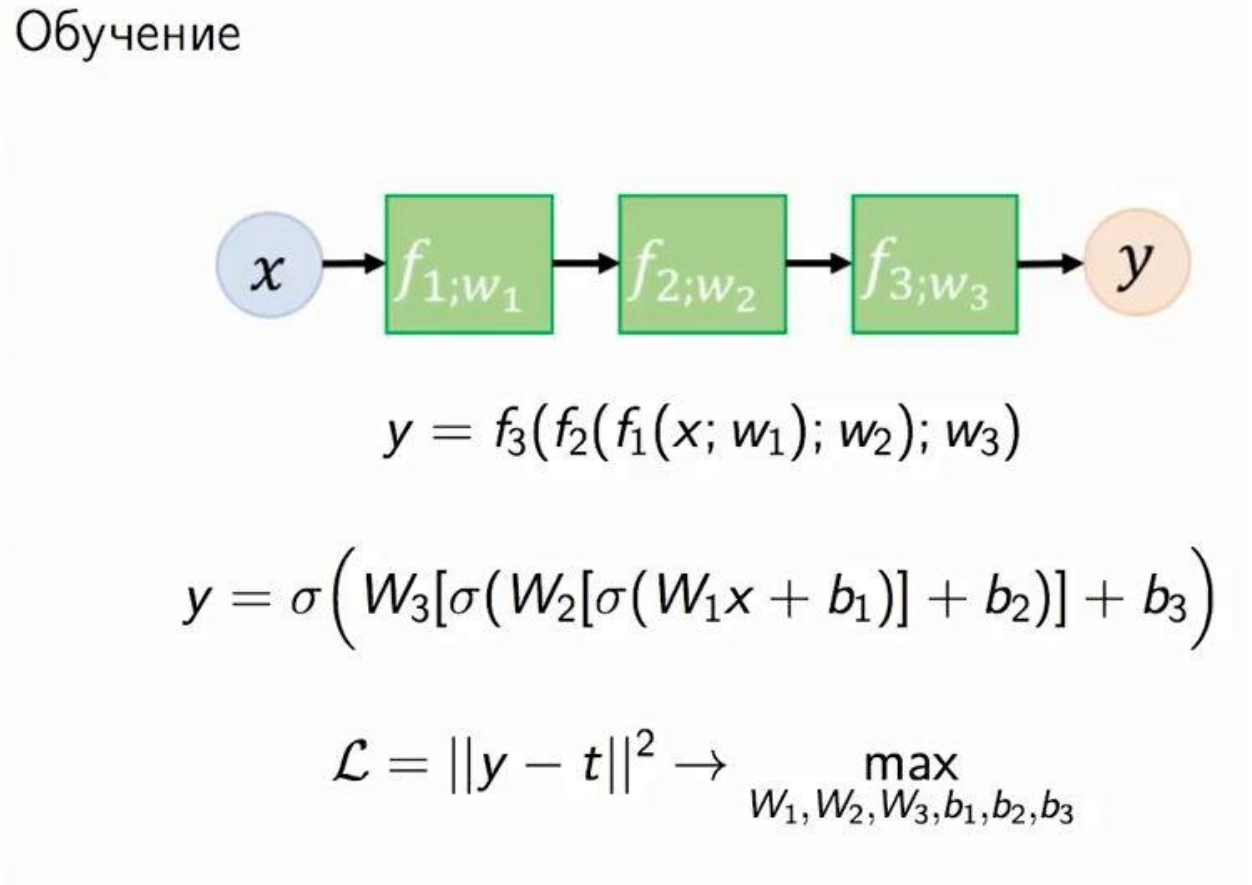
Многослойные сети, так же, как и однослойный персептрон (линейный классификатор), можно настраивать градиентными методами, несмотря на огромное количество весовых коэффициентов. В середине 80-х одновременно несколькими исследователями был предложен эффективный способ вычисления градиента, при котором каждый градиентный шаг выполняется за число операций, лишь немногим большее, чем при обычном вычислении сети на одном объекте. Это кажется удивительным — ведь количество операций, необходимых для вычисления градиента, обычно возрастает пропорционально размерности, то есть числу весовых коэффициентов. Здесь этого удаётся избежать благодаря аналитическому дифференцированию суперпозиции с сохранением необходимых промежуточных

величин. Метод получил название обратного распространения ошибок (errorback-

propagation) [58].

Метод обратного распространения ошибок

Рассмотрим многослойную сеть, в который каждый нейрон предыдущего слоя связан со всеми нейронами последующего слоя, Рис. 21. Такая сеть называется полносвязной. Для большей общности положим $X = R^n$, $Y = R^M$.



Введём следующие обозначения. Пусть выходной слой состоит из Mнейронов с функциями активации σ_mи выходами a^m, $m = 1,...,M$. Перед ним находится скрытый слой из Hнейронов с функциями активации σ_hи выходами u^h, $h = 1,...,H$.

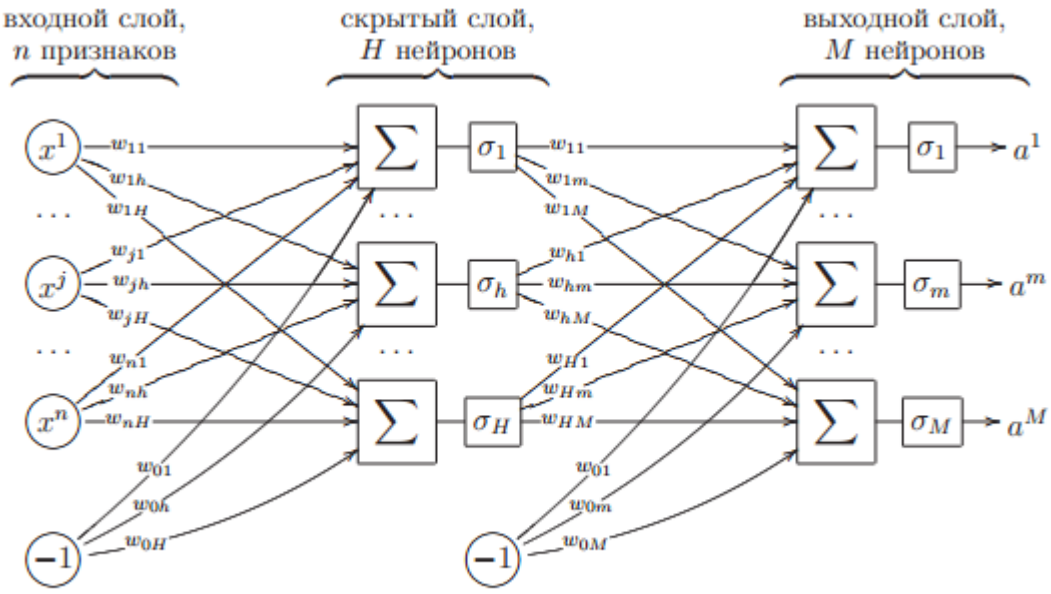


Рис. 21. Многослойная сеть с одним скрытым слоем.

Веса синаптических связей между h-м нейроном скрытого слоя и m-м нейроном выходного слоя будем обозначать через w_{hm}. Перед этим слоем может находиться либо входной слой признаков (называемый также распределительным слоем), либо ещё один скрытый слой с выходами v^j, $j = 1,...,J$ и синаптическими весами w_{jh}. В общем случае число слоёв может быть произвольным. Если сеть двухслойная, то v^j есть просто j-й признак: $v^j(x) \equiv f_j(x) \equiv x^j$, и $J = n$. Обозначим через w вектор всех синаптических весов сети.

Выходные значения сети на объекте x_i вычисляются как суперпозиция:

$$a^m(x_i) = \sigma_m\left(\sum_{h=0}^{H} w_{hm} u^h(x_i)\right); \qquad u^h(x_i) = \sigma_h\left(\sum_{j=0}^{J} w_{jh} v^j(x_i)\right). \tag{6.1}$$

Зафиксируем объект x_i и запишем функционал среднеквадратичной ошибки (для других функций потерь выкладки могут быть проделаны аналогично):

$$Q(w) = \frac{1}{2}\sum_{m=1}^{M}\left(a^m(x_i) - y_i^m\right)^2. \tag{6.2}$$

В дальнейшем нам понадобятся частные производные Q по выходам нейронов. Выпишем их сначала для выходного слоя:

$$\frac{\partial Q(w)}{\partial a^m} = a^m(x_i) - y_i^m = \varepsilon_i^m.$$

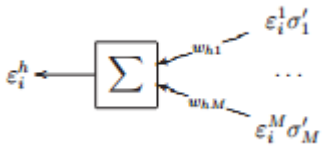
Оказывается, частная производная Q по a^m равна величине ошибки ε_i^m на объекте x_i. Теперь выпишем частные производные по выходам скрытого слоя:

$$\sigma'_m = \sigma_m(1-\sigma_m) = a^m(x_i)\big(1-a^m(x_i)\big)$$

$$\frac{\partial Q(w)}{\partial u^h} = \sum_{m=1}^{M}\big(a^m(x_i)-y_i^m\big)\sigma'_m w_{hm} = \sum_{m=1}^{M}\varepsilon_i^m\sigma'_m w_{hm} = \varepsilon_i^h .$$

Эту величину, по аналогии с ε_i^m, будем называть ошибкой сети на скрытом слое и обозначать через ε_i^h. Через σ'_m обозначена производная функции активации, вычисленная при том же значении аргумента, что и в (6.1). Если используется сигмоидная функция активации, то для эффективного вычисления производной можно воспользоваться формулой.

Заметим, что ε_i^h вычисляется по ε_i^m, если запустить сеть «задом наперёд», подав на выходы нейронов скрытого слоя значения $\varepsilon^m{}_i\sigma_m'$, а результат ε^h получив на входе. При этом входной вектор скалярно умножается на вектор весов w_{hm}, находящихся справа от нейрона, а не слева, как при прямом вычислении (отсюда и название алгоритма — обратное распространение ошибок):

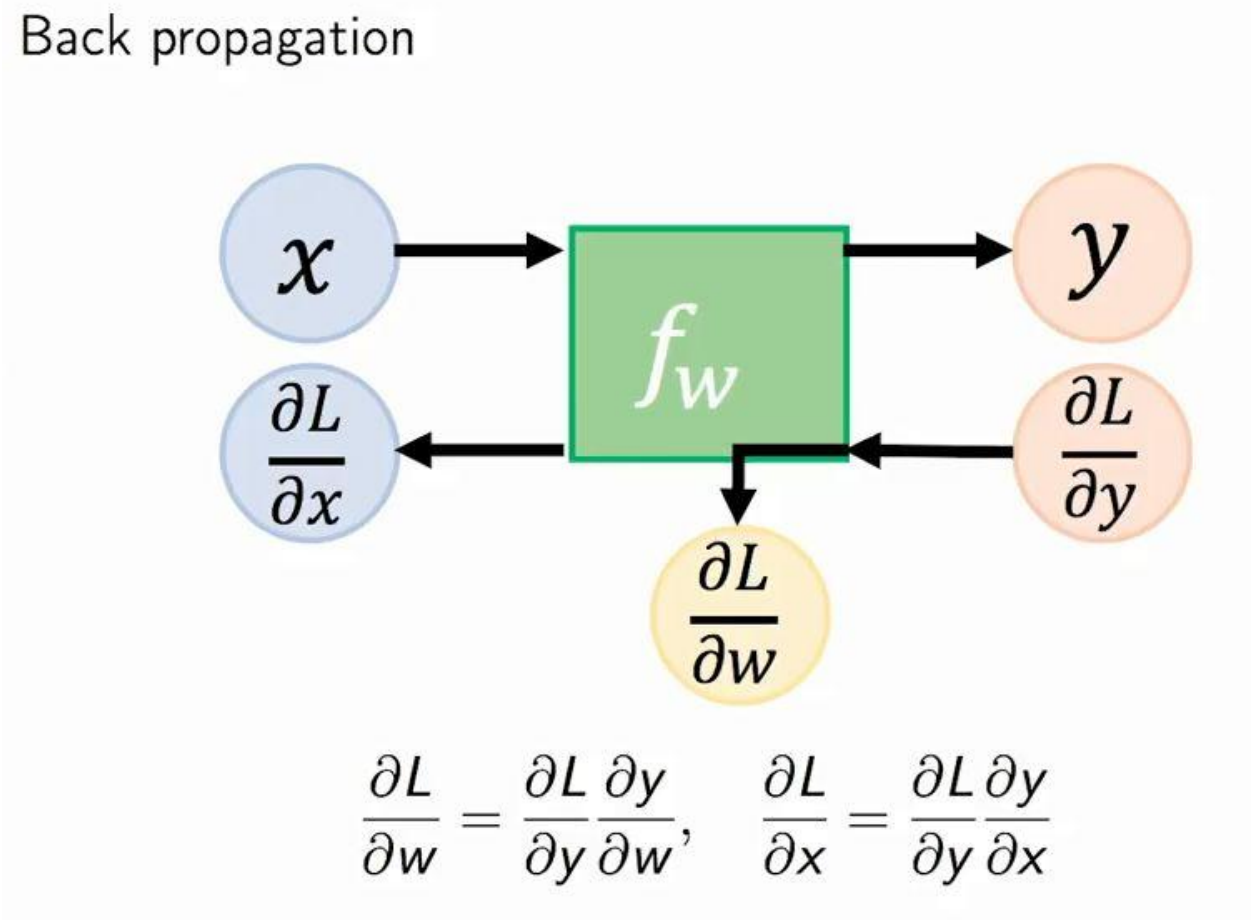


Имея частные производные по a^m и u^h, легко выписать градиент Q по весам:

$$\frac{\partial Q(w)}{\partial w_{hm}} = \frac{\partial Q(w)}{\partial a^m}\frac{\partial a^m}{\partial w_{hm}} = \varepsilon_i^m\sigma'_m u^h(x_i), \qquad m=1,\dots,M, \quad h=0,\dots,H; \tag{6.3}$$

$$\frac{\partial Q(w)}{\partial w_{jh}} = \frac{\partial Q(w)}{\partial u^h}\frac{\partial u^h}{\partial w_{jh}} = \varepsilon_i^h\sigma'_h v^j(x_i), \qquad h=1,\dots,H, \quad j=0,\dots,J; \tag{6.4}$$

и так далее для каждого слоя. Если слоёв больше двух, то остальные частные производные вычисляются аналогично — обратным ходом по слоям сети справа налево.



