Лекция 7. Изкуствени невронни мрежи

7.1. Биологична мотивация

Самообучението с невронни мрежи е един надежден метод за апроксимиране на целочислени, непрекъснати и векторни целеви функции. За определени типове задачи, като, например, научаването да се интерпретират сложните реални данни от сензори, изкуствените невронни мрежи (ИНМ) се нареждат сред най-ефективни известни досега алгоритми. Например, алгоритъм BACKPROPAGATION, който ще разгледаме в следващата лекция, е доказано много успешен при решаване на такива реални задачи, като разпознаване на ръкописен текст, разпознаване на реч и разпознаване на лица.

Изучаването на ИНМ е било частично вдъхновено от наблюдението, че биологичните самообучаващи се системи са построени от много големи мрежи от взаимосвързани неврони. По аналогия изкуствените невронни мрежи са построени от силно свързано множество прости единици – възли, където всеки възел има няколко непрекъснати входове (които, възможно, са изходи на други възли) и произвежда един единствен непрекъснат изход (който може да бъде вход на множество други възли).

За да почувстваме тази аналогия, нека да разгледаме няколко факта от невробиологията. Оценява се, например, че човешкият мозък съдържа силно взаимосвързана мрежа от приблизително 10^{11} неврона, всеки от които, в средно, е свързан с 10^4 други неврони. Активността на един неврон е обикновено се възбужда или се потиска чрез връзки с други неврони. Известно е, че най-бързото време за превключване на неврона е от порядъка на 10^{-3} секунди – доста бавно по сравнение със скоростта на превключване при компютрите – 10^{-10} секунди. И все пак хората са способни да приемат учудващо сложни решения при това учудващо бързо. Например, за разпознаване на лицето на майка ви, ви е необходимо около 10^{-1} секунди. Обърнете внимание, че последователността от възбужданията на неврони, които се осъществяват през този 10^{-1} секунден интервал, не може да бъде по-дълга от неколкостотин стъпки, отчитайки скоростта на превключване на единични неврони. Това наблюдение доведе до предположение, че възможностите за обработка на информация при биологичните невронни системи трябва да са следствие от силно паралелизирани процеси, изпълнявани върху представяния, които са разпределени между множество неврони. Една от мотивации за ИНМ-системи е да се реализира този вид високо паралелни изчисления, базирани върху разпределеното представяне. Повечето от ИНМ-програми се изпълняват върху последователни машини, емулиращи разпределени процеси, макар че по-бързите версии на алгоритмите вече бяха имплементирани върху високо паралелни компютри, специално конструирани за приложенията на ИНМ.

Макар че ИНМ са силно мотивирани от биологичните невронни системи (БНС), БНС имат много характеристики, които не се моделират в ИНМ, както и ИНМ имат много характеристики, за които е известно, че не са съвместими с БНС.

Например, ние разглеждаме ИНМ, в които изходът на всеки възел е една единствена постоянна величина, докато при биологичните неврони изходът е една сложна времева серия от пикови импулси.

Исторически, върху изкуствени невронни мрежи са работили две групи от учени. Първата група е била мотивирана от целта да бъдат използвани ИНМ за изучаване и моделиране на процеси на биологичното самообучение. Втората група имаше за цел създаване на високоефективни алгоритми за машинно самообучение, не зависимо от това, дали те отразяват биологичните процеси или не. В нашите лекции ще следваме посоката на втората група.

