Metody sztucznej inteligencji: (1) samoorganizująca się mapa Kohonena (SOM), (2) sieć neuronowa o bazach radialnych (RBF).

Przemysław Klęsk
Katedra Metod Sztucznej Inteligencji i Matematyki Stosowanej
Wydział Informatyki, ZUT Szczecin

1 Klasteryzacja za pomocą samoorganizującej się mapy Kohonena

1.1 Szkic idei działania

Dany jest zbiór punktów (przykładów uczących): $D = \left\{\mathbf{x}_i\right\}_{i=1,\ldots,m}$, gdzie $\mathbf{x}_i = (x_{i1}, \ldots, x_{in}) \in \mathbb{R}^n$.

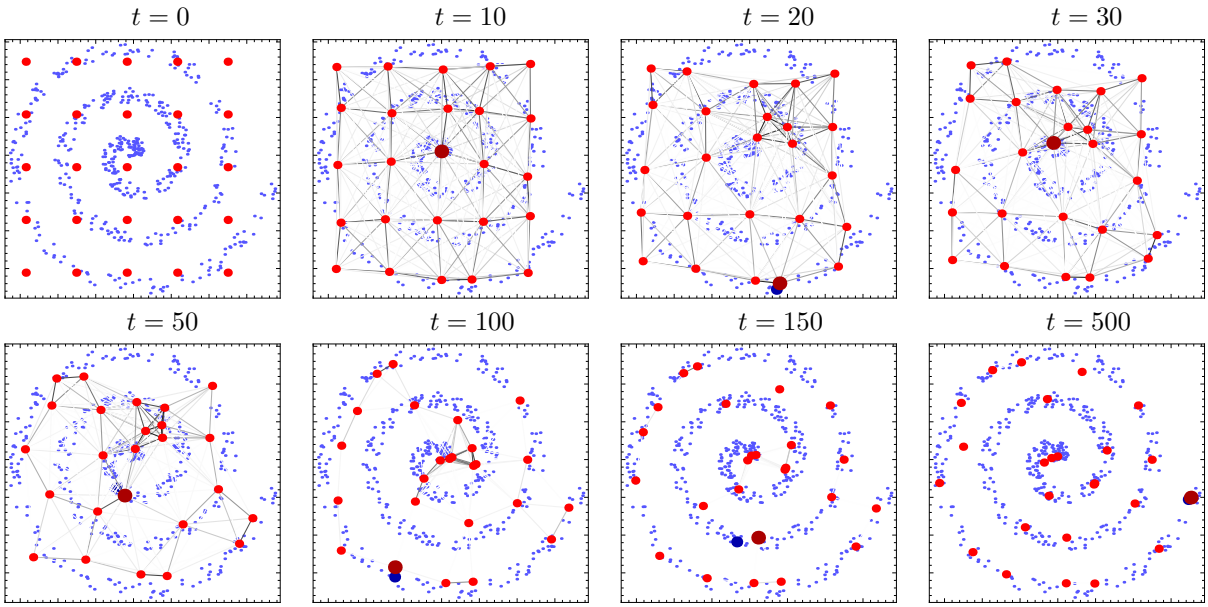
Zadanie *klasteryzacji* polega na przybliżeniu danego zbioru punktów za pomocą mniejszej liczby K punktów (zawykle $K << m$). W ten sposób można mówić o wykrywaniu skupień w zbiorze danych o pewnym rozkładzie czy też o wykrywaniu kształtów. Punkty, za pomocą których przybliżamy, nazywa się zwykle *prototypami* lub *centrami klastrów*. Zadanie klasteryzacji jest z definicji zadaniem uczenia się bez nadzoru (ang. *unsupervised learning*), co oznacza, że w zbiorze uczącym nie została wyróżniona zmienna y — zmienna decyzyjna lub objaśniana — jak ma to miejsce w zadaniach klasyfikacji lub aproksykmacji.

