8 Samoorganizacja w warunkach konkurencji, Kwantyzacja przestrzeni, Gaz neuronowy

Najpierw przedstawimy ogólne zasady na których opiera się działanie sieci samoorganizujących się, czyli uczących się bez nadzoru.

Potem omówimy metodę *gazu neuronowego*, gdzie neurony są całkowicie luźne i nie tworzą żadnej sieci; oraz metodę *Som*, czyli samoorganizujących się map Kokonena, gdzie neurony są zorganizowane w sieć nazywaną *mapą*, a mapa ta odzwierciedla topologiczne sąsiedztwo neuronów w przestrzeni R^d.

8.1 Co to znaczy: samoorganizująca się sieć. Ogólne zasady

Sieci samoorganizujące się na zasadzie konkurencji składają się najczęsciej z warstwy wejściowej i jednej warstwy dość luźno (lub wcale nie) powiązanych ze sobą neuronów. Określenie 'sieć' staje się dla pewnych konfiguracji neuronów dość problematyczne – i wtedy należałoby mówić raczej o 'kolekcji' neuronów, a nie o 'sieci'.

Każdy neuron ma swój wektor wag o liczbie składowych d równej liczbie cech (zmiennych) składających się na wektor wejściowy (wektor danych dla jednego 'osobnika' lub jednej 'próbki'). Wektor ten jest jednocześnie interpretowany jako punkt w przestrzeni cech R^d.

Ustalamy, że liczba neuronów wynosi M. Każdy z neuronów ma przypisany wektor wagowy $\mathbf{w}_i$, $i = 1, \ldots, M$. Każdy z wektorów wagowych może być interpretowany jako punkt w R^d. Wektory wagowe są również nazywany prototypami lub wektorami referencyjnymi.

Na początku wektory wagowe charakteryzujące neurony mają przydzielone wartości dość przypadkowe (ale oczywiście należące do R^d). W trakcie procesu uczenia neurony starają się przesunąć w kierunku zagęszczeń danych i wytworzyć swoje strefy wpływów (dominacji). Strefa wpływów neuronu jest określona przez punkty indywidualne danych znajdujące się najbliżej tego neuronu, – najbliżej w sensie przyjętej metryki. Domyślną metryką jest odległość euklidesowa, ale można wprowadzić inne metryki, o czym powiemy dalej.

Punkt indywidualny danych jest zaliczany do strefy wpływów tego neuronu, który jest najbliższy temu punktowi indywidualnemu.

Uczenie odbywa się na zasadzie konkurencji. Oznacza to, że neurony rywalizują ze sobą o możliwość zawłaszczenia sobie punktów indywidualnych danych.

Na koniec procesu uczenia cała przestrzeń cech okazuje się być podzielona na rozdzielne i spójne strefy wpływów poszczególnych neuronów. Jednocześnie można zaobserwować *pogrupowanie danych*: Mianowicie cały zbiór danych został podzielony na skupienia reprezentowane przez poszczególne neurony. Mówi się również, że skupienia te pozostają pod dominacją poszczególnych neuronów tworzących coś w rodzaju centrów tych skupień. Neurony te (a właściwie ich wektory wagowe) zajmują pozycje centralne w każdym skupieniu. W pewnych okolicznościach każdy taki neuron może być uważany za reprezentanta wszystkich punktów indywidualnych znajdujących sie w strefie wpływów tego neuronu.

Według Tadeusiewicza przy rozpatrywaniu sieci samoorganizujących się pojawiają się dwie nowe zasady działania sieci, których nie było w klasycznych sieciach typu perceptronu: mianowicie zasada *koherencja* i *kolektywność*.

Zasada *koherencji* postuluje, żeby grupować dane w pewne klasy podobieństwa. Grupowanie następuje na zasadzie samoorganizacji. Proces samoorganizacji jest automatyczny i spontaniczny, bez nauczyciela. Zakłada się, że komplet potrzebnych informacji jest zawarty w samych danych, z których jedne są podobne do siebie nawzajem, a inne nie.

Zasada *kolektywności* powoduje, że to, co rozpoznaje jeden neuron, zależy w dużej mierze od tego, co poznają inne neurony.

Zbiorowość neuronów (czyli ich kolektyw) może w sposób pełniejszy i bogatszy przetwarzać informacje, aniżeli każdy z neuronów wzięty z osobna.

Zbiorowość odpowiednio powiązanych i współpracujących elementów stwarza możliwość uzyskania nowych form zachowania i nowych postaci działań znacznie bogatszych, niż by można oczekiwać biorąc pod uwagę każdy z elementów tej zbiorowości z osobna. Przykłady z przyrody: Pszczoły, mrówki – jako kolektyw są zdolne do celowych działań. Poszczególna pszczoła lub mrówka jest ważna tylko jako element całości.

8.1.1 Miary odległości między wektorami

Odległość euklidesowa : $d(\mathbf{x}, \mathbf{w}) = \sqrt{\sum_{j=1}^{d}(x_j - w_j)^2}$

Iloczyn skalarny : $d(\mathbf{x}, \mathbf{w}) = ||\mathbf{x}||\ ||\mathbf{w}||\ \cos(\mathbf{x}, \mathbf{w})$ [1]

Manhattan czyli w normie L1 : $d(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{d} |x_j - w_j|$

Max czyli w normie Czebyszewa : $d(\mathbf{x}, \mathbf{w}) = \max_j |x_j - w_j|$

Cytat z Osowskiego ([2], str 251):

> Jeżeli używamy odległości euklidesowej, to utworzone w wyniku uczenia obszary stanowiące strefy wpływów poszczególnych neuronów są równoważne obszarom (mozaice) Voronoia. Zastosowanie innej miary przy samoorganizacji kształtuje podział stref wpływów inaczej. W szczególności zastosowanie iloczynu skalarnego bez normalizacji wektorów może prowadzić do niespójnego podziału przestrzeni, przy którym występuje kilka neuronów w jednym obszarze, a w innym nie ma żadnego. Przypadek taki może wystąpić dla wektorów wejściowych dwuwymiarowych i braku normalizacji.