Достоинства метода обратного распространения.

- Достаточно высокая эффективность. В случае двухслойной сети прямой ход, обратный ход и вычисления градиента требуют порядка $O(Hn+HM)$ операций. • Через каждый нейрон проходит информация только о связных с ним нейронах. Поэтому back-propagation легко реализуется на вычислительных устройствах с параллельной архитектурой.
- Высокая степень общности. Алгоритм легко записать для произвольного числа слоёв, произвольной размерности выходов и выходов, произвольной функции потерь и произвольных функций активации, возможно, различных у разных нейронов. Кроме того, back-propagation можно применять совместно с различными градиентными методами оптимизации: методом скорейшего спуска, сопряженных градиентов, Ньютона-Рафсона и др.

Недостатки метода обратного распространения.

- Метод наследует известные недостатки градиентной настройки весов в однослойном персептроне. Здесь также возникают проблемы медленной сходимости

Алгоритм 6.1. Обучение двухслойной сети методом back-propagation — обратного распространения ошибки

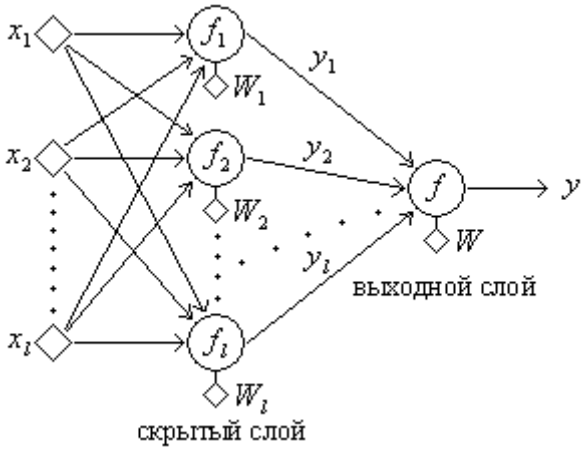
Вход:
$X^\ell = (x_i, y_i)_{i=1}^\ell$ — обучающая выборка, $x_i \in \mathbb{R}^n$, $y_i \in \mathbb{R}^M$;
H — число нейронов в скрытом слое;
η — темп обучения;
Выход:
синаптические веса w_{jh}, w_{hm};

1: инициализировать веса небольшими случайными значениями:
$w_{jh} := \text{random}\left(-\frac{1}{2n}, \frac{1}{2n}\right)$;
$w_{hm} := \text{random}\left(-\frac{1}{2H}, \frac{1}{2H}\right)$;
2: **повторять**
3: выбрать прецедент $(x_i, y_i) \in X^\ell$ случайным образом;
4: прямой ход:
$u_i^h := \sigma_h\left(\sum_{j=0}^J w_{jh} x_i^j\right)$, для всех $h = 1, \ldots, H$;
$a_i^m := \sigma_m\left(\sum_{h=0}^H w_{hm} u_i^h\right)$, для всех $m = 1, \ldots, M$;
$\varepsilon_i^m := a_i^m - y_i^m$, для всех $m = 1, \ldots, M$;
$Q_i := \sum_{m=1}^M (\varepsilon_i^m)^2$;
5: обратный ход:
$\varepsilon_i^h := \sum_{m=1}^M \varepsilon_i^m \sigma'_m w_{hm}$, для всех $h = 1, \ldots, H$;
6: градиентный шаг:
$w_{hm} := w_{hm} - \eta \varepsilon_i^m \sigma'_m u_i^h$, для всех $h = 0, \ldots, H$, $m = 1, \ldots, M$;
$w_{jh} := w_{jh} - \eta \varepsilon_i^h \sigma'_h x_i^j$, для всех $j = 0, \ldots, n$, $h = 1, \ldots, H$;
7: $Q := \frac{\ell-1}{\ell} Q + \frac{1}{\ell} Q_i$;
8: **пока** Q не стабилизируется;

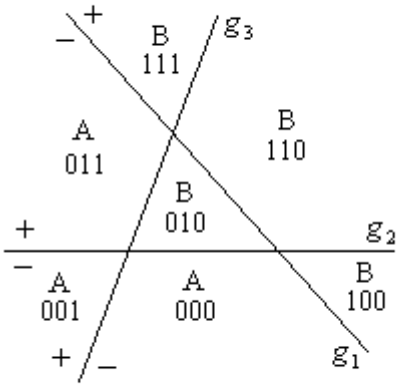
или расходимости, «застревания» в локальных минимумах функционала Q, переобучения и паралича. Причём парализоваться могут отдельные связи, нейроны, или вся сеть в целом.

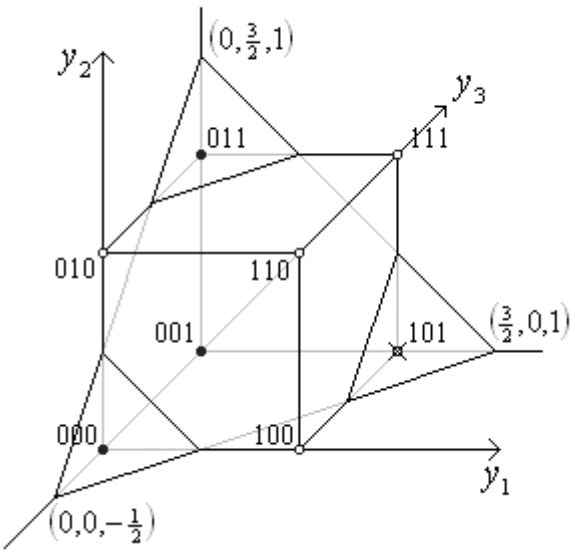
- Приходится заранее фиксировать число нейронов скрытого слоя H. В то же время, это критичный параметр сложности сети, от которого может существенно зависеть качество обучения и скорость сходимости.

Классификационные способности двухслойного персептрона



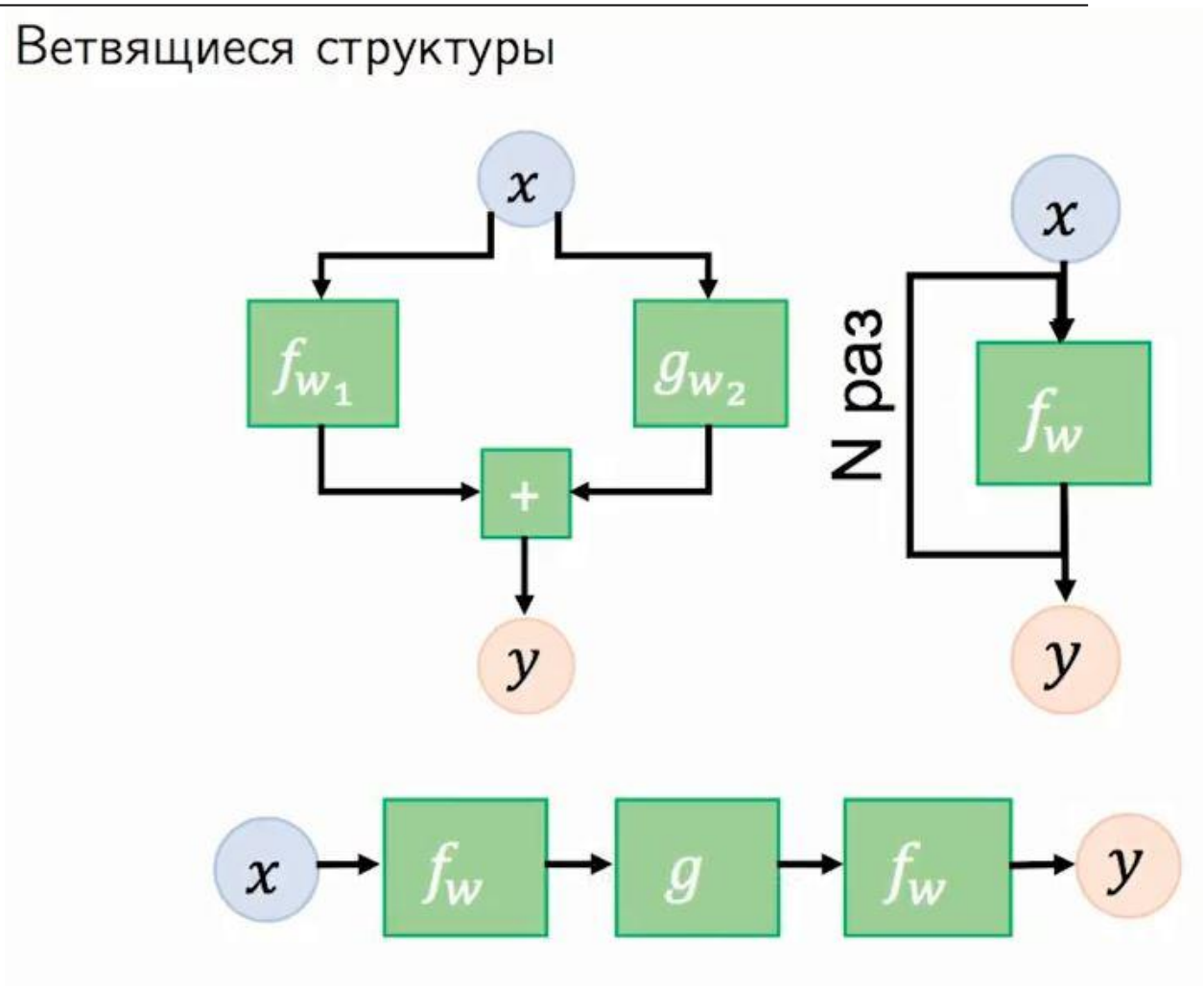
Рассмотрим общий случай двухслойного персептрона. Пусть $x \in R^l$ и в скрытом слое p нейронов. Скрытый слой нейронов отображает R^l в $H_p \in R^p$, где $H_p = \{(y_1, y_2, \ldots, y_p) \in R_p,\ y_i \in [0,1],\ 1 \le i \le p\}$ *– гиперкуб*. Другими словами, каждый *нейрон* задает *гиперплоскость*, которая разделяет *пространство* пополам, т.е. скрытый слой нейронов делит *пространство* R_l на *полиэдры*. Все вектора из каждого *полиэдра* отображаются в вершину p -мерного единичного куба. Выходной *нейрон* разделяет вектора в классах, описанных полиэдрами, т.е. производит *сечение* гиперкуба, полученного в скрытом слое.