Istnieje wiele algorytmów klasteryzacji m.in.: *algorytm K-średnich*, *algorytm Warda*, *algorytm rozmytych C-środków*. Pierwsza część niniejszego opracowania jest poświęcona algorytmowi znanemu jako: *samoorganizująca się mapa Kohonena*[1] (ang. *SOM — Self Organizing Map*). Ogólnej intuicji na temat działania tego algorytmu dostarcza Rys. 1. Ilustruje on przebieg algorytmu SOM dla dwuwymiarowego zbioru danych (ułożonego w formie pasa spiralnego), przedstawiając stan algorytmu w kilku wybranych krokach (iteracjach). Na wykresach kolorem niebieskim zaznaczono punkty danych a kolorem czerwonym prototypy. Pogrubiony punkt niebieski widoczny na niektórych wykresach reprezentuje przykład uczący wylosowany w bieżącym kroku, zaś pogrubiony punkt czerwony reprezentuje najbliższy mu prototyp — tzw. *zwycięzcę*. Jak zostanie wyjaśnione dalej, to właśnie prototyp-zwycięzca będzie podlegał największej poprawce położenia. Dodatkowo szare odcinki reprezentują połączenia pomiędzy prototypami. Można rozumieć je jako powiązania powodujące, że zmiana położenia pewnego prototypu pociąga za sobą także zmiany (słabsze) prototypów sąsiednich. Im jaśniejszy odcień szarości tym słabsze powiązanie. W późniejszych krokach algorytmu wszystkie powiązania stopniowo wygasają.

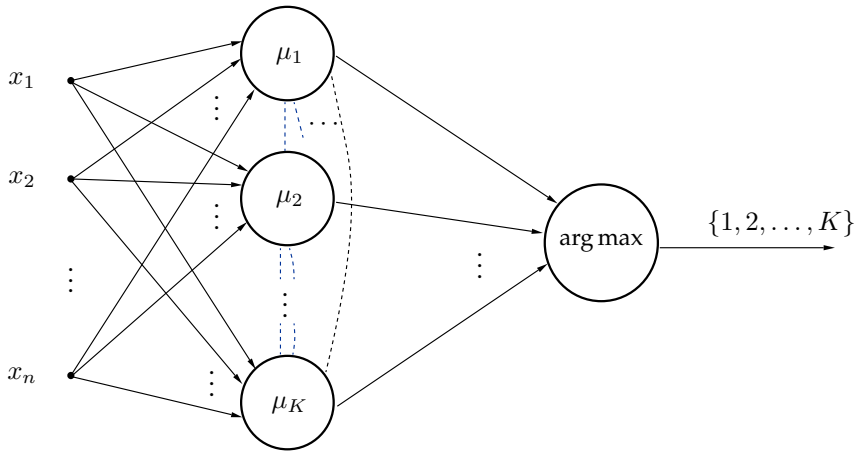
Warto dodać, że oprócz powyżej nakreślonego sensu geometrycznego, mapa Kohonena może być przedstawiana i rozumiana jako pewnego rodzaju sieć neuronowa. W takiej sieci neurony odpowiadają 1:1 prototypom, a ich wagi współrzędnym prototypów. Dodatkowo, oprócz połączeń od sygnałów wejściowych $x_1, \ldots, x_n$ do neuronów, sieć zostaje wyposażona w strukturę połączeń pomiędzy samymi neuronami. Te właśnie połączenia odpowiadają wspomnianym powiązaniom geometrycznym i decydują w trakcie uczenia o sile zmiany wag neuronów sąsiedzkich do neuronu-zwycięzcy. Sygnałem wyjściowym sieci neuronowej Kohonena jest zwykle indeks (numer) zwycięzkiego neuronu. A zatem po zakończeniu uczenia nowoprzychodzące przykłady (wektory) podawane na wejście sieci zostają zamapowane na skończony zbiór indeksów $\{1, 2, \ldots, K\}$. Stąd też, możemy mówić, że nauczona sieć realizuje zadanie kwantowania wektorowego lub kompresji stratnej[2]. Schemat tego typu sieci przedstawiono na Rys. 2.