Na poparcie swojej wypowiedzi Osowski przytacza rysunek 8.3 (na stronie 252 swojej książki), na którym pokazuje na przykładzie dwuwymiarowym ukształtowanie się obszarów dominacji 25 neuronów przy czterech wymienionych metrykach.

8.1.2 Problem normalizacji

Problem ten jest szczególnie ważny, jeśli liczba cech jest stosunkowo niewielka. Dla wymiaru $d > 200$ normalizacja nie odgrywa większej roli. Cytowane za Osowskim [2], str. 253.

[1] dla wektorów określonych w pierwszej ćwiartce

8.1.3 Adaptacja wag w warunkach konkurencji

Na początku uczenia neurony otrzymują przypadkowe wartości wag, a tym samym przypadkowe pozycje w przestrzeni cech R^d. Następnie, na podstawie prezentowanych danych, neurony kierują się do obszarów zajmowanych przez te dane i 'uczą' się je rozpoznawać.

Prezentacja danych polega na sekwencyjnym pokazywaniu sieci wektorów danych branych w porządku zrandomizowanym. Aktualizacja wag może następować po każdej prezentacji indywidualnego wektora danych, lub dopiero wtedy, gdy zaprezentowano ich całą paczkę (wszystkie wektory z tablicy danych X).

Zajmiemy się najpierw przypadkiem, gdy aktualizacja następuje po każdej indywidualnej prezentacji wektora danych. Drugi przypadek, nazywany uczeniem wsadowym, jest omawiany po krótce w sekcji 8.1.9.

Uczenie sekwencyjne polega na tym że kolejno w krokach $k = 1, ..., k_{max}$ prezentuje się sieci losowo wybrany z danych wektor $\mathbf{x}$, nazywany również próbką. Wektor prezentowany w k-tym kroku bywa oznaczany symbolem $\mathbf{x}(k)$. Zmienna k jest nazywana *numerem iteracji* lub *numerem konkurencji*.

Po każdej prezentacji wzorca $\mathbf{x}(k)$ zwycięzcą zostaje tylko jeden neuron, najbliższy zaprezentowanemu wektorowi $\mathbf{x}(k)$. Bliskość jest liczona w sensie ustalonej metryki. Neuron ten otrzymuje wskaźnik c (c – jak *conqueror* i jak punkt *centralny*; właściwie należałoby pisać $c = c(k)$, bo jest to zwycięzca w k-tej konkurencji; punkt ten w dalszym ciągu staje się centrum sąsiedztwa).

Neuron – zwycięzca ma prawo uaktualnić swoje wagi według jednej z dwóch zasad: *WTM* – Winner Takes Most, i *WTA* – Winner Takes All.

Zasada WTA. Tylko neuron zwycięski ma prawo uaktualnić swoje wagi, tzn. przysunąć się do wektora $\mathbf{x}(k)$. Inne neurony pozostają na swoich miejscach. Wagi zwycięskiego neuronu mogą zostać uaktualnione np. na podstawie następującego wzoru:

$$\mathbf{w}_c(k+1) = \mathbf{w}_c(k) + \eta(k)[\mathbf{x}^T(k) - \mathbf{w}_c(k)]. \tag{1}$$

Wzór ten mówi, że wektor wagowy zwycięskiego neuronu zostaje przysunięty do wektora $\mathbf{x(k)}$. Wielkość przemieszczenia zależy przedewszystkim od wielkości różnicy między położeniem $\mathbf{w}_c$ oraz $\mathbf{x}(k)$ (różnicę tę wyrażamy w odpowiedniej metryce). Ponadto wielkość przemieszczenia zależy również od współczynnika $\eta(k)$. nazywanego współczynnikiem uczenia. Współczynnik $\eta(k)$ jest na ogół malejącą funkcją numeru iteracji k. Wspólczynnik ten zostanie bliżej omówiony w dalszych ciągu wykładu.

Zasada WTM. Jeżeli obowiązuje zasada WTM to ulegają zmianie nie tylko wagi neuronu–zwycięzcy, ale również jego 'sąsiadów'. Sąsiedztwo neuronu c określa się zbiorem $\mathcal{N}_c(k)$, który jest wyznaczony na podstawie przyjętej (a tym samym znanej) funkcji sąsiedztwa. Wrócimy do tych pojęć trochę dalej, i wtedy te pojęcia zostaną zdefiniowane bardziej dokładnie. Teraz sformułujemy zasadę WTM:

Tylko wagi neuronu–zwycięzcy i jego sąsiadów zostają uaktualnione według zasady:

$$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) + \eta(k)G(i, c, k, \boldsymbol{\theta})[\mathbf{x}^T(k) - \mathbf{w}_i(k)], \quad i \in \mathcal{N}_c(k)\,, \tag{2}$$

natomiast wagi innych neuronów pozostają niezmienione.

Funkcja G występująca we wzorze (2) oznacza czynnik wynikający z sąsiedztwa między wektorami $\mathbf{w}_i$ i $\mathbf{w}_c$. W niektórych źródłach, szczególnie w kontekscie sieci Kohonena ([3, 4, 5]), funkcja ta jest pisana w formie $h_{ci}(k)$ wskazującej wyraźnie na zależność tej funkcji od numeru iteracji k. Obie funkcje (tj. G i h) wskazują na stopień (intensywność) sąsiedztwa między wektorami $\mathbf{w}_c$ i $\mathbf{w}_i$. Obie funkcje przyjmują wartości z przedziału [0,1].