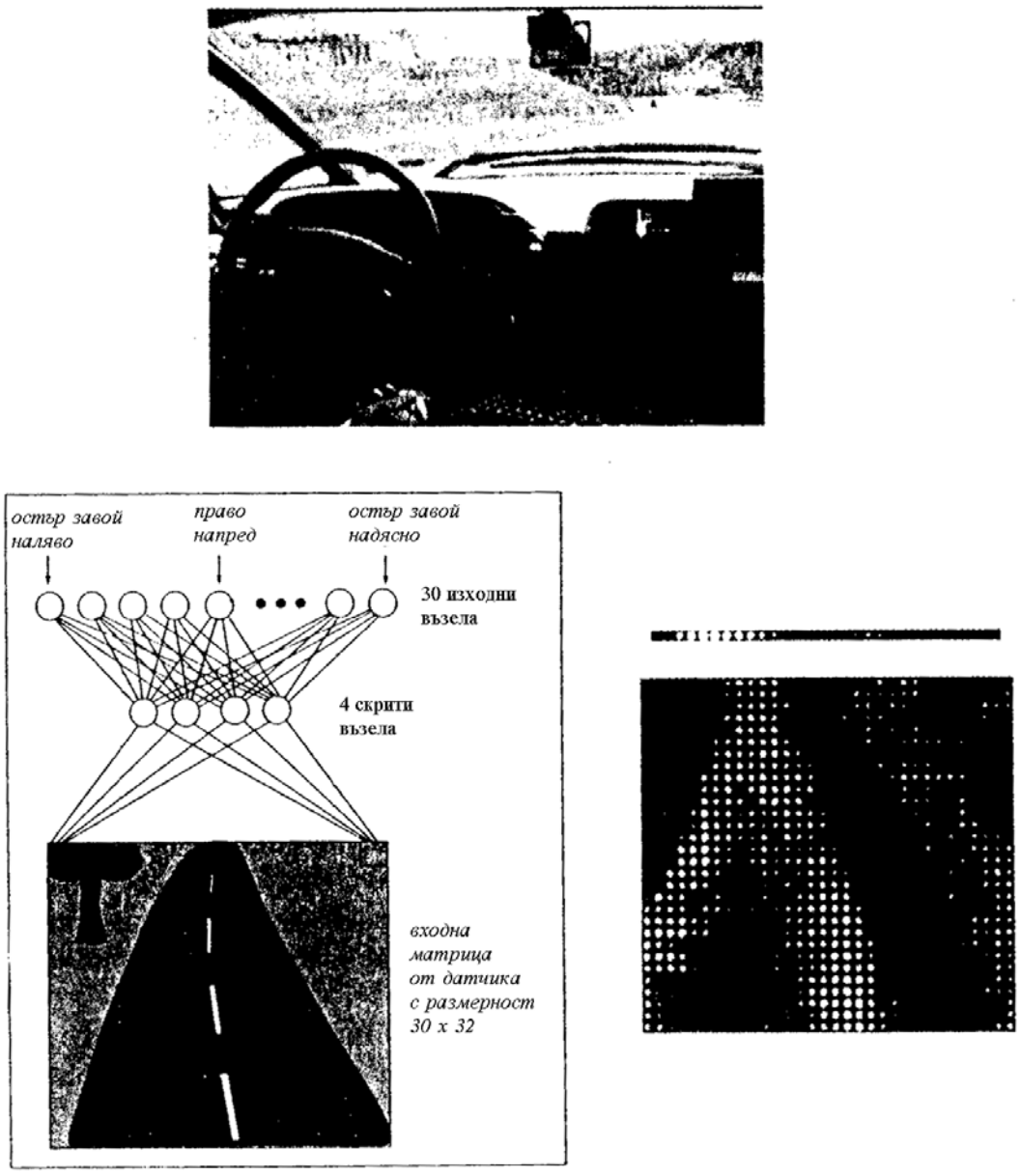
7.2. Представяне на невронни мрежи

Типичният пример на обучение с ИНМ е представен от системата ALVINN (Pomerleau 1993), където обучената ИНМ се използва за управление на автомобила, който се движи с нормална скорост по обществени пътища. Като вход на невронната мрежа се използва решетка от 30 x 32 пиксела от интензивности, получена от разположената отпред на колата камера. Изходът на мрежата е посоката, в която колата ще бъде насочена. ИНМ се обучава да подражава на наблюдаваните команди за управление, издавани от човека при каране на кола, в течение на около 5 минути. ALVINN използва своята обучена невронна мрежа, за да успешно кара колата със скорост до 70 мили в час на разстояние от 90 мили по обществени пътища (карането става в лявата линия на многолентов път, по който се движат и други автомобили).

Фигура 7-1 илюстрира представянето на невронна мрежа, използвана в една от версиите на ALVINN, която е типична за повечето ИНМ системи. Мрежата е показана в лявата част от фигурата заедно с входната картина от камерата. Всеки възел (кръгче) на мрежата върху диаграмата съответства на изхода на един възел на ИНМ, а линиите, влизащите отдолу във възела – на неговите входове. Има четири възела, които получават входове директно от всички 30 x 32 пиксела на картината. Те са наричат “скрити” възли, тъй като техните изходи са достъпни само вътре в мрежата и не са част от глобалния изход на мрежата. Всеки от тези скрити възли изчислява по един непрекъснат изход, базиран върху претеглената комбинация от неговите 960 входа. След това изходите на тези скрити възли се използват като входове за второто ниво от 30 “изходни” възли. Всеки изходен възел съответства на конкретно направление при управление, а стойностите на изхода на тези възли определят, кое от тези направления се препоръчва най-силно.

Диаграмата в дясната част от фигурата показва стойностите на научените тегла, асоциирани с един от четирите скрити възли на тази ИНМ. Голямата матрица от черни и бели квадратчета визуализира теглата на 30 x 32 пикселни входове към скрития възел. Тука едно бяло квадратче означава положително тегло, а черното – отрицателното, като размерът на квадратчето означава величината на теглото. По-малката правоъгълна диаграма над голямата матрица показва теглата на връзки от този скрит възел към всеки от 30 изходни възли.

Мрежовата структура на ALVINN е типична за повечето ИНМ. Отделните възли са взаимосвързани по нива, които формират един насочен ацикличен граф. В общия случай, ИНМ могат да бъдат графи с различен тип структура – ациклични или циклични, насочени или не. Ще разгледаме най-често срещани и използвани подходи към ИНМ, които се базират върху алгоритъма BACKPROPAGATION. В този алгоритъм се подразбира, че мрежата има една фиксирана структура, съответстваща на насочен граф, който може да има и цикли. Обучението съответства на избор на определена стойност на тегло за всяка дъга на графа. Макар, че някои типове цикли са разрешени, преобладаващото мнозинство от практическите приложения използват ациклични мрежи с пряко разпространение на активация, подобни на структурата, използвана от ALVINN.