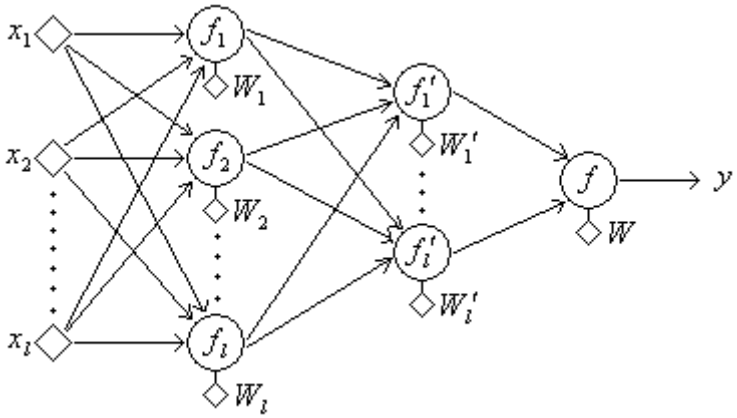
Пример. Рассмотрим нейронную *сеть* с двумя входами $(l = 2)$ и тремя

нейронами $(k = 3)$. Тогда *пространство* $R^l = R^2$. Пусть первый слой нейронов задает *разбиение* признакового пространства (плоскости) как показано на рисунке. В каждом многоугольнике (возможно, бесконечном) все точки соответствуют одному классу (A или B). При этом в каждом многоугольнике знаки линейных функционалов g_1, g_2, g_3 остаются постоянными. Следовательно, с каждым многоугольником связано определенное *значение* вектора выходов нейронов первого слоя, причем для разных многоугольников эти значения различны. Поскольку значениями *компонент* этого вектора являются 0 либо 1, получаем, что каждому многоугольнику соответствует некоторая *вершина* единичного куба H^3 в пространстве R^3. При этом каждой вершине куба сопоставлен один *класс* A или B. На рисунке изображен единичный куб H^3, у которого закрашенные вершины относятся к классу A, а не закрашенные – к классу B. Задача нейрона второго слоя состоит в разделении вершин этого куба. Нетрудно видеть, что в нашем примере *плоскость* $y_1 + y_2 - y_3 = \frac{1}{2}$ является разделяющей для куба H^3. Она и задает параметры нейрона второго слоя. Заметим, что *вершина* $(1, 0, 1)$ в кубе не загружена, т.е. в нее не отображается ни один многоугольник.



Трехслойный персептрон

Внешний (*выходной) нейрон* реализует лишь одну *гиперплоскость*. Очевидно, что одна разделяющая *гиперплоскость* не всегда может обеспечить желаемое разделение вершин гиперкуба. Например, если два конца одной его главной диагонали относятся к классу А, а два конца другой диагонали – к классу В. С аналогичной ситуацией мы уже сталкивались в задаче *исключающего или*. Попробуемввестищеодинслойнейронов.



Утверждение. *Трехслойная нейронная сеть позволяет описать любые разделения объединений полиэдров*.

Доказательство. Рассмотрим первый слой из p нейронов. На первом формируются гиперплоскости, т.к. строится полиэдральное *разбиение* пространства гиперплоскостями. Очевидно, что для заданного конечного *множества* прецедентов всегда можно построить *разбиение* пространства признаков на *полиэдры* такое, что ни в каком *полиэдре* не окажется пары точек из разных классов. Как было показано выше, первый слой отображает *полиэдры* в вершины p -мерного единичного гиперкуба. Поскольку с каждым полиэдром связаны образы одного класса, то и с каждой вершиной гиперкуба связан лишь один *класс*.

Каждый *нейрон* второго слоя описывает *сечение* полученного гиперкуба. Выберем в качестве таких сечений гиперплоскости, отсекающие ровно одну вершину гиперкуба. Поскольку число вершин в гиперкубе равно 2^p, число нейронов второго слоя также равно 2^p. Таким образом, *выход* нейронов второго слоя имеет следующий вид. Это *вектор* размерности 2^p, у которого всегда лишь одно *значение* равно 1, а остальные равны нулю. Назовем нейроны второго слоя нейронами класса A или B в соответствии с классом вершины гиперкуба, которую отсекает этот *нейрон*. Теперь становится понятно, каким образом строить третий слой нейронной сети. Нужно в выходном нейроне третьего слоя реализовать оператор логического сложения выходов нейронов второго слоя, относящихся к классу A. Таким образом разделяющая *гиперплоскость выходного нейрона* задается уравнением:

$$c_1 z_1, c_2 z_2, \ldots, c_k z_k = \tfrac{1}{2}, \text{ где } k = 2^p, \text{ а } c_i = \begin{cases} 1 & \text{если нейрон } i \text{ относится к классу } A \\ 0 & \text{в противном случае} \end{cases}$$

Таким образом, можно построить трехслойный *персептрон* следующим образом. Нейроны первого слоя разделяют *пространство признаков* на *полиэдры* одного класса и отображают их в вершины гиперкуба. Нейроны второго слоя отсекают вершины гиперкуба. *Нейрон* третьего слоя собственно осуществляет классификацию через оператор логического сложения. Тем самым утверждение доказано.

Рассмотрим, как строится уравнение гиперплоскости, отсекающей вершину p-мерного единичного гиперкуба. Диагональ куба имеет длину $\sqrt{p}$. Длины диагоналей $(p-1)$-мерных единичных гиперкубов, являющихся боковыми гранями p-мерного куба, равны $\sqrt{p-1}$. Центр куба находится в точке $\left(\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2}\right)$. *Расстояние* от центра куба до любой вершины равно $\frac{\sqrt{p}}{2}$. *Плоскость*проводим перпендикулярно главной диагонали куба, инцидентной вершине, которую надо отсечь, так, чтобы *расстояние* от этой вершины до секущей плоскости было равно $\frac{\sqrt{p}-\sqrt{p-1}}{2}$, причем данная точка должна находиться на диагонали куба, проведенной к отсекаемой вершине.

Пусть V – отделяемая *вершина*, $\overline{V}$ – диагонально противоположная *вершина* ($\overline{V} = E - V$, где E обозначает p-мерный *вектор*, состоящий из единиц). Следовательно, $W = V - \overline{V}$ – *направляющий вектор* разделяющей гиперплоскости. Тогда *гиперплоскость* проходит через точку:

$$U = \overline{V} + (V - \overline{V}) \cdot \frac{\sqrt{p} - \sqrt{p-1}}{2\sqrt{p}}$$

Обозначим:

$$\gamma = \frac{\sqrt{p} - \sqrt{p-1}}{2\sqrt{p}} = \frac{1}{2} \cdot \left(1 + \sqrt{1 - \frac{1}{p}}\right)$$

Тогда

$$U = \overline{V} + (V - \overline{V}) \cdot \gamma$$

и уравнение гиперплоскости запишется в виде: $((z - U), W) > 0$.

Эвристики для улучшения сходимости

Для улучшения сходимости и качества градиентного обучения применяются все те же приёмы, которые рекомендовались для обучения однослойного персептрона методом стохастического градиента. К ним добавляется ряд новых рекомендаций, связанных с многослойностью.

Выбор начального приближения. Для предотвращения паралича синаптические веса должны инициализироваться небольшими по модулю значениями. В Алгоритме 6.1 на шаге 1 веса инициализируются случайными значениями из отрезка $\left[-\frac{1}{2k}, \frac{1}{2k}\right]$, где k— число нейронов в том слое, из которого выходит данный синапс.

В этом случае (и при условии, что все признаки нормализованы) значения скалярных произведений гарантированно попадают в «рабочую зону» функций активации, представленных на Рис. 9.

Существует и более тонкий способ формирования начального приближения. Идея заключается в том, чтобы сначала настроить нейроны первого слоя по отдельности, как *Н*однослойных персептронов. Затем по-отдельности настраиваются нейроны второго слоя, которым на вход подаётся вектор выходных значений первого слоя. Чтобы сеть не получилась вырожденной, нейроны первого слоя должны быть существенно различными. Ещё лучше, если они будут хоть как-то приближать целевую зависимость, тогда второму слою останется только усреднить результаты первого слоя, сгладив ошибки некоторых нейронов[2]. Добиться этого совсем несложно, если обучать нейроны первого слоя на различных случайных подвыборках, либо подавать им на вход различные случайные подмножества признаков. Отметим, что при формировании начального приближения не требуется особая точность настройки, поэтому отдельные нейроны можно обучать простейшими градиентными методами.

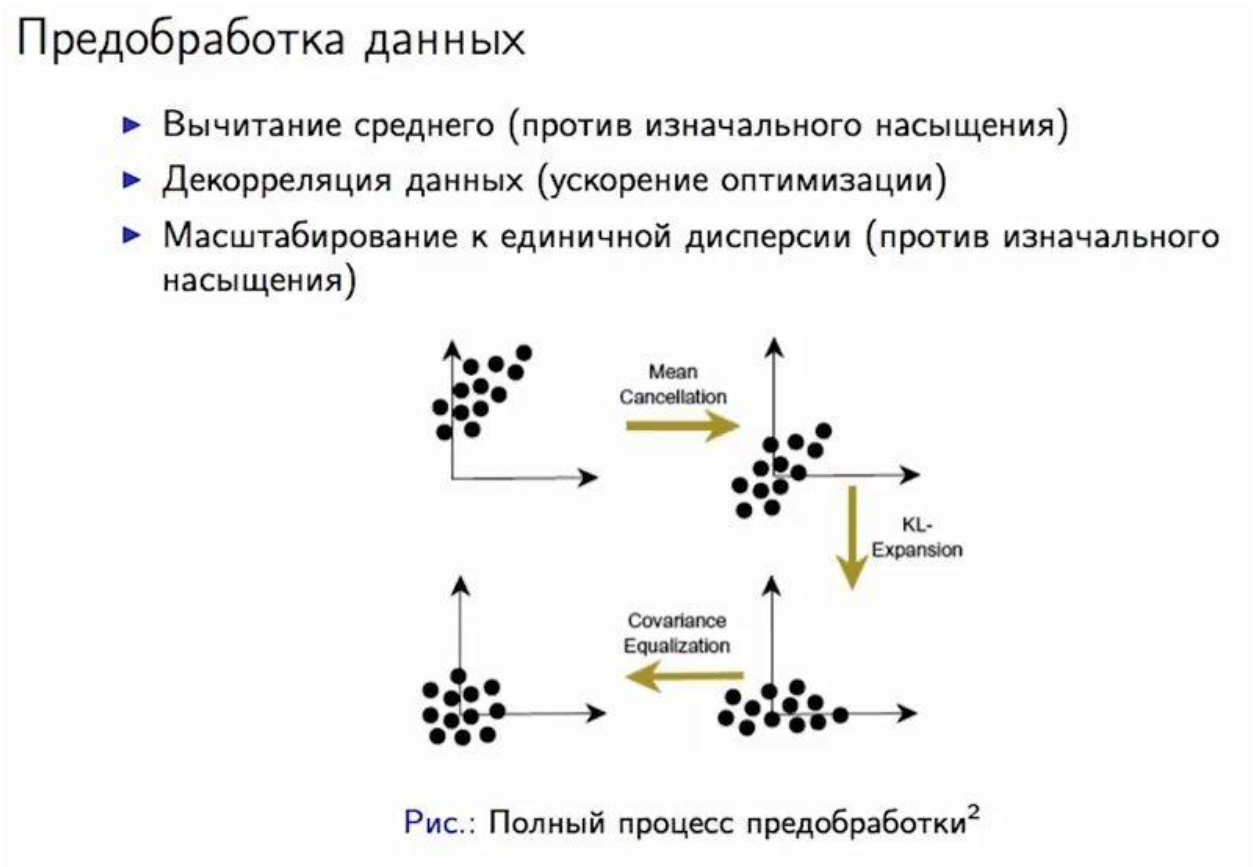


Рис.: Полный процесс предобработки[2]

Выбор градиентного метода оптимизации. К сожалению, градиентные методы первого порядка сходятся довольно медленно, и потому редко применяются на практике. Ньютоновские методы второго порядка также непрактичны, но по другой причине —

[2] Фактически, такая двухслойная сеть является простейшей алгоритмической композицией. Нейроны первого скрытого слоя играют роль базовых алгоритмов, нейроны второго слоя реализуют корректирующую операцию. Обучение первого слоя по случайным подвыборкам с последующим взвешенным усреднением во втором слое в точности соответствует методу бэггинга (bagging), см. раздел ??. Композициюобщеговидарассматриваются в главе ??.

они требуют вычисления матрицы вторых производных функционала $Q(w)$, имеющей слишком большой размер. Метод сопряжённых градиентов этого не требует, однако применять его непосредственно нельзя, так как он существенно опирается на предположение неизменности функционал $Q(w)$, а в методе стохастического градиента функционал меняется при предъявлении каждого нового объекта. Необходимы специальные ухищрения, чтобы приспособить стандартные методы оптимизации для настройки нейронных сетей. Даются следующие рекомендации.

1. Если обучающая выборка имеет большой объём (порядка нескольких сотенобъектов и более), или если решается задача классификации, то можно использовать метод стохастического градиента с адаптивным шагом.

2. Диагональный метод Левенберга–Марквардта сходится в несколько раз быстрее. В этом методе величина шага вычисляется индивидуально для каждого весового коэффициента, при этом используется только один диагональный элемент матрицы вторых производных:

$$\eta_{jh} = \frac{\eta}{\frac{\partial^2 Q}{\partial w_{jh}^2} + \mu},$$

где ηостаётся глобальным параметром темпа обучения, μ — новый параметр, предотвращающий обнуление знаменателя, и, соответственно, неограниченное увеличение шага. Отношение η/μ есть темп обучения на ровных участках функционала $Q(w)$, где вторая производная обращается в нуль. Диагональный элемент матрицы вторых производных вычисляется с помощью специального варианта back-propagation.