Wartości obu funkcji (G i h) zależy przede wszystkim od tego, jak daleko znajduje się i-ty neuron od aktualnego neuronu–zwycięzcy. W niektórych algorytmach nie bierze się pod uwagę fizycznej odleglości w przestrzeni R^d, lecz jedynie miejsce w rankingu odległości neuronów od neuronu-zwycięzcy lub od zaprezentowanego wektora $\mathbf{x}(k)$.

Funkcja G może zależeć również od innych parametrów, które we wzorze (2) zaznaczono ogólnie jako wektor parametrów $\boldsymbol{\theta}$; mogą nimi być pewne stałe określające, jak szybko zmniejszają się wartości funkcji G, gdy zmniejsza się promień sąsiedztwa. Wielkości parametrów $\boldsymbol{\theta}$ mogą zależeć również od numeru iteracji, tzn. może być: $\boldsymbol{\theta} = \boldsymbol{\theta}(k)$.

Określanie sąsiedztwa neuronu oraz funkcji G i h zostanie omawiane bliżej w podrozdziale 8.1.6.

W efekcie współzawodnictwa wyrażonego zasadami *WTA* i *WTM* następuje samoorganizacja procesu uczenia. Neurony dopasowują swoje wagi w ten sposób, że przy prezentacji grup wektorów wejsciowych zbliżonych do siebie zwycięża zawsze ten sam neuron.

Neuron, poprzez zwycięstwo we współzawodnictwie rozpoznaje swoją kategorię.

Ogólnie można powiedzieć, że: Przy podaniu na wejście sieci wielu wektorów zbliżonych do siebie będzie zwyciężać ciągle ten sam neuron, w wyniku czego jego wagi będą odpowiadać uśrednionym wartościom wektorów wejściowych, dla których dany neuron był zwycięzcą.

Neurony nie wygrywające nie zmieniają swoich wag. Mówi się, że pozostają one martwe.

8.1.4 Przykład uczenia WTA wektorów leżących na kole

Zasadę uczenia z konkurencją WTA ilustruje rysunek 7.2 zaczerpnięty z książki Osowskiego ([2], str 30.

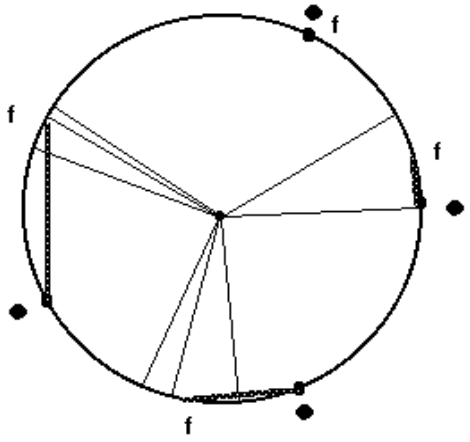
Zbiór danych składa się z 8 wektorów; każdy z tych wektorów jest dwuwymiarowy i ma długość jednostkową, wobec tego przestrzeń danych może być odwzorowana na okręgu (w przypadku wektorów danych wielowymiarowych o długości 1 – wektory danych mogłyby być odwzorowane na sferze).

Mamy również 4 neurony z przypisanymi do nich wektorami wagowymi $\mathbf{w_i} = [w_{i1}, w_{i2}]^T,\ i = 1, 2, 3, 4$, każdy z nich o długości jeden, tj. $||\mathbf{w}_i|| = 1,\ \forall_i$.

Neurony te miały się, w procesie konkurencyjnym uczenia, nauczyć prezentowanego zbioru danych i zaadaptować swoje wagi w ten sposób, aby możliwie dobrze reprezentować te dane.

Uczenie było konkurencyjne (WTA), a elementy zbioru danych prezentowano neuronom w sumie 320 razy.

Po prezentacji kolejnego wzorca zwyciezcą zostawał neuron najbliższy temu wzorcowi (najbliższy w sensie odległości kątowej mierzonej po okręgu koła).



Rysunek 8.1. Uczenie się z konkurencją. Jeden neuron nigdy nie zwyciężył, wskutek czego nie zdołał wytworzyć swojej strefy wpływów w prezentowanym zbiorze danych i nie uległ żadnej adaptacji. Początkowa pozycja wag jest zaznaczona na zewnątrz jednostkowego koła dużą kropką; końcowa pozycja – literą f (final).

Po zakończeniu prezentacji okazało się, że w poszczególnych prezentacjach zwycięzcami zostawały tylko trzy neurony; i tylko one uzyskały przywilej zaadaptowania swoich wag. Jeden z neuronów nigdy nie zwyciężył, wobec czego jego wagi pozostały niezmienione. Pozostałe neurony uzyskały swoje strefy wpływów. W rezultacie neurony te uplasowały się w okolicy średnich pewnych grupek danych i stały się reprezentantami tych grupek danych.