Фиг 7-1. Невронна мрежа за управление на автономен автомобил

7.3. Задачи, подходящи за самообучение с невронни мрежи

Самообучението с ИНМ е най-добре подхожда към задачите, в които обучаващите примери съответстват на зашумени, сложни данни от датчици, като, например, камери и микрофони. То също така е приложимо към задачите, за които често се използват и символното представяне, от типа на обучение с класификационни дървета, които сме обсъждали в лекции 3 и 4. В тези случаи класификационните дървета и ИНМ дават резултати със сравнима точност. Най-често използваният алгоритъм за обучение на ИНМ е BACKPROPAGATION. Той е подходящ за задачи със следните характеристики:

- *Примерите са представени чрез множество от двойки атрибут – стойност.* Целевата функция, подлежаща на научаване, се определя върху примери, които могат да бъдат описани като вектор от предварително определени признаци (като например стойностите на пиксели в примера с ALVINN). Тези входни атрибути могат да имат силна корелация помежду си или да бъдат независими един от друг. Входните стойности могат да бъдат всякакви реални числа.
- *Изходът на целевата функция може да бъде целочислен, непрекъснат (реални числа) или вектор от няколко непрекъснати или целочислени атрибута.* Например, в системата ALVINN изходът е вектор от 30 атрибута, всеки от които съответства на определена препоръка за посока на управление. Стойността на всеки изход е някакво реално число между 0 и 1, което, в този случай, отговаря на степента на убеденост в предсказваното направление. Можем също така да обучим мрежата да извежда както командата за управление, така и препоръчаното ускорение, използвайки проста конкатенация на вектори, кодиращи тези два предсказвания.
- *Обучаващите примери могат да съдържат грешки.* Методи за обучение на ИНМ са доста устойчиви на наличие на шума в обучаващите данни.
- *Дългото време за обучение е приемливо.* Алгоритмите за обучение на невронни мрежи обикновено изискват по-голямо време за обучение, отколко, например, класификационни дървета. Времето за обучение може да варира от няколко секунди до няколко часа, в зависимост от такива фактори, като броя на тегла в мрежата, броя на обучаващите примери и стойностите на различни параметри на алгоритмите.
- *Изисква се бързо оценяване на научената целева функция.* Макар, че времето за обучение на една ИНМ е сравнително голямо, прилагането на обучената мрежа с цел оценяването на някой новопостъпил пример, обикновено става много бързо. Например, за да променя постоянно своите управляващи команди при движението на автомобила напред, системата ALVINN прилага своята невронна мрежа по няколко пъти в секунда.
- *Способността на хора да разбират научената целева функция не е от значение.* Теглата, научени от невронната мрежа, често са трудно подлежат на интерпретацията от хората. Обучените невронни мрежи са значително по-тежко комуникират с хората, отколко научените правила.

7.4. Персептрони

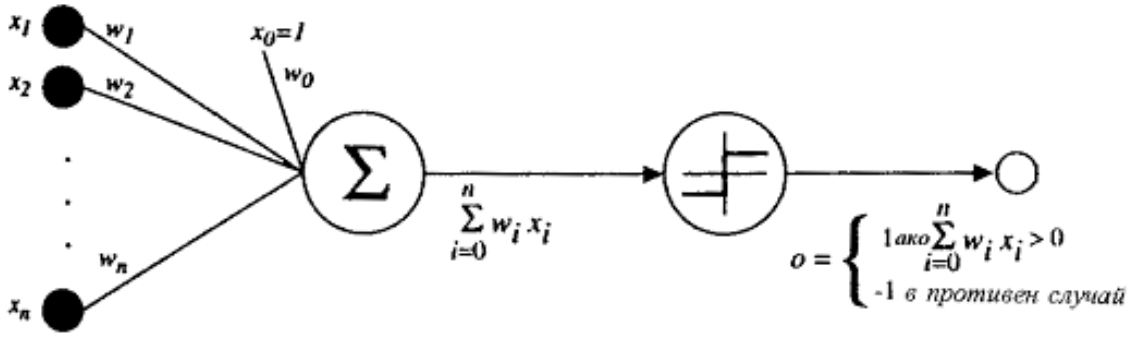
Един тип ИНМ-системи, който се базира върху устройство (възел), наречен *персептрон*, е показан на Фиг. 7-2. Персептронът получава вектор от непрекъснати входове, изчислява тяхната линейна комбинация и след това извежда като изход 1, ако резултатът е по-голям от зададения праг, или –1 – в противен случай. По-точно, при зададените входове $x_1, \ldots, x_n$ изходът $o(x_1, \ldots, x_n)$ се изчислява от персептрона по следния начин:

$$o(x_1,\ldots,x_n) = \begin{cases} 1, & \textit{ако } w_0 + w_1 x_1 + \ldots + w_n x_n > 0 \\ -1, & \textit{в противен случай} \end{cases}$$

където всяко w_i е *тегло* (реална константа), което определя приноса на входа x_i към изхода на персептрона. Величината ($-w_0$) е прагът, който комбинацията от входове $w_1 x_1 + \ldots + w_n x_n$ трябва да надхвърли, за да може персептронът да изведе 1.