3. Если обучающая выборка имеет небольшой объём, или если решается задачарегрессии, то лучше использовать адаптированные варианты метода сопряжённых градиентов. Адаптация заключается в том, что объекты предъявляются не по одному, а пакетами (batchlearning). Состав пакета может формироваться случайным образом. Для каждого пакета минимизируемый функционал остаётся фиксированным, что позволяет применить метод сопряжённых градиентов.

Оптимизация структуры сети

Выбор структуры сети, то есть числа слоёв, числа нейронов и числа связей для каждого нейрона, является, пожалуй, наиболее сложной проблемой. Существуют различные стратегии поиска оптимальной структуры сети: постепенное наращивание, построение заведомо слишком сложной сети с последующим упрощением, поочерёдное наращивание и упрощение.

Проблема выбора структуры тесно связана с проблемами недообучения и переобучения. Слишком простые сети не способны адекватно моделировать целевые зависимости в реальных задачах. Слишком сложные сети имеют избыточное число свободных параметров, которые в процессе обучения настраиваются не только на восстановление целевой зависимости, но и на воспроизведение шума.

Выбор числа слоёв. Если в конкретной задаче гипотеза о линейной разделимости классов выглядит правдоподобно, то можно ограничиться однослойным персептроном. Двухслойные сети позволяют представлять извилистые нелинейные границы, и в большинстве случаев этого хватает. Трёхслойными сетями имеет смысл пользоваться для представления сложных многосвязных областей. Чем больше слоёв, тем более богатый класс функций реализует сеть, но тем хуже сходятся градиентные методы, и тем труднее её обучить.

SGD / Mini-batch SGD

- Какие функции оптимизируем?
- Большие суммы функций: $J(\theta) = \sum_{i=1}^{N} J_i(\theta)$
- Формула пересчета: $\theta_t = \theta_{t-1} - \eta \nabla_\theta J_i(\theta_{t-1})$
- Mini-batch SGD: $\theta_t = \theta_{t-1} - \eta \sum_{i \in \{i_1, i_2, \ldots, i_k\}} \nabla_\theta J_i(\theta_{t-1})$

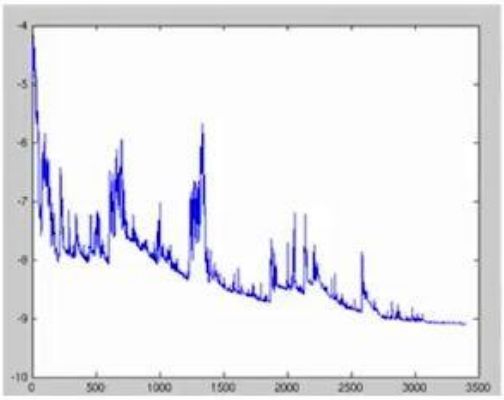


Рис.: Измененние значения J во время обучения

Выбор числа нейронов в скрытом слое H производят различными способами, но ни один из них не является лучшим.

1. Визуальный способ. Если граница классов (или кривая регрессии) слишком сглажена, значит, сеть переупрощена, и необходимо увеличивать число нейронов в скрытом слое. Если граница классов (или кривая регрессии) испытывает слишком резкие колебания, на тестовых данных наблюдаются большие выбросы, веса сети принимают большие по модулю значения, то сеть переусложнена, и скрытый слой следует сократить. Недостаток этого способа в том, что он подходит только для задач с низкой размерностью пространства (небольшим числом признаков).
2. Оптимизация H по внешнему критерию, например, по критерию скользящего контроля или средней ошибки на независимой контрольной выборке $Q(X^k)$. Зависимость внешних критериев от параметра сложности, каким является H, обычно имеет характерный оптимум. Недостаток этого способа в том, что приходится много раз заново строить сеть при различных значениях параметра H, а в случае скользящего контроля — ещё и при различных разбиениях выборки на обучающую и контрольную части.

Динамическое добавление нейронов. Сначала сеть обучается при заведомо недостаточной мощности среднего слоя $H \ll \ell$. Обучение происходит до тех пор, пока ошибка не перестанет убывать. Тогда добавляется один или несколько новых нейронов.

Веса новых связей инициализируются небольшими случайными числами, либо добавленные нейроны обучаются по-отдельности как однослойные персептроны. Во втором случае можно рекомендовать обучать новый персептрон на случайной подвыборке, возможно, добавив в неё те объекты, на которых текущая сеть допустила наибольшие ошибки. Веса старых связей не меняются. Затем проводится настройка сети методом обратного распространения.

После добавления новых нейронов ошибка, как правило, сначала резко возрастает, затем быстро сходится к меньшему значению. Интересно, что общее время обучения обычно оказывается лишь в 1.5–2 раза больше, чем если бы в сети сразу было нужное количество нейронов. Это означает, что информация, накопленная в сети, является полезной и не теряется при добавлении новых нейронов.

При постепенном наращивании сети целесообразно наблюдать за динамикой какого-нибудь внешнего критерия. Прохождение значения $Q(X^k)$ через минимум является надёжным критерием останова, так как свидетельствует о переобученности, вызванной чрезмерным усложнением сети.

Удаление избыточных связей. Метод оптимального прореживания сети (optimalbraindamage, OBD) удаляет те связи, к изменению которых функционал Qнаименее чувствителен. Уменьшение числа весов снижает склонность сети к переобучению.

Метод OBD основан на предположении, что после стабилизации функционала ошибки Qвектор весов wнаходится в локальном минимуме, где функционал может быть аппроксимирован квадратичной формой:

$$Q(w+\delta) = Q(w) + \tfrac{1}{2}\delta^{\intercal} H(w)\delta + o(\|\delta\|^2),$$

где $H(w) = \left(\frac{\partial^2 Q(w)}{\partial w_{jh}\partial w_{j'h'}}\right)$ — гессиан, матрица вторых производных. Как и в диагональном методе Левенберга–Марквардта, предполагается, что диагональные элементы доминируют в гессиане, а остальными частными производными можно пренебречь, положив их равными нулю. Это предположение носит эвристический характер и вводится для того, чтобы избежать трудоёмкого вычисления всего гессиана. Если гессиан $H(w)$ диагонален, то

$$\delta^{\intercal} H(w)\delta = \sum_{j=0}^{J}\sum_{h=1}^{H} \delta_{jh}^2 \frac{\partial^2 Q(w)}{\partial w_{jh}^2}.$$

Обнуление веса w_{jh}эквивалентно выполнению условия $w_{jh} + \delta_{jh} = 0$. Введём величину значимости (salience) синаптической связи, равную изменению функционала $Q(w)$ при обнулении веса: $S_{jh} = w_{jh}^2 \frac{\partial^2 Q(w)}{\partial w_{jh}^2}$.

Эвристика OBD заключается в том, чтобы удалить из сети dсинапсов, соответствующих наименьшим значениям S_{jh}. Здесь d— это ещё один параметр метода настройки. После удаления производится цикл итераций до следующей стабилизации функционала Q. При относительно небольших значениях dградиентный алгоритм довольно быстро находит новый локальный минимум Q. Процесс упрощения сети останавливается, когда внутренний критерий стабилизируется, либо когда заданный внешний критерий начинает возрастать.

Обнуление веса w_{jh}между входным и скрытым слоями означает, что h-й нейрон скрытого слоя не будет учитывать j-й признак. Тем самым происходит отбор информативных признаков для h-го нейрона скрытого слоя.

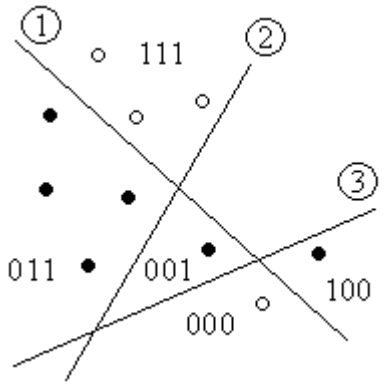
Метод OBD легко приспособить и для настоящего отбора признаков. Вводится суммарная значимость признака $S_j = \sum_{h=1}^{H} S_{jh}$, и из сети удаляется один или несколько признаков с наименьшим значением S_j.

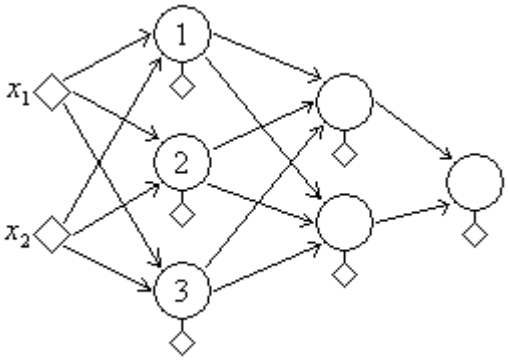
Обнуление веса w_{hm} между скрытым и выходным слоями означает, что m-е выходное значение не зависит от h-го нейрона скрытого слоя. Если выход одномерный (M= 1), то h-й нейрон можно удалить. В случае многомерного выхода для удаления нейронов скрытого слоя вычисляется суммарная значимость $S_h = \sum_{m=1}^{M} S_{hm}$.

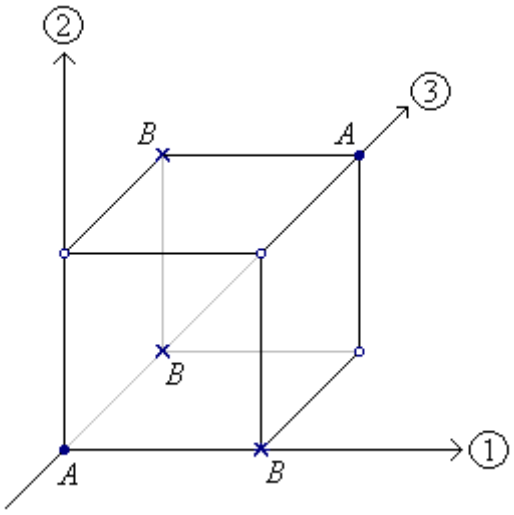
Построение нейросетевого классификатора

Существует два подхода к задаче построения нейронной сети-классификатора. Первый подход заключается в построении сети, варьируя архитектуру. Данный метод основан на точной классификации прецедентов. Второй подход состоит в подборке параметров (весов и порогов) для сети с заданной архитектурой.

Алгоритм, основанный на точной классификации множества прецедентов. Опишем общую идею метода. За основу берется один *нейрон*. Далее наращиваем *нейрон*, пока не получим правильную классификацию всех прецедентов.







Рассмотрим более подробно *алгоритм*. Начинаем с одного нейрона $n(X)$, называемого мастером. После его тренировки получаем разделение *множества* на X^+ и X^-. Если X^+ содержит вектора из двух классов, то вводим новый узел $n(X^+)$, называемый последователем.

Таким образом, на первом слое нейронов находится один мастер и несколько последователей. Никакие вектора из разных классов не имеют одинакового выхода из первого слоя.

$$X_1 = \{y : y = f_1(x), x \in X\},$$

где f_1 – *отображение*, задаваемое первым слоем.

Аналогичным образом строим второй слой, третий слой и т.д.

Утверждение. *При правильном выборе весов каждый очередной слой правильно классифицирует все вектора, которые правильно классифицировал мастер и еще хотя бы один вектор*.

Таким образом, получаем архитектуру, имеющую конечное число слоев, правильно классифицирующие все прецеденты.

Нейроны первого слоя – это биссекторы, разделяющие пары. Второй слой – нейроны and, определяющие *полиэдры*. Третий слой – нейроны or, определяющие классы.

Основным недостатком данного метода является слишком большое количество нейронов.

Алгоритм, основанный на подборе весов для сети с заданной архитектурой. Идея данного метода состоит в том, чтобы ввести критерий в виде функции стоимости, которую необходимо минимизировать.

Пусть

- L – числослоев в сети;
- k_r – число нейронов в слое r, где $r = 1, 2, \ldots, L$;
- k_L – число *выходныхнейронов*;
- $k_0 = l$ – размервхода;
- $x(i) = (x_1(i), x_2(i), \ldots, x_{k_0}(i))$ – входнойвекторпризнаков;
- $y(i) = (y_1(i), y_2(i), \ldots, y_{k_L}(i))$ а – выходной вектор, который должен быть правильно классифицирован.

Текущем состоянии *сеть* при обучении дает результат $\widehat{y}(i)$ не совпадающий с $y(i)$.

Обозначим:

$$J = \sum_{i=1}^{N} \varepsilon(i)$$

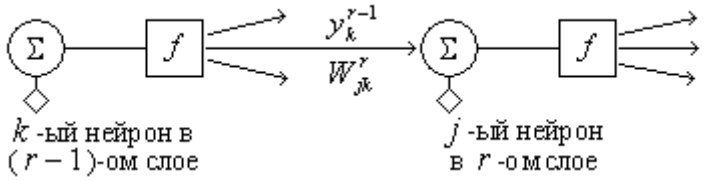
где N – число прецедентов; $\varepsilon(i)$ – ошибка на i -ом прецеденте;

$$\varepsilon(i) = \frac{1}{2}\sum_{m=1}^{k_L} e_m^2(i) = \frac{1}{2}\sum_{m=1}^{k_L}(y_m(i) - \widehat{y}_m(i))^2,$$

где $i = 1, 2, \ldots, N$. J – *функция* всех синоптических весов и порогов. Таким образом, целью обучения является решение оптимизационной задачи:

$$J(W) \to \min,$$

где W – множество синоптических весов.