8.1.5 Kwantyzacja przestrzeni cech R^d i wieloboki Voronoia

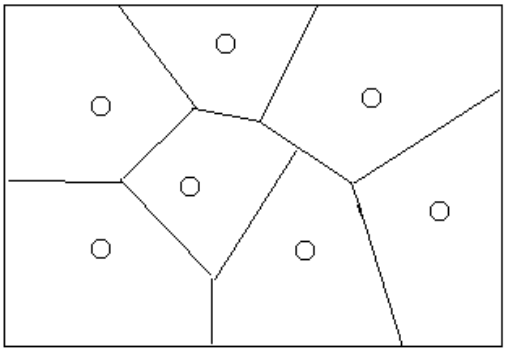
Opisany w poprzedniej sekcji przykład uczenia się neuronów dla danych należących do R^2 może być zrealizowany dla danych należących do dowolnego zbioru punktów w R^d. W trakcie procesu uczenia neurony starają się ulokować pośród tych punktów–danych i stać się ich reprezentantami. W ten sposób cała przestrzeń zostaje podzielona na obszary atrakcji odzwierciedlające strefy wpływów poszczególnych neuronów. W przypadku używania metryki euklidesowej sa to obszary wypukłe, które są nazywane są czasem wielobokami Voronoia, a reprezentujące je wektory – wektorami Voronoia lub wektorami kodowymi. Obszary Voronoia V_i są zdefiniowane jako miejsce geometryczne punktów $\mathbf{x}$ spełniających następujący warunek:

$$V_i = \{\mathbf{x} : \ ||\mathbf{w}_i - \mathbf{x}|| < ||\mathbf{w}_j - \mathbf{x}||, \ i \neq j\}$$

Przykład wieloboków Voronoia jest pokazany na rysunku 8.2 – gdzie mamy przestrzeń danych określoną jako prostokąt.

Podkreślmy tutaj, że używanie różnych metryk (tj. korzystanie z różnych definicji odległości między dwoma punktami leżącymi w przestrzeni danych) daje różne postaci obszarów atrakcji. Przykład rozkładu takich obszarów atrakcji - przy użyciu czterech różnych metryk – jest pokazany w książce Osowskiego, str. 252.

Po wykształceniu stref wplywów poszczególnych neuronów cała przestrzeń danych zostaje skwantowana; co oznacza, żezostaje podzielona na rozłączne obszary. Wtedy wektory danych należące do jednego obszaru Voronoia mogą być zgrubsza zastąpione odpowiadającym temu obszarowi wektorem (słowem kodowym).



Rysunek 8.2. Wieloboki Voronoia utworzone przez 7 wektorów kodowych na prostokącie z metryką euklidesową.

8.1.6 Niektóre zasady określania sąsiedztwa

Sąsiedztwo neuronu c będziemy oznaczać $\mathcal{N}_c$ (od neighbourhood). Jeżeli chcemy wyraźnie napisać, że jest to sąsiedztwo neuronu c który zwyciężyl w k-tej iteracji, to zapiszemy

$$\mathcal{N}_c = \mathcal{N}_c(k).$$

Zbiór $\mathcal{N}_c$ zawiera zbiór indeksów neuronów.

Na ogół sąsiedztwo $\mathcal{N}_c(k)$ charakteryzuje się pewnym promieniem, który maleje wraz z upływem czasu uczenia t, czyli w miarę zwiększania się wskaźnika k.

Jednak oprócz samego faktu przynależności do sąsiedztwa określa się jeszcze coś w rodzaju intensywności przynależności do danego sąsiedztwa. Funkcja intensywności przynależności do $\mathcal{N}_c$ jest oznaczana symbolem $G(i, c, k, \boldsymbol{\theta}$ lub $h_{ci}(k)$. Funkcje te opierają się na odległości neuronu i od zwycięskiego neuronu c. Funkcje te mają bardzo często postać funkcji radialnych scentrowanych w punkcie $\mathbf{w}_c$.

Najbardziej popularnymi funkcjami sąsiedztwa są *bubble* i *gaussian*. Są one pokazane na rysunku 8.3. Przykładowo funkcja radialna *gaussian* przyjmuje postać ([2], str. 257):

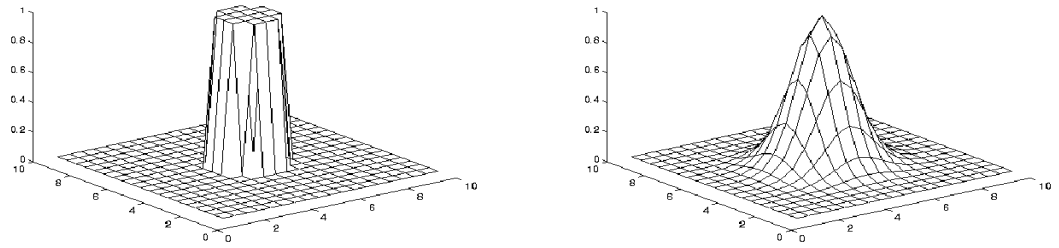
$$h_{c,i}^{gauss} = exp\left(-\frac{||\mathbf{w}_i - \mathbf{w}_c||}{2\sigma^2(t)}\right),$$

gdzie wektory $\mathbf{w}_i$ oraz $\mathbf{w}_c$ należy rozumieć jako wektory z wartościami otrzymanymi w k-tej iteracji.

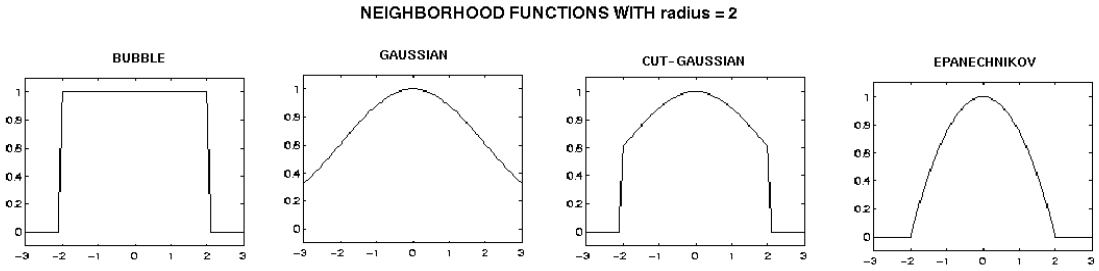
Jeszcze inne funkcje sąsiedztwa (*cut-gaussian* i *Epanechnikov*) są pokazane na rysunku 8.4.