За да опростим означения, ще въведем един допълнителен постоянен вход $x_0 = 1$, което ще ни позволи да препишем по-горното неравенство като $\sum_{i=0}^{n} w_i x_i > 0$ или във векторната форма, като $\vec{w} \cdot \vec{x} > 0$. За краткост, понякога ще записваме функцията на персептрона като $o(\vec{x}) = \text{sgn}(\vec{w} \cdot \vec{x})$, където

$$\text{sgn}(y) = \begin{cases} 1, & \textit{ако } y > 0 \\ -1, & \textit{в противен случай} \end{cases}$$



Фиг. 7-2. Персептрон

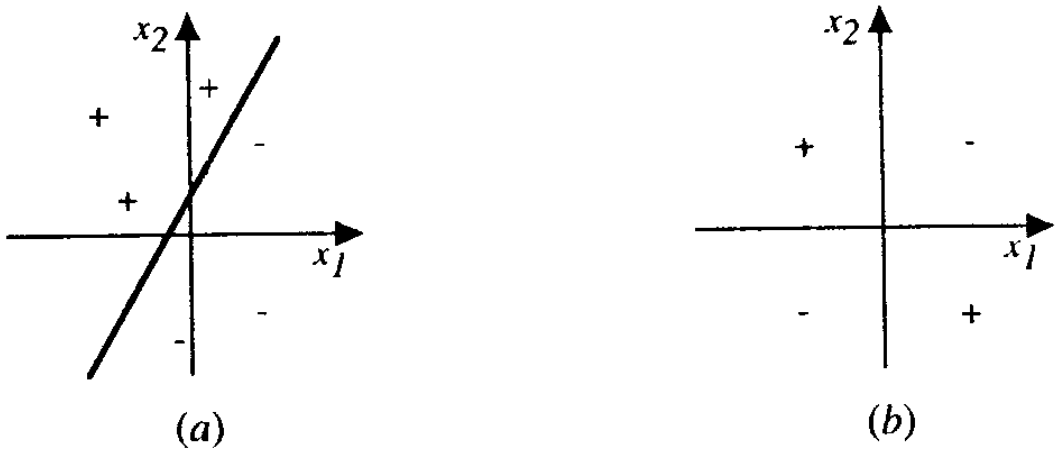
Обучението на персептрона се състои в избора на стойности на теглата $w_0, \ldots, w_n$. Следователно, пространството на хипотези H, разглеждано при обучение на персептрона, е множеството от всички възможни тегловни вектори от реални стойности.

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

7.4.1. Изразителната сила на представяне чрез персептрони

Персептронът може да се разглежда като представянето на повърхността на решение във вид на хиперплоскост в n-мерното пространство от примери (т.е.

точки). Персептронът извежда 1 за примери, лежащи от една страна на тази хиперплоскост, и –1 – за примери, лежащи от другата й страна, както това е показано на Фиг. 7-3. Уравнението на тази повърхнина на решение е $\vec{w} \cdot \vec{x} = 0$. Естествено, че съществуват множествата от положителни и отрицателни примери, които не могат да бъдат разделени чрез никаква хиперплоскост. Тези, които могат да бъдат разделени, се наричат *линейно сепарабелни* (разделими) множества от примери.



Фиг. 7-3. Повърхнина на решения за персептрон с два входа. (a) Множество от обучаващи примери (положителни и отрицателни) и повърхнина на решение на персептрона, която ги класифицира коректно; (b) едно линейно несепарабелно множество от обучаващи примери.

Един персептрон може да се използва за представяне на различни булеви функции. Например, ако предположим, че булевите стойности са 1 (истина) и -1 (лъжа), то един начин за използване на персептрона за имплементиране на логическата функция И (AND) е да установим теглата по следния начин*:* $w_0 = -0.8$, $w_1 = w_2 = 0.5$. Същия персептрон може да бъде направен да представя функцията ИЛИ (OR), ако изменим прага $w_0 = 0.3$. Фактически, И и ИЛИ могат да се разглеждат като специални случаи на функциите *m-от-n*: т.е. функциите, в които най-малко *m* от *n* входа на персептрона трябва да са в състояние ИСТИНА (1). Функцията ИЛИ съответства на $m = 1$, а функцията И – на $m = n$. Всяка от функциите *m-от-n* може да бъде лесно представена чрез използване на един персептрон, в който всички входни тегла са установени на една и съща стойност (например 0.5), а теглото на прага w_0 се избира в зависимост от съответната функция.

Персептроните могат да представят всички примитивни булеви функции: И, ИЛИ, NAND (¬И) и NOR (¬ИЛИ). За съжаление, някои от булевите функции не могат да бъдат представени чрез един единствен персептрон – например функцията XOR (изключващо ИЛИ), чиято стойност е 1, тогава и само тогава, когато $x_1 \neq x_2$. Обърнете внимание, че множеството от линейно несепарабелни обучаващи примери, показани на фигура 7.3 (b) съответства на тази функция XOR.