Пусть y_k^{r-1} – *выход* k -ого нейрона $(r-1)$ -ого слоя; $W - j^r$ – весовой *вектор* (включая порог) j -ого нейрона в r -ом слое, т.е. $W - j^r = (W_{j0}^r, W_{j1}^r, \ldots, W_{jk_{r-1}}^r)$, где k_{r-1} а – число нейронов в $(r-1)$ -ом слое. Таким образом, J – разрывная *функция* M переменных, где

$$M = \sum_{r=1}^{L} k_{r-1}k_r$$

J разрывна, т.к. разрывна *функция активации* f :

$$f(x) = \begin{cases} 1, x > 0 \\ 0, x < 0 \end{cases}$$

Алгоритм обратной волны. Суть – *аппроксимация* непрерывной дифференцируемой функцией за счет замены функции активации "сигмовидной" функцией:

$$f(x) = \frac{1}{1 + e^{-ax}}$$

Вычислим производную функции:

$$f'(x) = \frac{1}{(1+e^{-ax})^2} \cdot ae^{-ax} = a\left(\frac{1}{1+e^{-ax}} - \frac{1}{1+e^{-ax}}\right) = af(x)(1-f(x))$$

При данном чисто формальном приеме вектора признаков уже могут отображаться не только в вершины, но и внутрь гиперкуба. Необходиморешитьзадачуминимизации:
$J(W) \to \min$

Метод градиентного спуска решения задачи минимизации. Пусть $W = \{W - j^k;\ j = 1, 2, \ldots, k_r;\ r = 1, 2, \ldots, L\}$.
Тогда *метод градиентного спуска* выглядит так:

$$\Delta W = -\mu \frac{dJ}{dW},$$

где μ а – шаг *градиентного спуска*. Очевидно, для его реализации необходимо уметь вычислять *градиент* $\frac{dJ}{dW_j^r}$.

Вычисление градиента. *Аргумент* функции активации j -ого нейрона r -ого слоя

$$V_j^r = \sum_{k=1}^{k_{r-1}} W_{jk}^r y_k^{r-1}(i) + W_{j0}^r = \sum_{k=0}^{k_{r-1}} W_{jk}^r y_k^{r-1}(i)$$

принимает различные значения в зависимости от индекса прецедента. В данном случае $y_0^{r-1}(i) = 1$.

Во входном слое, при $r = 1 \quad y_k^{r-1}(i) = x_k(i),\ k = 1, 2, \ldots, k_0$. В выходном слое, при $r = L \quad y_k^r(i) = \widehat{y}_k(i),\ k = 1, 2, \ldots, k_L$.

Рассмотрим *выходнойслой* $r = L$.

$$\varepsilon(i) = \frac{1}{2}\sum_{m=1}^{k_L}(e_m(i))^2 = \frac{1}{2}\sum_{m=1}^{k_L}(f(V_m^L(i)) - y_m(i))^2 = \varepsilon(V_m^L(i)) = \varepsilon(V_m^L(W_m^L), i)$$

$$\frac{\partial \varepsilon(i)}{\partial W_j^L} = \frac{\partial \varepsilon(i)}{\partial V_j^L} \cdot \frac{\partial V_j^L}{\partial W_j^L}$$

$\frac{\partial V_j^L}{\partial W_j^L} = y^{r-1}(i)$ – не зависит от j -ого номера нейрона в слое, т.е. имеем одинаковый *вектор* производных для всех нейронов $(r-1)$ -ого слоя.

$$\frac{\partial \varepsilon(i)}{\partial V_j^L} = (f(V_j^L(i)) - y_j(i)) \cdot f'(V_j^L(i)) = e_j(i) \cdot f'(V_j^L(i))$$

Следовательно, для последнего слоя $\frac{\partial \varepsilon(i)}{\partial V_j^L} = y^{r-1}(i) \cdot e_j(i) \cdot f'(V_j^L(i))$

Рассмотрим скрытый слой $r < L$. Имеетсязависимость:

$$V_k^r = V_k^r(V_j^{r-1})$$

$$\frac{\partial \varepsilon(i)}{\partial V_j^{r-1}(i)} = \sum_{k-1}^{k_r} \frac{\partial \varepsilon(i)}{\partial V_k^r(i)} \cdot \frac{\partial V_k^r(i)}{\partial V_j^{r-1}(i)}$$

$$\frac{\partial V_k^r(i)}{\partial V_j^{r-1}(i)} = \frac{\partial}{\partial V_j^{r-1}(i)} \left[\sum_{m=0}^{k_{r-1}} W_{km}^r y_m^{r-1}(i) \right],$$

но $y_m^{r-1}(i) = f(V_m^{r-1}(i))$, следовательно:

$$\frac{\partial V_k^r(i)}{\partial V_j^{r-1}(i)} = W_{kj}^r \frac{\partial y_j^{r-1}(i)}{\partial V_j^{r-1}(i)} = W_{kj}^r f'(V_j^{r-1}(i))$$

$$\frac{\partial \varepsilon(i)}{\partial V_j^{r-1}(i)} = \left[\sum_{k-1}^{k_r} \frac{\partial \varepsilon(i)}{\partial V_k^r(i)} W_{kj}^r \right] \cdot f'(V_j^{r-1}(i))$$

Сумма, заключенная в квадратных скобках, известна из предыдущего шага.

Описание алгоритма.

0. Начальное приближение. Случайно выбираются веса небольших значений:
$W_{jk}^r,\ r = 1, 2, \ldots, L,\ j = 1, 2, \ldots, k_r,\ k = 0, 1, 2, \ldots, k_{r-1}$.

1. Прямой проход. Для каждого вектора прецедента $x(i), i = 1, 2, \ldots, N$, вычисляются
все $V_j^r(i),\ y_j^r(i) = f(V_j^r(i)),\ j = 1, 2, \ldots, k_r,\ r = 1, 2, \ldots, L$.
Вычисляетсятекущее *значение* ценовойфункции $J(W)$:

Цикл по $i = 1, 2, \ldots, N$ (по прецедентам):

Вычислить:

$y_k^0(i) = x_k(i),\ k = 1, 2, \ldots, k_0$

$y_0^0(i) = 1.$

Цикл по $r = 1, 2, \ldots, L$ (по слоям):

Цикл по $j = 1, 2, \ldots, k_r$ (по нейронам в слое):

$$V_j^r(i) = \sum_{k=0}^{k_{r-1}} W_{jk}^r y_k^{r-1}(i)$$

$$y_j^r(i) = f(V_j^r(i))$$

Конец цикла по j.

Конец цикла по r.

Конец цикла по i.

$$J(W) = \sum_{i=1}^{N} \frac{1}{2}(y_j^L(i) - y_j(i))^2$$

2. Обратный проход. Для каждого значения $i = 1, 2, \ldots, N$ и $j = 1, 2, \ldots, k_L$ вычисляется $\frac{\partial \varepsilon(i)}{\partial V_j^L(i)}$. Затем последовательно необходимо вычислить $\frac{\partial \varepsilon(i)}{\partial V_j^r(i)}$ для всех $r = L - 1, \ldots, 1$ и $j = 1, 2, \ldots, k_r$:

Цикл по $i = 1, 2, \ldots, k_r$ (по нейронам в слое):

Вычислить:

$e_j(i) = y_j^L(i) - y_j(i)$

$\delta_j^L(i) = e_j(i) \cdot f'(V_j^{r-1}(i))$

Цикл по $r = L, L-1, \ldots, 2$ (по слоям):

Цикл по $j = 1, 2, \ldots, k_r$ (по нейронам в слое):

$$e_j^{r-1}(i) = \sum_{k=1}^{(} k_r \delta_k^r(i) \cdot W_{kj}^r$$

$$\delta_j^{r-1}(i) = e_j^{r-1}(i) \cdot f'(V_j^{r-1}(i))$$

Конец цикла по j.

Конец цикла по r.

Конец цикла по i.

3. Пересчет весов. Для всех $r = 1, 2, \ldots, L$ и $j = 1, 2, \ldots, k_r$ $W_j^r(new) = W_j^r(old) + \Delta W_j^r$, где $\Delta W_j^r = -\mu \sum_{i=1}^N \frac{\partial \varepsilon(i)}{\partial V_j^r(i)} y^{r-1}(i)$.

- Останов алгоритма может происходить по двум критериям: либо $J(W)$ стала меньше порога, либо градиент стал очень мал.
- От выбора μ зависит скорость сходимости. Если μ мало, то скорость сходимости также мала. Если μ велико, то и скорость сходимости высока, но при такой скорости можно пропустить $\min$.
- В силу многоэкстремальности существует возможность спустить в локальный минимум. Если данный минимум по каким-то причинам не подходит, надо начинать алгоритм с другой случайной точки.

Данный алгоритм быстрее, чем алгоритм с обучением.

Сети Кохонена

До сих пор мы рассматривали нейронные сети, предназначенные для обучения с учителем, когда для каждого объекта x_i задан соответствующий ему ответ y_i. Сети Кохонена решают задачи обучения без учителя, когда задаются только сами объекты x_i, и требуется выделить обособленные «плотные сгустки» объектов — кластеры, и научиться относить новые объекты к этим кластерам. Кластеризация основывается на предположении, что в пространстве X введена метрика ρ: $X \times X \to \mathbb{R}$, адекватно оценивающая степень сходства любой пары объектов.

Модели конкурентного обучения

Пусть $X = \mathbb{R}^n$ — пространство объектов, $Y = \{1,\ldots,M\}$ — множество кластеров, число M фиксировано. Задана обучающая выборка объектов $X^\ell = \{x_i\}_{i=1}^\ell$. Требуется выработать правило отнесения объектов к кластерам a: $X \to Y$.

Правило жёсткой конкуренции WTA. Наиболее очевидный подход заключается в том, чтобы ввести векторы $w_m \in \mathbb{R}^n$, $m = 1,\ldots,M$, описывающие центры кластеров, и относить произвольный объект $x \in X$ к ближайшему кластеру:

$$a(x) = \arg\min_{m \in Y} \rho(x, w_m).$$

Образно говоря, кластеры соревнуются за право присоединить к себе объект x. Кластер, ближайший к x, называется кластером-победителем, а выражение (7.1) — правилом WTA (winnertakesall).

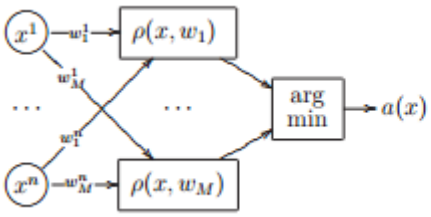


Рис. 22. Представление алгоритма кластеризации (7.1) в виде двухслойной нейронной сети.

Настройка алгоритма кластеризации $a(x)$ сводится к оптимизации расположения центров w_m. Для этого минимизируется функционал качества кластеризации, равный среднему квадрату расстояния между объектами и центрами кластеров:

$$Q(w_1, \ldots, w_M) = \frac{1}{2} \sum_{i=1}^{\ell} \rho^2(x_i, w_{a(x_i)}) \to \min_{\{w_m\}}.$$

Допустим, что метрика евклидова: $\rho(x,w) = \|x - w\|$. Тогда можно выписать градиент функционала Q по вектору w_m:

$$\frac{\partial Q}{\partial w_m} = \sum_{i=1}^{\ell} (w_m - x_i)\big[a(x_i) = m\big].$$

Для поиска центров кластеров w_mможно применить метод стохастического градиента — Алгоритм 4.1, практически без модификаций. Единственное отличие заключается в том, что теперь правило обновления весов на шаге 6 будет иметь вид

$$w_m := w_m + \eta(x_i \quad w_m)\big[a(x_i) = m\big], \tag{7.2}$$

где $x_i \in X^\ell$ — случайным образом выбранный обучающий объект, η— градиентный шаг, он же темп обучения. Смысл данного правила очевиден: если объект x_iотносится к кластеру m, то центр этого кластера w_mнемного сдвигается в направлении объекта x_i, остальные центры не изменяются.

Формальное сходство формулы (7.2) с персептронным правилом наводит на мысль, что кластеризация осуществляется алгоритмом, аналогичным нейронной сети. Выражение (7.1) и в самом деле представимо в виде двухслойной суперпозиции, только вместо привычных скалярных произведений $\langle x, w_m\rangle$вычисляются функции расстояния $\rho(x,w_m)$, а на выходе вместо суммирования применяется операция минимума, см. Рис. 22. При этом центры кластеров w_mвзаимно однозначно соответствуют нейронам скрытого стоя, которые называются нейронами Кохонена. Нейрон, выход которого минимален, называется нейроном-победителем.