Funkcje sąsiedztwa określamy najczęściej w ten sposób, aby przyjmowały one wartości z przedziału (0,1] lub nawet przyjmowały tylko dwie wartości: 1, gdy dany element należy, i 0, gdy nie należy do sąsiedztwa. Funkcja *gaussian* określa przynależność jako liczbę $\in [0, 1]$, która może być interpretowana jako pewien stopień przynależności rozmytej.

Niech $h_{ci}(k)$ oznacza wartość funkcji sąsiedztwa neuronu nr i względem zwycięskiego neuronu c – wyznaczoną podczas k-tej iteracji. Formułę *WTM* określoną wzorem (2) możemy zapisać również za pośrednictwem wprowadzonej funkcji h (pamiętamy, że wektory $\mathbf{x}$ stanowiące wiersze tablicy danych $\mathbf{X}$ zostały przez nas określone jako wektory wierszowe, natomiast wektory wag $\mathbf{w}_i$ według powszechnie



Rysunek 8.3. Funkcje sąsiedztwa: bubble i gaussian określone na płaszczyźnie. Funkcja bubble wyznacza sąsiedztwo w sposób ostry: 1 - tak, 0 - nie; funkcja gaussian w sposób łagodny jako liczbę z przedziału (0, 1].



Rysunek 8.4. Jednowymiarowe funkcje sąsiedztwa dla promienia R=2: bubble, gaussian, cut-gaussian, Epanechnikov.

stosowanej konwencji są wektorami kolumnowymi):

$$\mathbf{w}_i(k+1) = \mathbf{w}_i(k) + \eta(k) \cdot h_{ci}(k) \cdot [\mathbf{x}^T(k) - \mathbf{w}_i(k)], \quad \star \tag{3}$$

gdzie $\star$ $\eta(k)$ oznacza współczynnik uczenia - iszemy o nim w sekcji 8.6.
$\star$ c oznacza numer wektora–zwycięzcy, tj. znajdującego się najbliżej prezentowanego w k-tym kroku wektora $\mathbf{x}(k)$,
$\star$ wartość funkcji $h_{ci}(k)$ określa, w jakim stopniu należy uwzględnić przynależność neuronu i do sąsiedztwa zwycięskiego neuronu $c = c(k)$.

W szczególności, funkcja $h = h_{ci}(k)$, może być określona w następujący sposób:

$$h_{ci}(k) = \begin{cases} 1 & \text{gdy } \; i \in N_c(k)\,, \\ 0 & \text{gdy } \; i \notin N_c(k)\,. \end{cases}$$

Funkcja $h_{ci}(k)$ może też zależeć bezpośrednio od odległości $D = D(\mathbf{w}_i, \mathbf{w}_c)$ między wektorami $\mathbf{w}_i$ i $\mathbf{w}_c$, np.

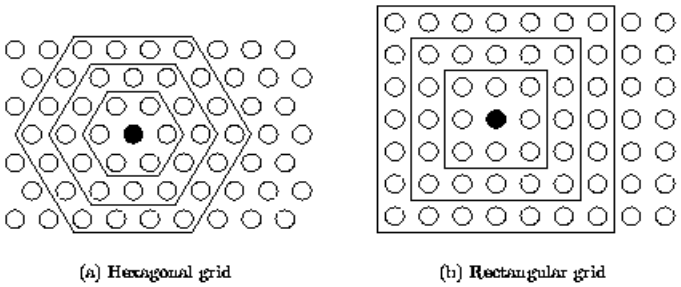
$$h_{ci}(k) = \begin{cases} \mathrm{g}(D(c,i)) & \text{gdy } \; i \in N_c(k)\,, \\ 0 & \text{gdy } \; i \notin N_c(k)\,, \end{cases}$$

gdzie $g(\cdot)$ jest funkcją malejącą (dokładniej: nierosnącą) swojego argumentu. W szczególnym przypadku mogą to być np. funkcje *bubble* lub *gaussian* pokazane na rysunku 8.3.

8.1.7 Przykłady określania sąsiedztwa na płaszczyźnie

Jak wiemy, przy uczeniu konkurencyjnym zwycięża tylko jeden neuron (jego numer oznaczaliśmy indeksem c), ale neurony znajdujące się w sąsiedztwie neuronu-

zwycięzcy mogą mogą też częściowo partycypować w zwycięzstwie neuronu c i adaptować swoje wagi. Dlatego niezmiernie ważnym pojęciem jest pojęcie **sąsiedztwa** zwycięskiego neuronu. Przy mapach sąsiedztwo to jest wyznaczane z wektorów referencyjnych na mapie.



(a) Hexagonal grid (b) Rectangular grid

Rysunek 8.5. Sąsiedztwo na mapach Kohonena: Neurony ułożone w siatkę (a) hexagonalną i (b) prostokątną.

8.1.8 Współczynnik uczenia

Współczynnik uczenia $\eta_i(k)$ maleje zazwyczaj wraz z upływem czasu uczenia wyznaczanego numerem iteracji k.