[1] Lub: *samorganizująca się sieć Kohonena*.

[2] Są to jeszcze inne równoważne nazwy klasteryzacji.



Rysunek 1: Ilustracja pracy algorytmu SOM w wybranych krokach — zbiór punktów „spirala” (kolor niebieski) jest przybliżany za pomocą $K = 25$ prototypów.



Rysunek 2: Schemat mapy Kohonena jako sieci neuronowej. Krzywe przerywane reprezentują strukturę powiązań pomiędzy samymi neuronami.

1.2 Uwaga ogólna o metodach odległościowych

Praktycznie każdy algorytm klasteryzacji jest metodą odległościową, tzn. wykorzystuje w trakcie pracy pewną metrykę do obliczenia odległości pomiędzy punktami. Jeżeli rzeczywiste dane wejściowe nie zostaną wstępnie przetworzone, to zmienne o mniejszej rozpiętości lub wariancji mogą być nieprawidłowo faworyzowane podczas obliczania odległości i wynik algorytmu może być daleki od oczekiwanego. W praktyce zaleca się zatem ujednolicenie wpływu wszystkich zmiennych wejściowych na przykład poprzez przeskalowanie zbioru danych do pewnego hipersześcianu lub też poprzez ustandaryzowanie wszystkich wariancji[3].

[3]Inną ewentualnością jest też używanie metryki Mahalanobisa, neutralizującej za pomocą macierzy kowariancji wpływy poszczególnych zmiennych wejściowych. Ta możliwość nie jest jednak zwykle używana w przypadku sieci Kohonena realizowanej z wykorzystaniem iloczynu skalarnego.

1.3 Obliczenia i uczenie w sieci Kohonena

Uczenie należy rozpocząć od pewnego rozmieszczenia prototypów w dziedzinie. Spotyka się w tym zakresie trzy podejścia:

1. rozmieszczenie losowe jednostajne,
2. rozmieszczenie losowe spośród punktów danych (prototypy pokrywają się z wylosowanymi punktami danych),
3. rozmieszczenie regularne (grid).

Przykłady przedstawione w niniejszym opracowaniu ograniczają się do podejścia nr 3. Od tej pory niech wektory $\mathbf{c}_k = (c_{k,1}, c_{k,2}, \ldots, c_{k,n})$ oznaczają współrzędne określające położenie poszczególnych prototypów, $k = 1, 2, \ldots, K$.

Pomyślmy przez chwilę o reprezentacji SOM jako sieci neuronowej takiej jak na Rys. 2. Wektory $\mathbf{c}_k$ potraktujmy zatem jako współczynniki wagowe. Wspólnym elementem w wielu rodzajach sieci neuronowych jest obliczanie ważonej sumy wejść czyli iloczynu skalarnego:

$$\langle \mathbf{c}_k, \mathbf{x} \rangle = c_{k,1}x_1 + c_{k,2}x_2 \cdots + c_{k,n}x_n. \tag{1}$$

Jak wiadomo iloczyn skalarny może być rozumiany jako stopień podobieństwa dwóch wektorów o ustalonych długościach:

$$\langle \mathbf{c}_k, \mathbf{x} \rangle = \|\mathbf{c}_k\| \|\mathbf{x}\| \cos \angle(\mathbf{c}_k, \mathbf{x}). \tag{2}$$

Jak wspomniano wcześniej, pierwszym zadaniem sieci SOM jest wyłonienie neuronou będącego tzw. *zwycięzcą* (ang. *winner*). Chcielibyśmy posłużyć się do tego celu właśnie iloczynem skalarnym. Niestety okazuje się, że sam iloczyn skalarny nie wystarczy. Mówiąc ściślej maksimum iloczynu skalarnego $\langle \mathbf{c}_k, \mathbf{x} \rangle$ nie leży w ogólności tym samym miejscu co minimum odległości $\|\mathbf{c}_k - \mathbf{x}\|$. Potrzebna jest pewna korekta, którą przedstawia poniższe wyprowadzenie:

$$\begin{aligned}
\arg\min_{k=1,\ldots,K} \|\mathbf{c}_k - \mathbf{x}\| &= \arg\min_{k=1,\ldots,K} \|\mathbf{c}_k - \mathbf{x}\|^2 = \arg\min_{k=1,\ldots,K} \langle \mathbf{c}_k - \mathbf{x}, \mathbf{c}_k - \mathbf{x} \rangle \\
&= \arg\min_{k=1,\ldots,K} \Big(\langle \mathbf{c}_k, \mathbf{c}_k \rangle - 2\langle \mathbf{c}_k, \mathbf{x} \rangle + \langle \mathbf{x}, \mathbf{x} \rangle \Big) = \arg\min_{k=1,\ldots,K} \Big(\|\mathbf{c}_k\|^2 - 2\langle \mathbf{c}_k, \mathbf{x} \rangle + \|\mathbf{x}\|^2 \Big) \\
&= \arg\max_{k=1,\ldots,K} \Big(2\langle \mathbf{c}_k, \mathbf{x} \rangle - \|\mathbf{c}_k\|^2 - \|\mathbf{x}\|^2 \Big) = \arg\max_{k=1,\ldots,K} \Big(2\langle \mathbf{c}_k, \mathbf{x} \rangle - \|\mathbf{c}_k\|^2 \Big).
\end{aligned} \tag{3}$$

Ostatnie przejście wynika z faktu, że norma (długość) podanego wektora wejściowego $\mathbf{x}$ jest stała dla wszystkich neuronów i może być pominięta przy wyborze argumentu maksimum.

A zatem wyłonienie zwycięzcy może wykorzystywać w pewien sposób iloczyn skalarny, przy czym należy go podwoić i odjąć kwadrat normy wektora wag. Ostatecznie można przyjąć więc, że funkcja aktywacji neuronu μ_k pracuje wg wzoru:

$$\mu_k(\mathbf{x}) = 2\langle \mathbf{c}_k, \mathbf{x} \rangle - \|\mathbf{c}_k\|^2. \tag{4}$$

Dla wygody dalszej notacji oznaczmy indeks zwycięzcy jako $w(\mathbf{x})$, tzn.:

$$w(\mathbf{x}) = \arg\max_{k=1,\ldots,K} \mu_k(\mathbf{x}). \tag{5}$$

Wyłoniwszy zwycięzcę należy zmodyfikować jego wagi (jego położenie w dziedzinie). Można by to zrobić na przykład wg wzoru:

$$\mathbf{c}_{w(\mathbf{x})} := \mathbf{c}_{w(\mathbf{x})} + \eta(\mathbf{x} - \mathbf{c}_{w(\mathbf{x})}), \tag{6}$$

gdzie stała $\eta \in (0, 1]$ oznacza wielkość zmiany i może być interpretowana jako współczynnik uczenia. Innymi słowy przesuwamy wektor $\mathbf{c}_{w(\mathbf{x})}$ o pewną długość wzdłuż wektora $\mathbf{x} - \mathbf{c}_{w(\mathbf{x})}$. W szczególności przy ustawieniu $\eta = 1$, neuron-zwycięzca w wyniku poprawki wskoczy dokładnie w miejsce przykładu uczącego $\mathbf{x}$.

Jeżeli modyfikacji w danej iteracji podlegałby tylko sam zwycięzca, to mamy do czynienia z tzw. wariantem WTA (ang. *Winner Takes All*) algorytmu uczenia. W praktyce używa się zwykle jednak wariantu