Рассмотренная разновидность нейронных сетей называется сетью с конкурентным обучением. Исходно она была предложена как чисто математическая модель. Позже нейрофизиологам удалось найти некоторые её аналоги в биологических нейронных сетях.

> Альтернативное название — *обучающееся векторное квантование* (learning vector quantization, LVQ) — связано с тем, что по исходной выборке из ℓ объектов x_i строятся M новых объектов w_m, похожих на исходные, — это центры ячеек, по которым распределяются («квантуются») исходные объекты. Как правило, $M \ll \ell$, поэтому замена объектов на ближайшие к ним центры позволяет эффективно сжимать данные при незначительной потере информации. Объём сохраняемой информации регулируется единственным параметром M, что достаточно удобно в приложениях.

Правило справедливой конкуренции CWTA. Недостаток конкурентного обучения по правилу WTA заключается в том, что при случайной инициализации весов нейрон Кохонена может попасть в такую область, где он никогда не станет победителем. В результате появится неинформативный пустой кластер.

Для преодоления этого недостатка алгоритм (7.1) немного модифицируется. Вводится «механизм утомления» победителей — правило CWTA (conscienceWTA):

$$a(x) = \arg\min_{m \in Y} C_m \rho(x, w_m), \tag{7.3}$$

где C_m— количество побед m-го нейрона в ходе обучения. Таким образом, кластеры штрафуются за слишком частое присоединение объектов.

Правило мягкой конкуренции WTM. Другим недостатком правила WTA является медленная скорость сходимости, связанная с тем, что на каждой итерации модифицируется только один нейрон-победитель w_m. Для ускорения сходимости, особенно на начальных итерациях, можно подстраивать сразу несколько нейронов, близких к объекту x_i. Для этого вводится ядро — неотрицательная монотонно убывающая на $[0,+\infty)$ функция расстояния $K(\rho)$. Иногда накладывается дополнительное требование

нормировки $K(0) = 1$. Часто берут гауссовское ядро $K(\rho) = \exp(-\beta\rho^2)$ при некотором $\beta>0$. Вместо правила жёсткой конкуренции WTA вводится мягкая конкуренция — правило WTM (winnertakesmost):

$$w_m := w_m + \eta(x_i - w_m)\,K\big(\rho(x_i, w_m)\big), \quad m = 1, \ldots, M. \tag{7.4}$$

Теперь на каждой итерации центры всех кластеров смещаются в сторону x_i, но чем дальше центр находится от x_i, тем меньше величина смещения. Заметим, что (7.2) является частным случаем (7.4), если положить $K\big(\rho(x_i, w_m)\big) = [a(x_i) = m]$.

На начальных итерациях имеет смысл выбрать небольшое значение коэффициента β, чтобы все весовые векторы успели переместиться ближе к области входных векторов. Затем βможно увеличивать, делая конкуренцию всё более жёсткой, и постепенно переходя к коррекции только одного нейрона-победителя.

Благодаря способности к сглаживанию, правило WTM имеет многочисленные применения. Одно из важнейших — самоорганизующиеся карты Кохонена.

Самоорганизующиеся карты Кохонена

Самоорганизующиеся карты Кохонена (self-organizingmaps, SOM) применяются для визуализации многомерных данных. Они дают лишь общую картину, довольно размытую и подверженную искажениям, поскольку спроецировать многомерную выборку на плоскость без искажений в общем случае невозможно. Тем не менее, карты Кохонена позволяют увидеть ключевые особенности кластерной структуры выборки. Они используются на стадии разведочного анализа данных, скорее для общего понимания задачи, чем для получения каких-либо точных результатов.

Идея заключается в том, чтобы спроецировать все объекты выборки на плоскую карту, точнее, на множество узлов прямоугольной сетки заранее заданного размера $M\times H$. На практике Mи Hимеют порядок десятков или сотен. Каждому узлу сетки приписывается нейрон Кохонена с вектором весов $w_{mh}\in \mathrm{R}^n$, $m= 1,\ldots,M$,

Алгоритм 7.7. Обучение карты Кохонена методом стохастического градиента

Вход:
X^ℓ — обучающая выборка;
η — темп обучения;
Выход:
Векторы синаптических весов w_{mh}, $m = 1, \ldots, M$, $h = 1, \ldots, H$;

1: инициализировать веса:
$w_{mh} := \text{random}\left(-\frac{1}{2MH}, \frac{1}{2MH}\right)$;
2: **повторять**
3: выбрать объект x_i из X^ℓ случайным образом;
4: WTA: вычислить координаты узла, в который проецируется объект x_i:
$(m_i, h_i) := a(x_i) \equiv \arg\min_{(m,h)\in Y} \rho(x_i, w_{mh})$;
5: **для всех** $(m, h) \in Y$, достаточно близких к (m_i, h_i)
6: WTM: сделать шаг градиентного спуска:
$w_{mh} := w_{mh} + \eta(x_i - w_{mh})\,K\big(r((m_i, h_i), (m, h))\big)$;
7: **пока** размещение всех объектов в узлах сетки не стабилизируется;

$h = 1,\ldots,H$. Таким образом, множество Y совпадает с множеством узлов сетки, $Y = \{1,\ldots,M\} \times \{1,\ldots,H\}$.

Алгоритм кластеризации $a(x)$ выдаёт пару индексов $(m,h) \in Y$, показывающих, в какой узел сетки проецируется объект x. Чтобы карта отражала кластерную структуру выборки, близкие объекты должны попадать в близкие узлы сетки.

Обучение нейронов производится методом стохастического градиента, см. Алгоритм7.7. После случайного выбора объекта x_i на шаге 3 определяется нейронпобедитель согласно правилу WTA. Соответствующий ему узел сетки обозначается в алгоритме через (m_i,h_i). Затем, согласно правилу WTM, этот нейрон и нейроны, расположенные в ближайших узлах сетки, сдвигаются в сторону вектора x_i. Правило обучения, применяемое на шаге 6, схоже с (7.4), только вместо метрики $\rho(x,x')$, определённой на пространстве объектов, используется евклидова метрика на множестве узлов сетки Y:

$$r\big((m_i,h_i),(m,h)\big) = \sqrt{(m-m_i)^2 + (h-h_i)^2}.$$

По прежнему, $K(\rho)$ — ядро сглаживания, например, $K(\rho) = \exp(-\beta\rho^2)$. Параметр β задаёт степень сглаженности карты: чем меньше β, тем мягче конкуренция нейронов, и тем более сглаженными будут выглядеть границы кластеров на карте. Имеет смысл увеличивать значение параметра β от итерации к итерации, чтобы сеть Кохонена сначала обучилась кластерной структуре «в общих чертах», а затем сосредоточилась на деталях.

Алгоритм останавливается, когда проекции всех, или хотя бы большинства, объектов выборки $(m_i,h_i) = a(x_i)$ перестанут меняться от итерации к итерации.

Искусство интерпретации карт Кохонена. Если через настроенную карту Кохонена (алгоритм $a(x)$) пропустить все объекты обучающей выборки X^ℓ, то кластеры исходного пространства отобразятся в сгустки точек на карте. При этом векторы

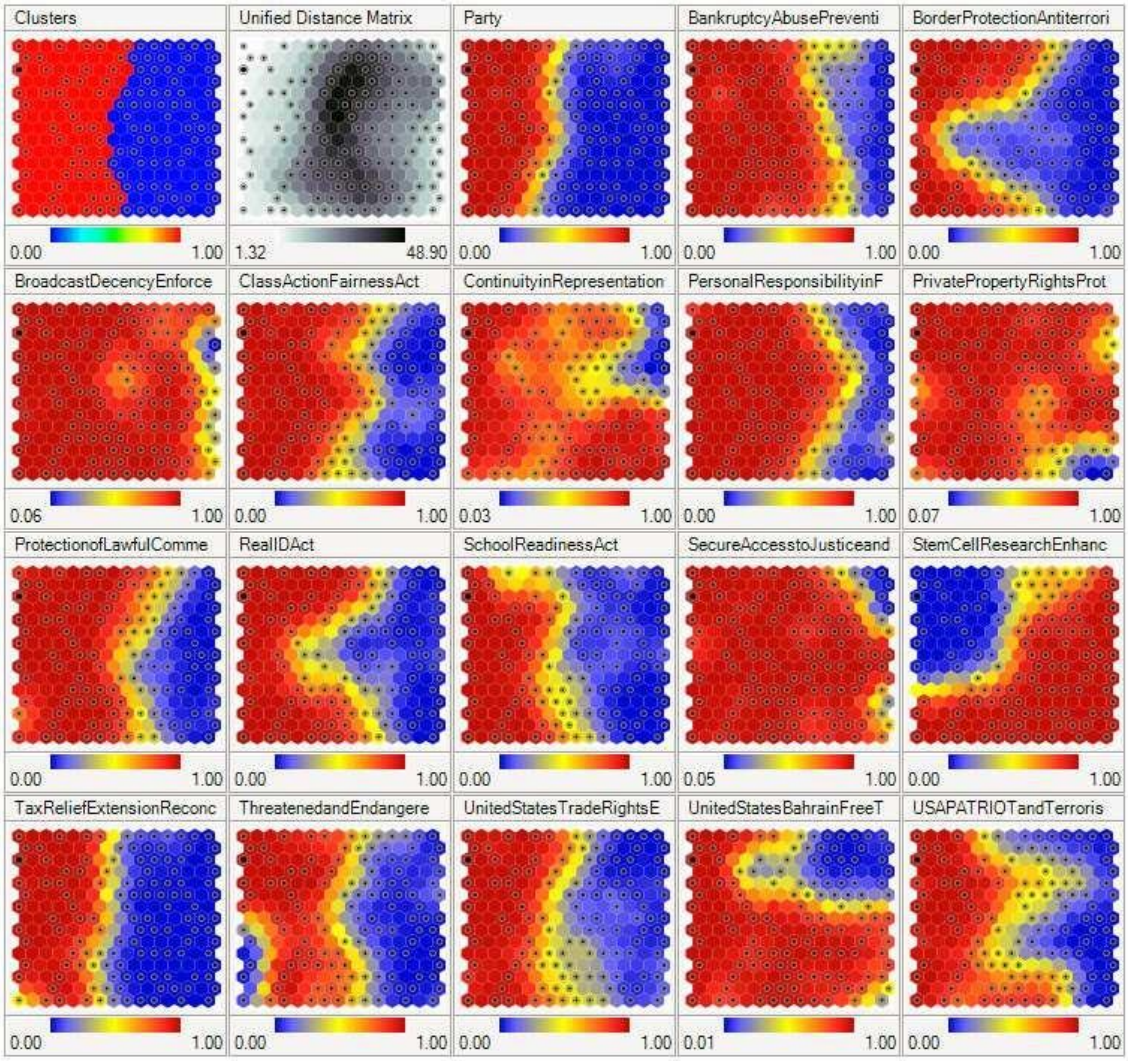


Рис. 23. Карты Кохонена для задачи UCI.house-votes (объекты — конгрессмены, признаки — результаты голосования по 17 вопросам). Первые три графика в верхнем ряду: (1) разделение выборки на два кластера, (2) двухцветная полутоновая карта, (3) цветная карта по признаку «партия» {демократ, республиканец}. Голосование по большинству вопросов хорошо согласуется с партийной принадлежностью конгрессменов.

весов в узлах-сгустках должны быть одновременно похожи на все объекты соответствующих кластеров.

Обычно для каждой точки карты вычисляют среднее расстояние до kближайших точек выборки, и полученное распределение средних расстояний отображают в виде двухцветной полутоновой карты. Чем меньше среднее расстояние, тем выше локальная плотность точек на карте. На той же карте отмечают и сами точки обучающей выборки. На карте хорошо видно, сколько кластеров выделено, в каких местах они расположились, и какие объекты выборки в них попадают, Рис.23.

Следующий шаг — поиск интерпретации кластеров. Для этого строятся ещё nкарт, по одной карте на каждый признак. Теперь цвет узла (m,h) соответствует значению j-й компоненты вектора весов $w_{m,h}$. Обычно эти карты раскрашивают в геодезические цвета

от синего до коричневого. Синие участки карты соответствуют наименьшим значениям *j*-го признака, красные — наибольшим. Сравнивая карты признаков с общей картой кластеров, можно найти кластеры, которым свойственны повышенные или пониженные значения отдельных признаков.

В результате содержательная интерпретация кластеров формулируется путём перечисления всех признаков, имеющих либо повышенные, либо пониженные значения в каждом из кластеров.

Ещё один способ интерпретации — выделение на карте всех узлов, в которые попадает выбранное подмножество объектов. Если объекты сгруппировались в одном месте, значит мы имеем дело с кластером или его частью. Выделяя различные подмножества объектов, имеющие заведомо известную содержательную интерпретацию, можно находить интерпретации различных зон на карте.