Способността на персептрони да представят функциите И, ИЛИ, ¬И и ¬ИЛИ е важна, тъй като всяка булева функция може да се представи чрез определена мрежа от свързани възли, базирани върху тези примитиви. Фактически, всяка булева функция може да бъде представена от определена мрежа от персептрони, разположени само в два слоя, в която входовете захранват няколко възела, а изходите на тези възли са входове на второто, финално ниво. Един от способите за имплементация на подобна мрежа е да представим булевата функция в дизюнктивната нормална форма (т.е. като дизюнкцията (ИЛИ) от множество от конюнкциите (И-та) от входовете и техните отрицания. Ясно е, че входът към И-персептрон може лесно да бъде превърнат в своето отрицание само чрез промяна на знака на съответното за този вход тегло.

7.4.2. Обучаващото правило на персептрона

Преди да опишем начина на обучение на мрежата от множество взаимосвързани възли, да разгледаме, как може да се научи тегловният вектор на един единствен персептрон. В този случай задачата за обучение е да определим такъв тегловен вектор, който кара персептронът да произвежда правилния изход ±1 за всеки зададен обучаващия пример.

За решаване на този проблем са известни няколко алгоритъма. Ще разгледаме само два от тях: *правилото на персептрона* и *делта правилото*. Тези два алгоритъма гарантират сходимостта към малко по-различни приемливи хипотези при малко по-различни условия. Алгоритмите са важни за ИНМ, тъй като осигуряват основата за обучение на мрежа, състояща от много възли.

Един от начините да научим приемливия тегловен вектор е да за почнем със случайни тегла, а след това итеративно да прилагаме персептрона към всеки обучаващ пример, модифицирайки теглата всеки път, когато примерът е класифициран неправилно. Този процес се повтаря, итеративно обхождайки всички обучаващи примери толкова пъти, колко е необходимо на персептрона, за да класифицира всички обучаващи примери без грешки. Теглата се модифицират на всяка стъпка в съответствие с *правилото за обучение на персептрона*, което променя теглото w_i, асоциирано с входа x_i, в съответствие с формулата:

$$w_i \leftarrow w_i + \Delta w_i \text{ , където } \Delta w_i \leftarrow \eta(t-o)x_i$$

Тука t е целевият изход за текущ обучаващ пример, o е изходът, генериран от персептрона, а η - една положителна константа, наречена *скоростта на обучение*. Ролята на скоростта на обучение е да отслабва степента, в която теглата се променят на всяка стъпка. Обикновено, тя се установява равна на някоя малка стойност (например 0.1) и понякога се намалява при нарастването на броя на итерациите по настройка на теглата.

Защо това правило за обновление трябва да има сходимост към успешните тегловни стойности? Да предположим, че обучаващият пример е вече правилно класифициран от персептрона. В този случай $(t - o) = 0$, което прави $\Delta w_i = 0$, така че никакви тегла не се обновяват. Да предположим сега, че персептронът извежда –1, когато целевият изход е равен на +1. За да стане изходът на персептрона +1 вместо –1, теглата трябва да се променят, за да увеличат стойността на $\vec{w} \cdot \vec{x}$. Например, ако $x_i > 0$, то увеличаването на w_i ще докара

персептронът по-близо до правилната класификация на този пример. Обърнете внимание, че в този случай обучаващото правило ще увеличи w_i, тъй като всички $(t - o)$, η и x_i са положителни. Например, ако $x_i = 0.8$, $\eta = 0.1$, $t = 1$ и $o = -1$, то обновяването на теглото ще бъде $\Delta w_i = \eta(t - o)x_i = 0.1(1 - (-1))0.8 = 0.16$. От другата страна, *ако* $t = -1$, а $o = 1$, то теглата, асоциирани с положителни x_i ще се намалеят, вместо да се увеличат.

Може да се докаже, че за краен брой прилагания на правилото за обучение на персептрона, описаната по-горе процедура за обучение има сходимост към тегловния вектор, който правилно класифицира всички обучаващи примери, *ако обучаващите примери са линейно сепарабелни* и стойността на η е избрана достатъчно малка (Minsky and Paper 1969). Ако данните не са линейно сепарабелни, сходимостта не се гарантира.

7.4.3. Градиентното спускане и делта правилото

Макар, че правилото на персептрона намира нужния тегловен вектор, когато обучаващите примери са линейно сепарабелни, неговата сходимост може да пропадне, ако те не са линейно сепарабелни. Второто обучаващо правило, наречено *делта правилото*, е създадено да преодолее това затруднение. Ако обучаващите примери не са линейно сепарабелни, делта правилото осигурява сходимост към апроксимацията, най-добре прилягаща към целевото понятие.