Niech T oznacza maksymalną liczbę iteracji. Liczbę tę ustala się z góry[2]. Dość często stosuje się następujące wzory na zmniejszanie współczynnika uczenia:

1. Liniowe zmniejszanie
   $\eta(t) = \eta_0 \, (T - t)/T, \quad t = 1, 2, \ldots, T$.
2. Wykładnicze zmniejszanie
   $\eta(t) = \eta_0 \, exp(-Ct), \quad t = 1, 2, \ldots, T, \quad C > 0$ jest pewną stałą.
3. Hiperboliczne zmniejszanie
   $\eta(t) = C_1/(C_2 + t), \quad t = 1, 2, \ldots, T, \quad C_1, C_2 > 0$ pewne stałe.
4. Indywidualny współczynnik uczenia, np.
   $\eta_i(t) = 1/n_i(t)$, gdzie $n_i(t)$ oznacza liczbę zwyciestw i-tego neuronu.

8.1.9 Dwie fazy uczenia

Na ogół uczenie przebiega w dwóch fazach. Najpierw przyjmuje się duże wartości η i duży promień sąsiedztwa.

W drugiej fazie (*fine tuning*) obydwie te wielkości ulegają istotnemu zmniejszeniu; w szczególności promień sąsiedztwa maleje do zera.

Pierwsza faza – przebiega według zasady WTM – promień sąsiedztwa jest duży, co powoduje, że oprócz neuronu-zwycięzcy również jego sąsiedzi zmieniają swoje wektory kodowe (przy mapach Kohonena są to sąsiedzi z mapy). Również współczynnik uczenia η jest w tej fazie stosunkowo duży.

Tak więc, przy każdej prezentacji kolejnego wektora $\mathbf{x}$ zostanie do niego przyciągnięty odpowiadający mu wektor-zwycięzca, który pociąga za sobą neurony z najbliższego sąsiedztwa.

[2]należy zwrócić uwagę, co oznacza faktycznie maksymalna liczba iteracji; często jest to liczebność próbki uczącej przemnożona przez liczbę epok

Zmiany wag następują według wzoru (2) lub (3).

Druga faza uczenia. Obowiązuje tu zasada WTA. Adaptacji podlegają tylko neuron–zwycięzca c, ponieważ promień sąsiedztwa zmalał do zera. Zmiany wag następują według wzoru (1).

8.1.10 Uczenie wsadowe

Dotychczas omawiane uczenie *sekwencyjne*, inaczej *na bieżąco*, lub *on–line* polegało na tym, że dla $t = 1, 2, \ldots$ prezentowaliśmy sieci wektory danych $\mathbf{x}(t)$, po czym następowało uaktualnienie wag zwycięskiego neuronu (i ewentualnie jego sąsiadów) wedlug zasad opisanych wzorami 1, 2 lub 3. Tak więc istotą było pojedyncze prezentowanie próbek (wektorów danych) i uaktualnianie wag po każdej prezentacji.

Wsadowe uczenie podobno (przynajmniej w przypadku SOM-ów) jest znacznie szybsze i bardziej stabilne; polega na wykonywaniu aktualizacji wag tylko na zakończenie każdej epoki (tj. gdy zostały zaprezentowane wszystkie próbki danych wynikające z ich randomizacji). Wariant uczenia wsadowego jest wariantem domyślnym w pakiecie somtoolbox2 przy trenowaniu sieci Kohonena. Algorytm uczenia wsadowego jest następujący (por. Skubalska [5], str. 187, za Kohonenem [3], również Vesanto [4], str. 9):

1. Ustal M początkowych wektorów kodowych. Początkowymi wektorami kodowymi mogą być wektory wygenerowane losowo, lub też M dowolnych wektorów danych z próbki uczącej.
2. Rozpocznij nową epokę i przedstawiaj sieci według w porządku zrandomizowanycm elementy próbki uczącej $\mathbf{x}_k = \mathbf{x}(k)$. Zaprezentuj sieci w ten sposób N wektorów danych.

   Zapamiętuj w czasie prezentacji dla każdego wektora $\mathbf{w}_i$ zbiór wektorów uczących $\{\mathbf{x}_k\}$ które oddziaływałyby na $\mathbf{w}_i$ w zwykłym algorytmie uczenia, oraz intensywność sąsiedztwa $h_{c(\mathbf{x}_k)i}$.
3. Na koniec epoki wyznacz nowe wartości wag (symbol $c(\mathbf{x}_k)$ oznacza neuron wygrywający przy prezentacji wektora danych $\mathbf{x}_k$, natomiast N jest ogólną liczebnością próbek uczących)

$$\mathbf{w}_i = \sum_{k=1}^{N} \mathbf{x}_k h_{c(\mathbf{x}_k)i} / \sum_{k=1}^{N} h_{c(\mathbf{x}_k)i}$$

4. Jeśli nie jest spełnione kryterium STOP-u (tutaj nie podaliśmy go), wróć do kroku 2.

8.2 Algorytm gazu neuronowego NG – Neural Gas

Algorytm ten został zaproponowany przez Martinetza i wspł. [1] w celu dokonania kwantyzacji przestrzeni cech i uzyskania pewnych wektorów prototypów które mogły by odgrywać rolę reprezentantów zbioru danych. Wektory– prototypy są znajdywane metodą uczenia bez nadzoru w warunkach konkurencji. Używa się łagodnej funkcji sąsiedztwa obejmującej wszystkie neurony i malejącej wykładniczo (według rankingu) w miarę zwiększania się odległości neuronu od prezentowanego wektora danych $\mathbf{x}(k)$.

W metodzie NG nie ma żadnej sieci, wytępuje raczej *kolekcja* swobodnych neuronów. Każdy neuron może poruszać sie swobodnie w przestrzeni, w której został określony. Swoboda poruszania jest jedynie ograniczona współzawodnictwem, inaczej mówiąc: konkurencją.