WTM (ang. *Winner Takes Most*). Oznacza to, że nie tylko sam zwycięzca, ale i jego neurony-sąsiedzi (z punktu widzenia struktury powiązań międzyneuronowych) podlegają modyfikacjom, tyle że w słabszym stopniu. Można użyć do tego celu następującego wzoru:

$$\mathbf{c}_k := \mathbf{c}_k + \eta(\mathbf{x} - \mathbf{c}_k)\alpha G_\sigma\big(k, w(\mathbf{x})\big), \qquad k = 1, \ldots, K, \tag{7}$$

gdzie $\alpha = \|\mathbf{x} - \mathbf{c}_{w(\mathbf{x})}\|/\|\mathbf{x} - \mathbf{c}_k\|$, zaś funkcja G jest gaussowską funkcją bliskości pomiędzy neuronami:

$$\begin{aligned} G_\sigma\big(k, w(\mathbf{x})\big) &= \exp\left(-\frac{(c_{k1} - c_{w(\mathbf{x}),1})^2 + \cdots + (c_{kn} - c_{w(\mathbf{x}),n})^2}{2\sigma^2}\right) \\ &= \exp\left(-\frac{\|\mathbf{c}_k - \mathbf{c}_{w(\mathbf{x})}\|^2}{2\sigma^2}\right). \end{aligned} \tag{8}$$

Parametr σ steruje szerokością dzwonu gaussowskiego. Im większe σ, tym większy zasięg sąsiedztwa. Zasięg ten powinien być wygaszany w trakcie pracy algorytmu. Popularnym podejściem jest wygaszanie wykładnicze, poczynając od pewnej zadanej wartości maksymalnej $\sigma_{\max}$ aż do zadanej wartości minimalnej $\sigma_{\min}$. I tak niech $t = 1, 2, \ldots, T$ oznacza licznik głównej pętli uczącej. Wówczas wygaszanie parametru σ przebiega wg wzoru:

$$\sigma_t = \sigma_{\max}\left(\frac{\sigma_{\min}}{\sigma_{\max}}\right)^{\frac{t-1}{T-1}}. \tag{9}$$

Wartość σ_t jest używana do poprawek w iteracji o numerze t.

Jeżeli chodzi o sam dobór przedziału $[\sigma_{\min}, \sigma_{\max}]$, to niestety należy tu trochę poeksperymentować, uwzględniając przede wszystkim rozpiętość dziedziny (np. po odrzuceniu odstających punktów danych[4]). Rzecz w tym, że zbyt duża wartość początkowa $\sigma_{\max}$ może prowadzić do niechcianego efektu zbiegnięcia się wszystkich prototypów w jeden punkt już po kilku iteracjach. Z kolei zbyt szybkie osiągnięcie szerokości bliskich $\sigma_{\min}$ spowoduje, że algorytm będzie pracował przez większość czasu de facto wg wariantu WTA zamiast bardziej porządanego WTM.

W przykładach towarzyszących temu opracowaniu $\sigma_{\max}$ dobrano jako 0.25 rozpiętości początkowego grida neuronów po odrzuceniu 5% punktów odstających po każdej ze stron, patrz też Rys. 1. Z kolei $\sigma_{\min}$ ustawiono na wartość $0.01\sigma_{\max}$.

Warto na marginesie wspomnieć, że przedstawiona powyżej funkcja sąsiedztwa między neuronami nie jest jedyną możliwością. Funkcja gaussowska jest przykładem sąsiedztwa geometrycznego (euklidesowego). Inną spotykaną możliwością jest wykorzystanie sąsiedztwa strukturalnego, np. sąsiedztwa w ramach grida prostokątnego. Rozważmy dla przykładu przypadek dwuwymiarowy. Jeżeli neurony w początkowym ustawieniu gridowym zostałyby ponumerowane za pomocą par indeksów (i, j) na zasadzie: $\mathbf{c}_{1,1}$ — lewy górny neuron, $\mathbf{c}_{1,2}$ — jego prawy sąsiad, itd., to funkcja sąsiedztwa strukturalnego mogłaby być np. określona z wykorzystaniem metryki Manhattan w następujący sposób:

$$G_\sigma\big((i_1, j_1), (i_2, j_2)\big) = \exp\left(-\frac{|i_1 - i_2| + |j_1 - j_2|}{2\sigma^2}\right). \tag{10}$$

Innymi słowy sąsiedztwo strukturalne ignoruje faktyczną odległość geometryczną pomiędzy aktualnym położeniem neuronów, patrzy zaś na ich odległość w sensie różnicy numeracyjnej.