Недостатки карт Кохонена.

- Субъективность. Не всегда ясно, какие особенности карты обусловлены кластерной структурой данных, а какие — свойствами сглаживающего ядра. -

  От выбора параметра βсущественно зависит сглаженность границ кластеров и степень детализации карты.

- Наличие искажений. Близкие объекты исходного пространства могут перехо-дить в далёкие точки на карте. В частности, объективно существующие кластеры могут разрываться на фрагменты. И наоборот, далёкие объекты могут случайно оказаться на карте рядом, особенно, если они были одинаково далеки от всех кластеров. Искажения неизбежны при проецировании многомерной выборки на плоскость. Распределение точек на карте позволяет судить лишь о локальной структуре многомерных данных, и то не всегда.

- Зависимость от инициализации. Начальное распределение весов существенновлияет на процесс обучения и может сказываться не только на расположении кластеров, но даже на их количестве. Иногда рекомендуется построить несколько карт и выбрать из них наиболее «удачную».

Не секрет, что популярность карт Кохонена обусловлена в значительной степени их эстетической привлекательностью и впечатлением наукообразной загадочности, которое они производят на неискушённого пользователя. На практике они применяются в основном как вспомогательное средство для предварительного визуального анализа и выявления неочевидных закономерностей в многомерных данных.

Гибридные сети встречного распространения

Ещё одно важное применение нейронов Кохонена с их способностью кластеризовать исходные векторы — кусочная аппроксимация функций.

Рассмотрим задачу восстановления регрессии, когда на элементах обучающей выборки $X^\ell = \{x_i\}_{i=1}^\ell$ заданы вещественные ответы $y_i = y^*(x_i)$. Идея заключается в том, чтобы сначала разбить обучающие объекты (и всё пространство X) на кластеры, не пользуясь ответами y_i, затем для каждого кластера построить свою локальную аппроксимацию целевой зависимости y^*. При использовании жёсткой конкуренции WTA эти аппроксимации образуют кусочно-непрерывную функцию. Мягкая конкуренция WTM «склеивает» из локальных кусков гладкую аппроксимацию.

Кусочно-постоянная аппроксимация. Чтобы алгоритм кластеризации (7.1) превратить в алгоритм кусочной аппроксимации, к нему добавляется ещё один нейрон с линейной функцией активации, образующий третий слой с весами $v_1,\ldots,v_M$:

$$a(x) = v_{m^*(x)} = \sum_{m=1}^{M} v_m \big[m^*(x) = m\big]; \qquad (7.5)$$

$$m^*(x) = \arg\min_{m=1,\ldots,M} \rho(x, w_m);$$

где $m^*(x)$ — номер нейрона-победителя на объекте x, вычисляемый по правилу WTA.

При квадратичной функции потерь задача настройки весов v_m легко решается аналитически:

$$Q(v) = \frac{1}{2}\sum_{i=1}^{\ell}\big(a(x_i) - y_i\big)^2 \to \min_v;$$

$$\frac{\partial Q}{\partial v_m} = \sum_{i=1}^{\ell}\big(a(x_i) - y_i\big)\big[m^*(x_i) = m\big] = 0.$$

Подставляя сюда выражение для $a(x_i)$ из (7.5), получаем

$$v_m = \frac{\sum_{i=1}^{\ell} y_i\big[m^*(x_i) = m\big]}{\sum_{i=1}^{\ell}\big[m^*(x_i) = m\big]}.$$

Проще говоря, оптимальное значение весового коэффициента v_m есть среднее y_i по всем объектам x_i, попавшим в m-й кластер. Ответ алгоритма $a(x)$ для всех объектов, попадающих в m-й кластер, будет одинаков и равен v_m. Это означает, что $a(x)$ реализует кусочно-постоянную функцию.

Гладкая аппроксимация строится с помощью правила мягкой конкуренции WTM при выбранном гладком ядре $K(\rho)$. Алгоритм $a(x)$ представляет собой формулу Надарая-Ватсона для сглаживания ответов v_m по M точкам — центрам кластеров:

$$a(x) = \sum_{m=1}^{M} v_m \frac{K\big(\rho(x, w_m)\big)}{\sum_{s=1}^{M} K\big(\rho(x, w_s)\big)}.$$

В этом случае задача минимизации $Q(v)$ сводится к решению системы линейных уравнений размера $M \times M$. Вместо явного решения системы можно снова воспользоваться

методом стохастического градиента. На каждой итерации обновляются и веса слоя Кохоненaw_m, и веса третьего (выходного) слоя v_m:

$$\begin{cases} w_m := w_m - \eta(w_m - x_i)K\big(\rho(x_i, w_m)\big) \\ v_m := v_m - \eta(a(x_i) - y_i)\dfrac{K\big(\rho(x_i, w_m)\big)}{\sum_{s=1}^{M} K\big(\rho(x_i, w_s)\big)}; \end{cases}$$

Заметим, что обновление весов w_mвлияет на веса v_m, а обратного влияния нет.

Данный метод получил название встречного распространения ошибки, поскольку второй слой настраивается по правилу Кохонена, а третий слой — так же, как в методе back-propagation.

Многомерное шкалирование

Визуализация кластерной структуры заданной выборки объектов X^ℓ — непростая проблема, особенно если выборка содержит тысячи объектов, а пространство объектов существенно многомерно.

Задача многомерного шкалирования (multidimensionalscaling, MDS) заключается в следующем. Имеется обучающая выборка объектов $X^\ell = \{x_1,...,x_\ell\} \subset X$. Заданы расстояния $R_{ij} = \rho(x_i,x_j)$ для некоторых пар обучающих объектов $(i,j) \in D$. Требуется для каждого объекта $x_i \in X^\ell$ построить его признаковое описание — вектор $x_i = (x^1_i,...,x^n_i)$ в евклидовом пространстве $\mathbb{R}^n$так, чтобы евклидовы расстояния d_{ij}между объектами x_iи x_j

$$d_{ij}^2 = \sum_{d=1}^{n}(x_i^d - x_j^d)^2$$

как можно точнее приближали исходные расстояния R_{ij}для всех $(i,j) \in D$. Данное требование можно формализовать по-разному; один из наиболее распространённых способов — через минимизацию функционала, называемого стрессом:

$$S(X^\ell) = \sum_{(i,j)\in D} w_{ij}(d_{ij} - R_{ij})^2 \to \min,$$

где минимум берётся по совокупности ℓnпеременных $(x_i^d)_{i=1,\ell}^{d=1,n}$.

Размерность nобычно задаётся небольшая. В частности, при $n = 2$ многомерное шкалирование позволяет отобразить выборку в виде множества точек на плоскости (scatterplot). Плоское представление, как правило, искажено ($S>0$), но в целом отражает основные структурные особенности многомерной выборки, в частности, её кластерную структуру. Поэтому двумерное шкалирование часто используют как инструмент предварительного анализа и понимания данных.

На практике расстояния чаще бывают известны для всех пар объектов, но мы будем рассматривать более общий случай, когда D— произвольное заданное множество пар индексов объектов.

Веса w_{ij}задаются исходя из целей шкалирования. Обычно берут $w_{ij} = (R_{ij})^\gamma$. При $\gamma<0$ приоритет отдаётся более точному приближению малых расстояний; при $\gamma>0$ — больших

расстояний. Наиболее адекватным считается значение $\gamma = -2$, когда функционал стресса приобретает физический смысл потенциальной энергии системы ℓ материальных точек, связанных упругими связями; требуется найти равновесное состояние системы, в котором потенциальная энергия минимальна.

Функционал стресса $S(X^\ell)$ сложным образом зависит от ℓnпеременных, имеет огромное количество локальных минимумов, и его вычисление довольно трудоёмко. Поэтому многие алгоритмы многомерного шкалирования основаны на итерационном размещении объектов по одному. На каждой итерации оптимизируются евклидовы координаты только одного из объектов при фиксированных остальных объектах.

Размещение одного объекта методом Ньютона–Рафсона. Рассмотрим аддитивную составляющую функционала стресса, зависящую только от одного объекта $x \in X^\ell$ с координатами $x \equiv (x^1,\ldots,x^n)$, которые и требуется найти:

$$S(x) = \sum_{x_i \in U} w_i \big(d_i(x) - R_i\big)^2 \to \min_x,$$

где $U \subseteq X^\ell$ — все объекты с известными расстояниями $R_i = \rho(x,x_i)$; $d_i(x)$ — евклидово расстояние между векторами$x \equiv (x^1, \ldots, x^n)$ и $x_i \equiv (x_i^1, \ldots, x_i^n)$.

Применим метод Ньютона–Рафсона для минимизации $S(x)$.

Поскольку функционал многоэкстремальный, очень важно выбрать удачное начальное приближение $x^{(0)}$. Вектор xдолжен быть похож на те векторы $x_i \in U$, для которых расстояния R_iмалы. Неплохая эвристика — взять в качестве $x^{(0)}$ взвешенное среднее всех векторов из U, например, с весами $c_i = R_i^{-2}$:

$$x^{(0)} = \frac{\sum_{x_i \in U} c_i x_i}{\sum_{x_i \in U} c_i}.$$

Затем запускается итерационный процесс

$$x^{(t+1)} := x^{(t)} - h_t \big(S''(x^{(t)})\big)^{-1} S'(x^{(t)}),$$

где $S'(x^{(t)})$ — вектор первых производных (градиент) функционала $S(x)$ в точке $x^{(t)}$, $S''(x^{(t)})$ — матрица вторых производных (гессиан) функционала $S(x)$ в точке $x^{(t)}$, h_t— величина шага, который можно положить равным 1, но более тщательный его подбор способен увеличить скорость сходимости.

Найдём выражения для градиента и гессиана.

$$\frac{\partial d_i}{\partial x^a} = \frac{x^a - x_i^a}{d_i};$$

$$\frac{\partial S}{\partial x^a} = 2 \sum_{x_i \in U} w_i \left(1 - \frac{R_i}{d_i}\right)\left(x^a - x_i^a\right);$$

$$\frac{\partial^2 S}{\partial x^a \partial x^a} = 2 \sum_{x_i \in U} w_i \left(\frac{R_i}{d_i}\left(\frac{x^a - x_i^a}{d_i}\right)^2 - \frac{R_i}{d_i} + 1\right)$$

$$\frac{\partial^2 S}{\partial x^a \partial x^b} = 2 \sum_{x_i \in U} w_i \frac{R_i}{d_i}\left(\frac{x^a - x_i^a}{d_i}\right)\left(\frac{x^b - x_i^b}{d_i}\right). !;$$

Заметим, что в пространствах малой размерности n6 3 обращение гессиана — довольно простая операция, однако с ростом размерности вычислительные затраты растут как $O(n^3)$.

Итерации Ньютона–Рафсона прекращаются, когда значение стресса $S(x)$ стабилизируется или вектор xперестаёт существенно изменяться, то есть стабилизируется норма разности $\|x^{(t+1)} - x^{(t)}\|$.

Будем полагать, что размещение объекта xотносительно множества уже размещённых объектов $U \subseteq X^\ell$ реализуется процедурой НьютонРафсон(x,U).