Ключовата идея на делта правилото е да се използва *градиентното спускане* за търсене в пространството на хипотези от възможни тегла, за да намери теглата (хипотезата), които най-добре прилягат към обучаващите примери. Това правило е много важно, тъй като градиентното спускане осигурява основа на алгоритъма BACKPROPAGATION, който може да обучава мрежи от множество взаимосвързани възли. То също така е важно, тъй като градиентното спускане може да служи като основа за алгоритми за обучение, които трябва да търсят в пространства на хипотези, съдържащи множество от различни типове непрекъснато параметризирани хипотези.

Делта правилото най-добре се разбира чрез разглеждане на задачата за обучение на *безпраговият персептрон*: т.е. един *линеен възел*, чийто изход o се задава от формулата:

$$o(\vec{x}) = \vec{w} \cdot \vec{x} \quad (7.1)$$

С други думи, линейният възел съответства на първия етап на персептрона - без прага (виж Фиг. 7-2).

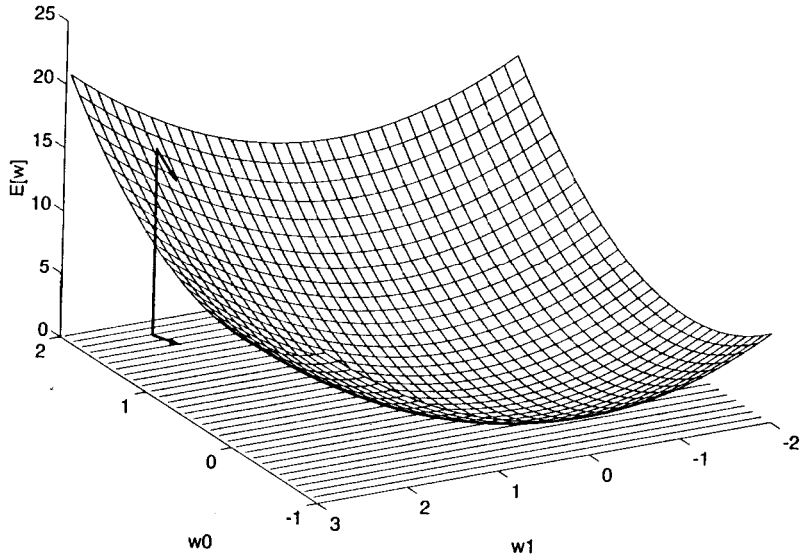
За да изведем правилото за научаване на тегла, ще въведем мярката за грешка на хипотеза (тегловен вектор) при обучение, по отношение на обучаващите примери:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \quad (7.2)$$

където D е множеството от обучаващи примери, t_d е целевият изход, а o_d е изходът на линейния възел за обучаващия пример d. Тука E представена като функцията на $\vec{w}$, тъй като линейният изход o зависи от този тегловен вектор. Ясно е, че E зависи и от конкретното множество от обучаващите примери, но в случая смятаме, че те са фиксирани при обучение, така че няма смисъл явно да бъдат записвани като аргумент на E.

6.4.4. Визуализация на пространство от хипотези

За разбиране на алгоритъма за градиентното спускане помага визуализацията на цялото пространство на хипотези от тегловни вектори и асоциираните с тях стойностите на мярката E, както това е показано на Фиг. 7-4. Осите w_0 и w_1 са представят възможните стойности на два тегла на един прост линеен възел. Плоскостта w_0, w_1 представя пълното пространство от хипотези. Вертикалната ос указва грешката E относително някой фиксиран набор от обучаващите примери. Повърхнината на грешката, показана на фигурата, резюмира предпочтителността на всеки тегловен вектор в пространството на хипотези (за предпочитане е хипотезата с най-малката грешка). При избрания начин за дефиниране на E в случая на линейни възли тази повърхнина на грешката е винаги параболична с един единствен глобален минимум (виж формула 7.2). Конкретната парабола ще зависи, естествено, от конкретното множество от обучаващите примери.



Фиг. 7-4. Повърхнина на грешката на различни хипотези

Търсенето чрез градиентното спускане определя тегловния вектор, който минимизира E, чрез стартиране от произволен начален тегловен вектор и повтарящо се неговото модифициране на малки стъпки. На всяка стъпка тегловният вектор се променя в направление, което осигурява най-стръмното спускане по повърхнината на грешката, показана на Фиг. 7-4. Този процес се продължава, докато не бъде достигнат глобален минимум.

7.4.5. Извод на правилото за градиентното спускане

Как може да се изчисли направлението на най-стръмното спускане по повърхнината на грешката? То може да се намери чрез изчисляване на производната на E по отношение на всеки компонент на вектора $\vec{w}$. Този вектор от производни се нарича *градиент* на E по $\vec{w}$ и се означава с $\nabla E(\vec{w})$.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_o}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n} \right] \quad (7.3)$$

Обърнете внимание, че $\nabla E(\vec{w})$ е вектор, чиито компоненти са частни производни на E по всеки от w_i. При интерпретацията като вектор в пространството на теглата, градиентът указва направлението, което дава най-бързото изкачване по E. Обратният вектор дава направление на най-стръмното спускане. Например, стрелката на Фиг. 7-4. показва отрицателният градиент - $-\nabla E(\vec{w})$ за една конкретна точка от плоскостта w_0, w_1.