Podsumowując treści przedstawione w tym punkcie, ostateczny algorytm uczenia on-line sieci Kohonena można sformułować w postaci poniższego pseudokodu.

Algorytm TRAINSOM($D; K, T, \eta$)
  Ustal początkowe rozstawienie neuronów: $\mathbf{c}_k$, $k = 1, \ldots, K$, np. w formie grida.
  Wykorzystując informacje o wariancji zbioru danych ustal wartości: $\sigma_{\min}$, $\sigma_{\max}$.
  Dla $t := 1, \ldots, T$ **powtarzaj**
    Wylosuj przykład uczący $\mathbf{x}_i$ ze zbioru D.
    Oblicz wartości wyjściowe wszystkich neuronów w sieci SOM:
      $\mu_k(\mathbf{x}_i) := 2\langle \mathbf{c}_k, \mathbf{x}_i \rangle - \|\mathbf{c}_k\|^2, \qquad k = 1, \ldots, K.$

[4] Tzw. *outlierów*.

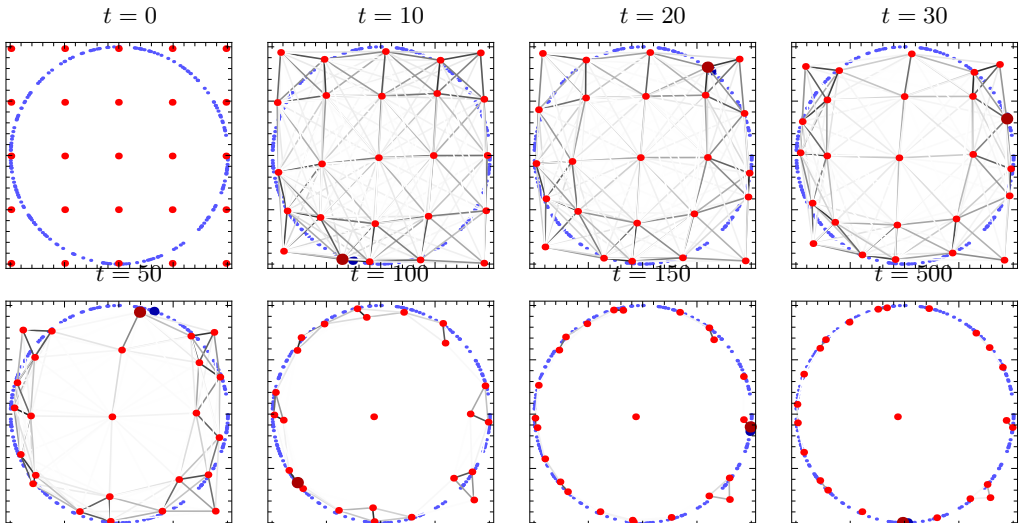
Wołoń neuron-zwycięzcę:
$w(\mathbf{x}_i) = \arg\max_{k=1,\ldots,K} \mu_k(\mathbf{x}_i)$.
Oblicz wartość parametru sąsiedztwa σ_t obowiązującą dla aktualnej iteracji:
$\sigma_t = \sigma_{\max} \left(\frac{\sigma_{\min}}{\sigma_{\max}}\right)^{\frac{t-1}{T-1}}$.
Oblicz poprawki dla wszystkich neuronów:
$\Delta_k := \eta(\mathbf{x}_i - \mathbf{c}_k)\alpha G_{\sigma_t}\big(k, w(\mathbf{x}_i)\big), \qquad k = 1, \ldots, K.$
Aktualizuj położenie wszystkich neuronów:
$c_k := c_k + \Delta_k, \qquad k = 1, \ldots, K.$
Zwróć końcowe położenie neuronów $\mathbf{c}_k$, $k = 1, \ldots, K$.