Ogólna idea tego algorytmu jest następująca: Neurony – w zadeklarowanej liczbie M – są sortowane w zależności od ich odległości od wektora $\mathbf{x}$, a następnie ulegają adaptacji w zależności od ich pozycji $m(i)$ w posortowanym ciągu odległości – wg funkcji wykładniczej $exp(-m(i)/\lambda)$ – por. wzór (4).

Tak jak w ogólnej teorii sieci neuronowych, neurony mają przypisane indywidualne *wektory wagowe* $\mathbf{w}_i$, $i = 1, \ldots, M$. Składowe wektora $\mathbf{w}_i$ są jednocześnie interpretowane jako współrzędne punktu $\mathbf{w}_i$ w przestrzeni cech R^d. Liczba rozważanych neuronów (u nas oznaczona jako M), jest zadawana z góry przez użytkownika.

Sąsiedztwo jest określane na zasadzie uporządkowania wszystkich wektorów $\mathbf{w}_i$ w zależności od ich odległości od wektora $\mathbf{x}$. Wektor najbliższy staje się wektorem – zwycięzcą, a inne wektory ustawiają się w kolejce za nim, w zależności od odległości $d^2(\mathbf{x}, \mathbf{w}_i)$. Pierwszy otrzymuje nazwę $\mathbf{w}_c$, jednocześnie neuron ten otrzymuje numer zerowy, czyli jest poza rankingiem. Pozostałe neurony otrzymują miejsca relatywnie do zwycięzcy, są to miejsca $1, 2, \ldots, M-1$.

Funkcja sąsiedztwa przyjmuje postać:

$$G(i) = exp\{-m(i)/\lambda\}, \tag{4}$$

gdzie $m(i)$ oznacza miejsce neuronu i w rankingu $1, \ldots, M-1$, z dodatkowym warunkiem, że $m(c)$ dla neuronu zwycięzcy równa się 0. Tak jest w funkcji neural_gas pakietu somtb2.

Tak więc mamy tu do czynienia z funkcją wyrażającą intensywność sąsiedztwa przypominającą 'membership function' z teorii zbiorów rozmytych.

Stosuje się uczenie sekwencyjne, przy czym wagi neuronów są uaktualniane według zasady *WTM* opisanej wzorem (2) z funkcją $G(i)$ opisaną wyżej. Występujący w tym wzorze parametr λ ma początkowo jakąś wartość startową λ_0, która maleje stopniowo tak żeby osiągnąć jakąś wartość końcową λ_{min} na zakończenie wszystkich iteracji (to znaczy wtedy, gdy $k = k_{max}$).

Można przyjąć następującą funkcję zmian parametru (taką funkcję przyjęto w procedurze neural_gas pakietu somtb2):

$$\lambda(k) \;=\; \lambda_0\Big(\frac{\lambda_{min}}{\lambda_0}\Big)^{k/k_{max}}. \tag{5}$$

Iteracje numeruje się $k = 0, 1, \ldots, k_{max}$.

Przy uaktualnianiu wag według wzoru (2) lub (3) potrzebny jest również współczynnik uczenia $\eta(k)$. Na ogół przyjmuje się, że współczynnik ten maleje ze wzrostem numeru iteracji, od jakiejś wartości początkowej η_0, zmniejszającej się stopniowo aż do wartości η_{min} osiąganej po ostatniej iteracji ostatniej epoki. Bardzo często przyjmuje się, że współczynnik η maleje według funkcji potęgowej

$$\eta(k) \;=\; \eta_0\Big(\frac{\eta_{min}}{\eta_0}\Big)^{k/k_{max}}. \tag{6}$$

Więcej o parametrze λ występującym we wzorze (4), cytowane za Skubalską [5], str. 207.:

Przy $\lambda \to 0$ mamy do czynienia z zerowym sąsiedztwem i regułą WTA. Przy $\lambda > 0$ są aktualizowane wagi wszystkich neuronów (czyli również tych, które są 'dalekimi' sąsiadami), lecz współczynnik uczenia odległych sąsiadów szybko dąży do zera. Jest to sytuacja podobna do tej, z jaką mamy do czynienia w klasycznym algorytmie SOM (self organizing map) z gaussowską funkcją sąsiedztwa. Tam również formalnie wagi wszystkich neuronów podlegają aktualizacji.

W przypadku asymptotycznym, gdy liczba neuronów dąży do nieskończoności, rozkład neuronów w przestrzeni wejść opisuje równanie różniczkowe o postaci takiej, jak równanie charakteryzujące dyfuzję gazu – stąd nazwa sieci 'gaz neuronowy'.

Metoda gazu neuronowego może być łączona z algorytmem uczenia konkurencyjnego typu Hebb'a, który nie ma wpływu bezpośrednio na wagi neuronów, służy jedynie do tworzenia na bieżąco struktury topologicznej sieci [9].

Metoda NG daje bardzo dobre rezultaty szczególnie, gdy przestrzeń cech składa się z rozłącznych podzbiorów lub ma skomplikowany kształt.

Algorytm gazu neuronowego nie jest tak popularny jak np. metoda samoorganizujących sie sieci–map Kohonena. Jednak badania na pewnych danych uważanych za zbiory typu 'benchmark' wykazały, że metoda gazu neuronowego (NG) jest bardziej efektywna niż metoda sieci Kohonena i parę innych metod [2]. Z instytutowych doświadczeń wynika, że metoda ta wymaga trochę więcej czasu, jednak uzyskane reprezentacje (wektory–prototypy) dają mniejszy błąd reprezentacji. Błąd ten był wyznaczany jako średnia odległość każdego punktu próbki danych uczących (testowych) do najbliższego znalezionego reprezentanta.