Тъй като градиентът определя направлението на най-стръмното спускане по E, обучаващото правило за градиентното спускане приема вид:

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}, \text{ където } \Delta\vec{w} = -\eta \nabla E(\vec{w}) \quad (7.4)$$

Тука η е положителна константа, наречена скоростта на обучение, която определя размер на стъпката при градиентното спускане. Отрицателният знак присъства, тъй като искаме да придвижим тегловния вектор в посока, в която E *намалява*. Правилото може да се препише и в по-компонентната форма:

$$w_i \leftarrow w_i + \Delta w_i, \text{ където } \Delta w_i = -\eta \frac{\partial E}{\partial w_i} \quad (7.5)$$

която показва, че най-стръмното спускане се постига чрез промяна на всяка от компонентите w_i на $\vec{w}$ в пропорцията $\frac{\partial E}{\partial w_i}$.

За построяване на реалния алгоритъм за итеративно обновяване на теглата съгласно уравнение (7.5), е необходим ефективният начин за изчисляване на градиента на всяка стъпка. Това не е трудно. Векторът от производни $\frac{\partial E}{\partial w_i}$, който формира градиента, може да се получи чрез диференциране на E от уравнение (7.2):

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 = \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 = \\ &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) = \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) = \\ &= \sum_{d \in D} (t_d - o_d)(-x_{i\,d}) \end{aligned} \quad (7.6)$$

където x_{id} означава входния компонент x_i на обучаващия пример d. Подставяйки (7.6) в уравнение (7.5) получаваме правилото за обновяване на тегла при градиентното спускане:

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id} \quad (7.7)$$

И така, алгоритъмът за градиентното спускане, използван за обучение на линейни възли, се състои в следното: избира се случаен начален тегловен вектор. Линейният възел се прилага към *всички* обучаващите примери, след това се изчисляват Δw_i за всяко тегло съгласно уравнение (7.7). Теглото w_i се обновява чрез добавяне на Δw_i, след което процесът се провтаря. Този алгоритъм е приведен в Таблица 7-1. Тъй като повърхнината на грешката съдържа само един глобален минимум, алгоритмът има гарантирана сходимост към тегловния вектор, който минимизира грешката, не зависимо от това, дали обучаващите примери са линейно сепарабелни или не, при условието, че стойността на η е избрана достатъчно малка. Ако скоростта на обучение е прекалено голяма, търсенето чрез градиентното спускане има опасност да прескочи минимума на повърхността на грешката. По тази причина една общоприета модификация на описания алгоритъм се състои в постепенно намаляване на стойността на η при увеличаване на броя на стъпките, извършвани при градиентното спускане.

Градиентно-спускане(*обучаващи_примери, η*)

Всеки обучаващ пример е двойка във вид $< \vec{x}, t >$, където $\vec{x}$ е вектор на входните величини, а t е величина на целевия атрибут. η е скоростта на обучение.

- Инициализирай всеки w_i с някакво случайно малко число.
- **Докато** не бъдат изпълнени условия за спиране **Направи**
  - Инициализирай всеки $\Delta w_i \leftarrow 0$.
  - За всеки $< \vec{x}, t >$ от *обучаващи_примери* **Направи**
    - Използвай $\vec{x}$ като вход на линейния възел и изчисли неговият изход o.
    - За всяко тегло на линеен възел w_i **Направи**

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i \quad (*)$$

  - За всяко тегло на линеен възел w_i **Направи**

$$w_i \leftarrow w_i + \Delta w_i \quad (**)$$

Таблица 7-1. Алгоритъм за обучение на линейния възел чрез градиентното спускане

7.4.6. Стохастична апроксимация на градиентното спускане

Градиентното спускане е една много важна обща парадигма за самообучение. Това е стратегията за търсене в голямо или безкрайно пространство от хипотези, която може да се прилага, когато 1) пространството на хипотези съдържа непрекъснато параметризирани хипотези (например, теглата на линейния възел) и 2) грешката е диференцируема по тези параметри на хипотези. Основните практически затруднения с прилагане на градиентното спускане са 1) сходимостта към локалния минимум може да бъде твърде бавна (т.е. тя може да изисква няколко хиляди стъпки от спускането) и 2) ако повърхнината на

грешката има няколко локални минимума, то няма гаранция, че процедурата ще намери глобален минимум.

Една от често използваните вариации на метода за градиентното спускане е предназначена да преодолее тези затруднения. Тя се нарича *инкрементално градиентно спускан*е или, алтернативно, *стохастично градиентно спускане*. Докато обучаващото правило на градиентното спускане, представено от ф-ла (7.7), изчислява обновените тегла след сумиране по *всички* обучаващи примери, идеята на стохастичното градиентно спускане е да апроксимира това градиентно търсене чрез инкременталното обновяване на теглата след изчисляване на грешката за *всеки* обучаващ пример. Модифицираното обучаващо правило напомня това от ф-ла (7.7) с изключението, че итерациите стават по всеки обучаващ пример, а обновяване на теглата става съгласно формулата:

$$\Delta w_i = \eta(t-o)x_i \quad (7.10)$$

където *t, o* и x_i са, съответно, целевата стойност, изходната стойност на възела и i-я вход на обучаващия пример. За да модифицираме алгоритъма за градиентното спускане от Таблица 7-1 с цел имплементация на стохастичната апроксимация, ф-ла (**) трябва да бъде премахната, а ф-ла (*) да бъде заменена с $w_i \leftarrow w_i + \eta(t-o)x_i$. Стохастичното градиентно спускане може да се разбере по-добре, ако разгледаме другата функция за грешката - $E_d(\vec{w})$, определена за всеки отделен обучаващ пример *d* по следния начин:

$$E_d(\vec{w}) \equiv \frac{1}{2}(t_d - o_d)^2 \quad (7.11)$$

където t_d и o_d са целевата и изходната стойности на обучаващия пример *d*. Стохастичното градиентно спускане прави итерация по обучаващи примери *d* от *D*, като на всяка итерация променя теглата в съответствие с градиента по отношение на $E_d(\vec{w})$. Последователността от тези обновявания на теглата дава разумната апроксимация към градиента по отношение на оригиналната функция за грешката $E(\vec{w})$. Чрез установяване на размера на стъпката за градиентното спускане η достатъчно малка, стохастическото градиентно спускане може да бъде направено да апроксимира истинското градиентно спускане с произволна точност. Ключовите разлики между стандартното градиентно спускане и стохастическото градиентно спускане са:

- При стандартното градиентно спускане грешката се сумира по всички примери *преди* да бъдат обновени теглата, докато при стохастичното градиентно спускане теглата са обновяват *след* проверката на всеки обучаващ пример.
- Сумирането по всички примери при стандартното градиентно спускане изисква по-големи изчисления за всяка стъпка по обновяване на теглата. От друга страна, тъй като се използва истинският градиент, стандартното градиентно спускане често се използва с по-голям размер на стъпката на обновяването на теглото, от колко при стохастичното градиентно спускане.
- В случаи на наличието на няколко локални минимума по отношение на $E(\vec{w})$, стохастичното градиентно спускане може понякога да избегне попадане в тези локални минимуми, тъй като използва различни $\nabla E_d(\vec{w})$ вместо $\nabla E(\vec{w})$, за да управлява търсенето.

На практиката широко се използват и двата метода.

Обучаващото правило от ф-ла (7.7) или (7.10) е известно като *делта правилото*, обаче има и други имена, например, правилото на най-малки квадрати (LMS-правило), Adaline-правило или правило на Widrow-Hoff. Делта правилото е подобно на правилото за обучение на персептрона, разгледано в раздела 7.4.2, обаче те са различни, тъй като в делта правилото *o* означава изход на линейния възел: $o(\vec{x}) = \overline{w} \cdot \vec{x}$, докато в правилото за обучение на персептрона *o* означава праговият изход: $o(\vec{x}) = \text{sgn}(\overline{w} \cdot \vec{x})$.

Макар, че делта правилото е представено като метод за научаване на тегла за безпрагови линейни възли, то може лесно да бъде използвано и за обучение на прагови персептронни възли. Да предположим, че $o(\vec{x}) = \overline{w} \cdot \vec{x}$ е изходът на безпраговия линеен възел, а $o' = \text{sgn}(\overline{w} \cdot \vec{x})$ е резултат от прилагането на прага към *o,* както това става в персептрона. Сега, ако искаме да обучим персептрона да покрива обучаващите примери с целевите стойности ±1 за *o'*, можем да използваме същите целевите стойности и примери, за да обучим *o*, използвайки делта правилото. Ясно е, че ако безпраговият изход *o* може да бъде научен да пасва към тези стойности перфектно, то и праговият изход *o'* също ще пасва към тях (тъй като *sgn(1) = 1* и *sgn(-1) = -1*). Дори, когато целевите стойности не могат да бъдат покрити перфектно, праговата стойност на *o'* ще коректно пасва на целевите стойности ±1 всеки път, когато изходът на линейния възел *o* има правилен знак. Обърнете внимание, че докато тази процедура добре научава теглата, минимизиращи грешката в изхода на линейния възел *o*, същите тегла не е задължително ще минимизират броя на обучаващите примери, неправилно класифицирани от праговият изход *o'*.

И така, ние разгледахме два подобни алгоритъма за итеративно научаване на теглата на персептрона. Основната разлика между тях е, че правилото за обучение на персептрона обновява теглата на базата на грешката от *праговия изход* на персептрона, докато делта правилото обновява теглата, базирайки се на грешката от *безпраговата линейна комбинация от изходите*.

Разликата между тези два обучаващи правила се отразява върху различните свойства на сходимостта. Правилото за обучение на персептрона се схожда след определено крайно множество от итерации към хипотезата, която перфектно класифицира обучаващите данни, *при условие, че обучаващите примери са линейно сепарабелни*. Делта правилото се схожда само асимптотически към хипотеза, минимизираща грешката, възможно изисквайки и неограничено време, обаче се схожда *не зависимо от това, дали обучаващите примери са линейно сепарабелни или не*.