1.4 Przykłady do ćwiczeń domowych

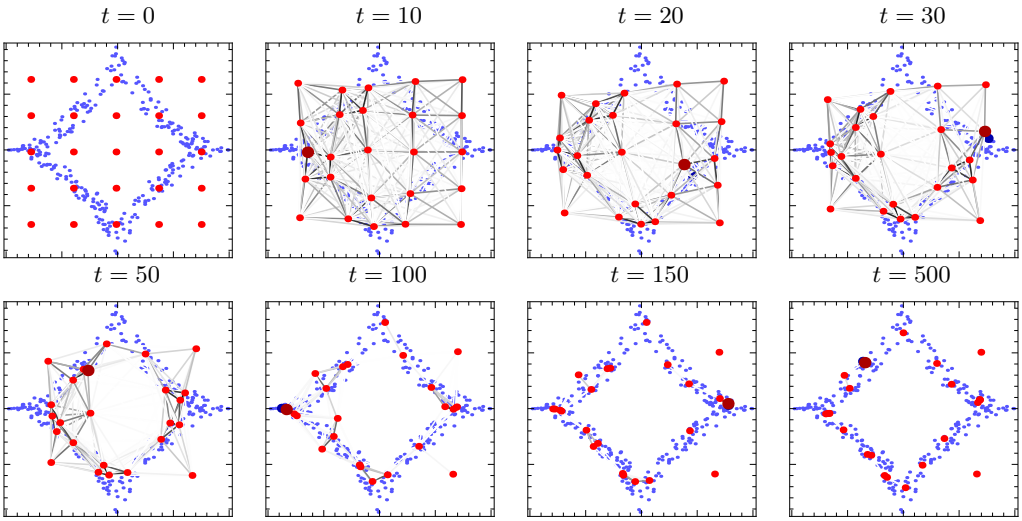
Należy wykonać eksperymenty dla czterech zbiorów danych na płaszczyźnie wygenerowanych z pewną losowością wg poniższych kształtów geometrycznych (wskazówki na zajęciach):

1. okrąg,
2. symbol karo (pas punktów),
3. spirala (pas punktów),
4. okrąg plus koło.

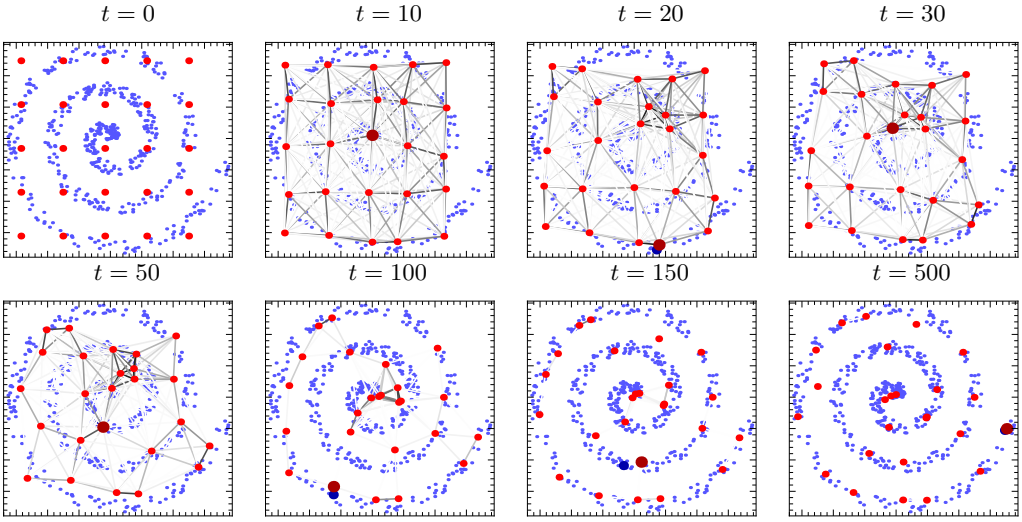
Wizualizacje działania algorytmu przedstawiono na rysunkach 3–6. Wszystkie uruchomienia przeprowadzono z ustawieniami: liczba neuronów $K = 25$ (układ początkowy jako grid 5×5), liczba iteracji uczących $T = 500$, współczynnik uczenia $\eta = 0.5$, maksykmalna szerokość sąsiedztwa $\sigma_{\max} = 0.25 \cdot$ początkowy rozstęp grida, minimalna szerokość sąsiedztwa $\sigma_{\min} = 0.01 \cdot \sigma_{\max}$.