Metodę NG badano również w aspekcie redukcji wymiarowości zmiennych (input variables [7]) próbując dodawać stopniowo zmienne aż uzyskałoby się ich dostateczną ilość ("growing neural gas" until a satisfactory representation is achieved) [8]. Metoda ta znalazła również interesujące zastosowania w robotyce (interesting applications concerned with visuo-motor control of an industrial robot) [6].

Algorytm gazu neuronowego jest podobno bardziej skuteczny niż algorytm Kohonena, będący jego szczególnym przypadkiem.

Implementacja w pakiecie somtb2

[Neurons] = neural_gas(X, M, epochs, eta0, lambda0);
X – tablica danych,
M – ile neuronów,
epochs – liczba epok; w każdej epoce jest prezentowany sekwencyjnie (ale w porządku zrandomizowanym) cały zbiór danych, czyli wszystkie wiersze tablicy X.
eta0 – współczynnik uczenia, można podstawić np. eta0=0.5; ulega sukcesywnemu zmniejszaniu w trakcie kolejnych iteracji; w oryginalnej procedurze wspólczynnik ten jest oznaczony symbolem alpha,
lambda0 – promień sąsiedztwa, wartość domyślna: = M/2; ulega sukcesywnemu zmniejszaniu w trakcie kolejnych iteracji.
Liczba iteracji wynosi: $epochs \times M$.
Parametry lambda0 i eta0 są parametrami opcyjnymi. Ich wartości domyślne to: *lambda0=M/2*, *eta0=0.5*.

Wartości λ i *eta* zmniejszają się wraz z numerem iteracji zgodnie z wypisanymi wyżej wzorami (5) i (6) – od początkowych wartości λ_0 i η_0 do końcowych wartości η_{min} and λ_{min} (w procedurze przyjęto: $\eta_{min} = 0.005,\ \lambda_{min} = 0.01$).

Do wizualizacji otrzymanych prototypów polecamy metodę Sammona opisan"a w rozdzale 10 notatek.

8.3 Samo-organizujace się mapy Kohonena

Typowy przedstawiciel takich sieci: Mapa Kohonena na płaszczyźnie[3] nazywane również *SOM*-ami od 'Self Organizing Maps'. SOM-y realizują generalnie dwa zadania:

1. Wektorowej kwantyzacji (kompresji danych),
2. Odtwarzanie przestrzennej organizacji danych wejściowych

Istotną nowością jest pojawienie się tutaj 'mapy' która pokazuje topologiczne sąsiedztwo punktów znajdujących się w wielowymiarowej przestrzeni R^d.

Samoorganizacji podlega cały zbiór danych: nie rozróżnia się próbki uczącej i testowej.

Mapy Kohonena zosta"ly om"owione w oddzielnym rozdziale notatek.

8.4 Inne algorytmy uczenia sieci samoorganizujących

Miary odległości między wektorami, problem normalizacji wektorów (należy dodać tzw. normalizację na 0–1, o czym Osowski nie mówi)

- Algorytmy uwzględniające "zmęczenie" neuronów po zwycięstwie.
- Algorytm stochastycznej relaksacji (neurony ulegają adaptacji z pb-stwem określonym wzorem Gibbsa).
- SCS, Soft Competition (uwzględnia się stopień aktywności neuronów, faworyzując jednostki mało aktywne).
  Wspolczynnik uczenia może być indywidualny dla każdego neuronu, jest uzależniony od tego, co robił neuron w ostatnich $k-1$ prezentacjach

Dalsze przykłady pokazujące samo-organizowanie się sieci na płaszczyźnie można znaleźć w rozdziale 8 książki Osowskiego [2].

Literatura

[1] Martinetz M., Berkovich S., Schulten K.: 'Neural-gas' network for vector quantization and its application to time series prediction. IEEE Trans. Neural Networks, V. 4, 1993, 558–569.

[2] S. Osowski, *Sieci neuronowe w ujęciu algorytmicznym.* WNT W-wa 1996.

[3] T. Kohonen, *Self-organising Maps*. Springer, Berlin - Heidelberg, 1995.

[4] Vesanto J., Himberg J., Alhoniemi E., Parhankangas J., *SOM Toolbox for Matlab 5.* Som Toolbox team, Helsinki University of Technology, Finland, Libella Oy, Espoo 2000, 1–54. See also:
http://www.cis.hut.fi/projects/somtoolbox/ Version 0beta 2.0, November 2001.

[3] opisana np. w książce Osowskiego, str 249–275

[5] Ewa Skubalska-Rafajłowicz, Samoorganizujące sieci neuronowe. W: M. Nałęcz (red), *Biocybernetyka i Inżynieria Biomedyczna 2000. Tom 6: Sieci neuronowe,* str. 187–188

[6] Walter J.A., Schulten K.J., Implementation of Self-organizing Neural Networks for Visuo-motor Control of an Industrial robot. *IEEE Trans. on Neural Network*, 1993.

[7] Hammer B., Strickert M., Villman Th., Supervised neural gas for learning vector quantization. In: D. Polani, J. Kim, T. Martinez (eds.), Fifth German Workshop on Artificial Life, IOS Press, 9–18, 2002.

[8] Heinke D., Hamker F.H., Comparing neural network benchmarks on growing neural gas, growing cell structure, and fuzzy ARTMAP. *IEEE Trans. on Neural Network*, 1998, pp. 1279–1291. Also by the same authors: Report Nr. 1/97, Technische Universität Ilmenau, Fakultät für Informatik and Automatisierung. D–98684 Ilmenau.

[9] Martinez T.M., Schulten K.J. (1994). Topology representing networks. Neural Networks **7**, 507–522.