Субквадратичный алгоритм многомерного шкалирования. Алгоритм 7.8 начинает с того, что находит две самые удалённые точки выборки x_i,x_j. Достаточно решить эту задачу приближённо. В частности, можно взять произвольную точку, найти самую удалённую от неё, затем для этой точки найти самую удалённую, и так далее. Обычно 3–4 итераций хватает, чтобы найти пару достаточно далёких точек. Найденным точкам приписываются (в двумерном случае) евклидовы координаты (0,0) и (0,R_{ij}). Затем находится третья точка x_k, наиболее удалённая от первых

Алгоритм 7.8. Субквадратичный алгоритм многомерного шкалирования

Вход:

Вход:
R_{ij} — матрица попарных расстояний между объектами, возможно, разреженная
K — размер скелета;
Выход:
евклидовы координаты всех объектов выборки $x_i \equiv (x_i^1, \ldots, x_i^n)$, $i = 1, \ldots, \ell$;

1: Инициализировать скелет:
$U :=$ три достаточно далёкие друг от друга точки;
2: **пока** $|U| < K$ — наращивать скелет:
3: $x := \arg\max_{x_i \in X^\ell \setminus U} \left(\min_{x_j \in U} R_{ij} \right)$;
4: НьютонРафсон(x, U);
5: $U := U \cup \{x\}$;
6: **пока** координаты точек скелета не стабилизируются:
7: найти наиболее напряжённую точку в скелете:
$x := \arg\max_{x_i \in U} S(x)$;
8: НьютонРафсон$(x, U \setminus \{x\})$;
9: **для** $x \in X^\ell \setminus U$
10: НьютонРафсон(x, U);

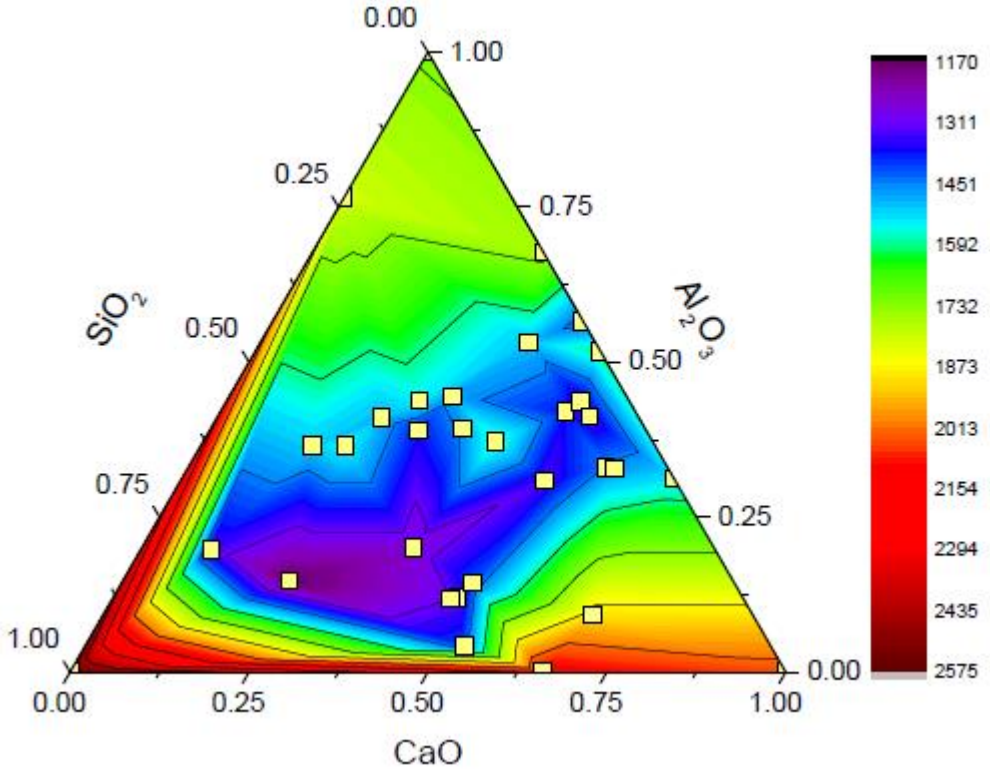
двух, то есть для которой значение $\min\{R_{ki},R_{kj}\}$ максимально. Евклидовы координаты x_kопределяются (в двумерном случае) исходя из того, что треугольник Δijk жёстко задан длинами своих сторон.

Затем начинается поочерёдное добавление точек и их размещение относительно уже имеющихся. После размещения первых Kточек их положение уточняется друг относительно друга. Эти точки, размещённые с особой тщательностью, становятся

«скелетом», относительно которого размещаются все остальные точки. Расстояния между «не-скелетными» точками в алгоритме вообще не задействуются.

Если $K = \ell$, то все точки будут «скелетными»; в этом случае размещение является наиболее точным, но и наиболее долгим, требуя $O(\ell^2)$ операций. В общем случае число операций алгоритма $O(K^2) + O(K\ell)$. Выбирая размер «скелета» K, можно находить компромисс между точностью и временем построения решения.

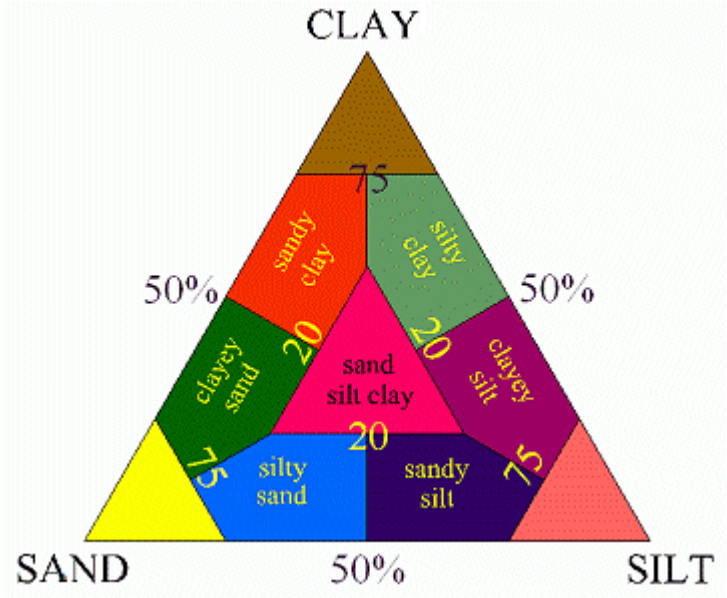
Карта сходства отображает результат многомерного шкалирования при $n = 2$ в виде плоского точечного графика. Евклидова метрика и функционал стресса инвариантны относительно произвольных ортогональных преобразований карты сходства — сдвигов, поворотов и зеркальных отражений. Поэтому оси на карте не имеют интерпретации. Для понимания карты на ней обозначают ориентиры — те объекты выборки, интерпретации которых хорошо известны. Если искажения расстояний на карте не велики, то объекты, близкие к ориентирам, должны иметь схожие интерпретации.



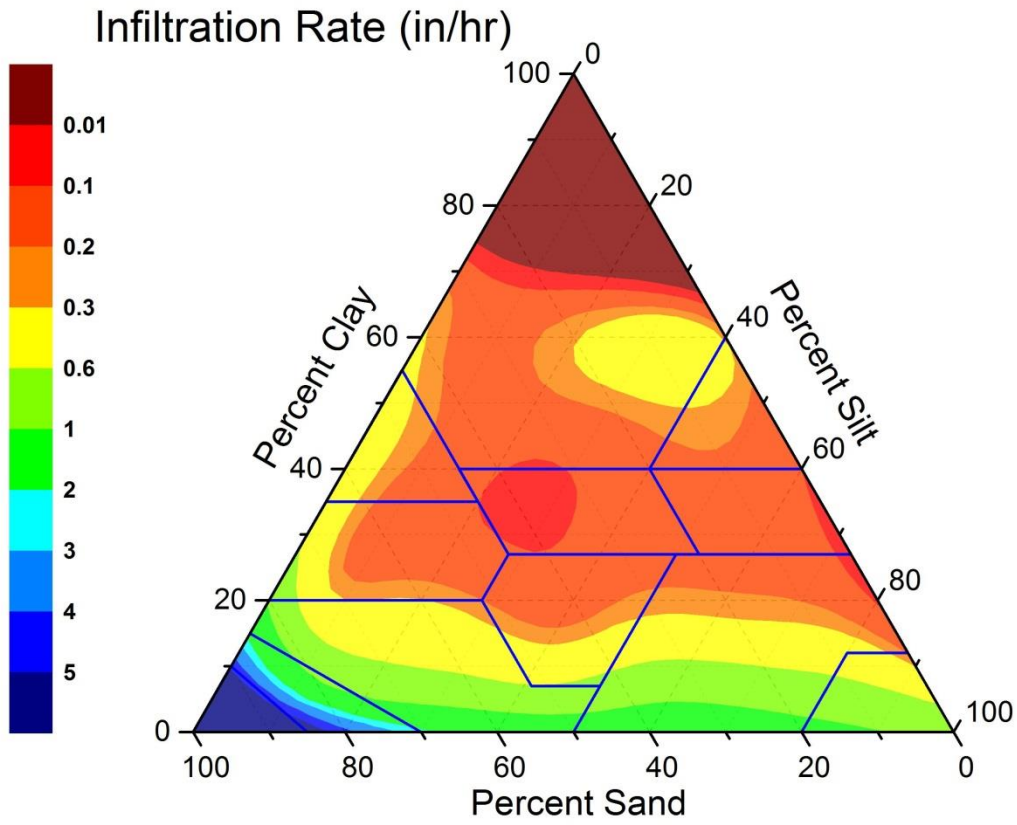
Карта сходства даёт лишь общее представление о взаимном расположении объектов выборки. Делать на её основе какие-либо количественные выводы, как правило, нельзя.

Диаграмма Шепарда позволяет сказать, насколько сильно искажены расстояния на карте сходства. Это точечный график; по горизонтальной оси откладываются исходные расстояния R_{ij}; по вертикальной оси откладываются евклидовы расстояния d_{ij}; каждая точка на графике соответствует некоторой паре объектов $(i,j) \in D$. Если число пар превышает несколько тысяч, отображается случайное подмножество пар. Иногда на диаграмме изображается сглаженная зависимость $d_{ij}(R_{ij})$, а также сглаженные границы верхних и нижних доверительных интервалов, в которых $d_{ij}(R)$ находится с высокой вероятностью (например, 90%) при каждом значении R.

Идеальной диаграммой Шепарда является наклонная прямая — биссектриса первой четверти. Чем «толще» облако точек, представленное на диаграмме, тем сильнее искажения, и тем меньшего доверия заслуживает карта сходства.



Пример 7.1. Один из надёжных способов тестирования методов многомерного шкалирования— размещение изначально двумерных (или близких к двумерным) выборок. Например, можно взять множество городов Российской Федерации, и задать R_{ij}как евклидовы расстояния между их географическими координатами (долгота, широта). Хороший алгоритм шкалирования должен построить карту сходства, похожую на географическую карту (с точностью до поворотов и отражений), а диаграмма Шепарда должна оказаться практически диагональной.



Список литературы

[1] Vapnik V. Statistical Learning Theory. — Wiley, New York, 1998.

[2] Vapnik V. The nature of statistical learning theory. — 2 edition. — Springer-Verlag, New York, 2000.

[3] Waterhouse S. R., Robinson A. J. Classification using hierarchical mixtures of experts // Proceedings of the 1994 IEEE Workshop on Neural Networks for Signal Processing IV. — Long Beach, CA: IEEE Press, 1994.— Pp. 177–186. http://citeseer.ist.psu.edu/waterhouse94classification.html.

[4] Wolpert D. H. Stacked generalization // Neural Networks.— 1992.— no. 5.— Pp. 241–259. http://citeseer.ist.psu.edu/wolpert92stacked.html.

[5] Marchand M., Shawe-Taylor J. Learning with the set covering machine // Proc. 18th International Conf. on Machine Learning.— Morgan Kaufmann, San Francisco, CA, 2001.— Pp. 345–352. http://citeseer.ist.psu.edu/452556.html.

[6] Mason L. Margins and Combined Classifiers: Ph.D. thesis / Australian National University. — 1999.

[7] Mason L., Bartlett P., Baxter J. Direct optimization of margins improves generalization in combined classifiers // Proceedings of the 1998 conference on Advances in Neural Information Processing Systems II. — MIT Press, 1999.— Pp. 288–294. http://citeseer.ist.psu.edu/mason98direct.html.

[8] Рудаков К. В. О применении универсальных ограничений при исследовании алгоритмов классификации // Кибернетика.— 1988.— № 1.— С. 1–5. http://www.ccas.ru/frc/papers/rudakov88universal.pdf.

[9] Рудаков К. В., Воронцов К. В. О методах оптимизации и монотонной коррекции в алгебраическом подходе к проблеме распознавания // Докл. РАН.— 1999.— Т. 367, № 3.— С. 314–317. http://www.ccas.ru/frc/papers/rudvoron99dan.pdf. .

[10] Adaptive mixtures of local experts / R. A. Jacobs, M. I. Jordan, S. J. Nowlan, G. E. Hinton // Neural Computation.— 1991.— no. 3.— Pp. 79–87.

[11] Boosting algorithms as gradient descent / L. Mason, J. Baxter, P. Bartlett, M. Frean // Proceedings of the 1999 conference on Advances in Neural Information Processing Systems 12.— MIT Press, 1999.

[12] Fu¨rnkranz J., Flach P. A. Roc ‘n’ rule learning-towards a better understanding of covering algorithms // Machine Learning.— 2005.— Vol. 58, no. 1. — Pp. 39–77. http://dblp.uni-trier.de/db/journals/ml/ml58.html#FurnkranzF05.

[13] Hidber C. Online association rule mining // SIGMOD Conf. — 1999.— Pp. 145–156. http://citeseer.ist.psu.edu/hidber98online.html.

[14] Hipp J., Gu¨ntzer U., Nakhaeizadeh G. Algorithms for association rule mining — a general survey and comparison // SIGKDD Explorations.— 2000.— Vol. 2, no. 1.— Pp. 58–64.

[15] Норушис А. Построение логических (древообразных) классификаторов методами нисходящего поиска (обзор) // Статистические проблемы управления. Вып. 93 / Под ред. Ш. Раудис. — Вильнюс, 1990.— С. 131–158.

[16] Полякова М. П., Вайнцвайг М. Н. Об использовании метода «голосования» признаков в алгоритмах распознавания // Моделирование обучения и поведения. — М., 1975.— С. 25—28.