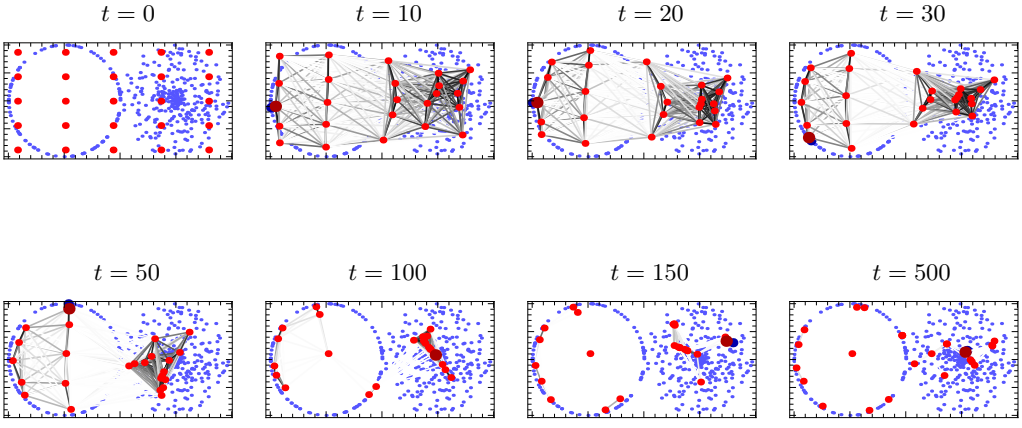
Rysunek 3: Zbiór punktów „okrąg”, liczba przykładów w zbiorze $m = 200$.



Rysunek 4: Zbiór punktów „karo”, liczba przykładów w zbiorze $m = 300$.



Rysunek 5: Zbiór punktów „spirala”, liczba przykładów w zbiorze $m = 500$.



Rysunek 6: Zbiór punktów „okrąg plus koło”, liczba przykładów w zbiorze $m = 500$.

1.5 Problemy obserwowane w działaniu algorytmu SOM

Można zaobserwować występowanie dwóch niepożądanych zjawisk w wynikowej klasteryzacji dostarczanej przez algorytm SOM:

1. prototypy odległe od punktów danych („osamotnione"),
2. prototypy pokrywające się.

Pierwsze zjawisko dotyczy sytuacji, kiedy to początkowe położenie pewnego prototypu (lub kilku prototypów) względem przybliżanego kształtu / rozkładu punktów uczących jest na tyle odległe, że prototyp ten ma małe szanse na bycie wyłonionym jako zwycięzca w trakcie uczenia. Co prawda czasami tego typu prototypy mogą być częściowo przyciągnięte do skupisk danych pośrednio jako sąsiadzi innych przesuwanych prototypów, jednakże jeżeli siła sąsiedztwa gaśnie dość szybko, to nadal finalne przesunięcie może być niewystarczające. W konsekwencji w końcowym stanie zwracanym przez algorytm można zaobserwować istnienie „osamotnionych" prototypów, rozlokowanych dalego od skupisk danych. Do rozwiązania tego problemu bywają stosowane różne techniki licznikowe — zliczanie zwycięstw poszczególnych prototypów. Powodują one, że zwycięzcy w dalszych iteracjach są wybierani nie tylko na podstawie bliskości do przykładu uczącego, ale też na podstawie swojego licznika poprzednich zwycięztw. Im mniejsza wartość licznika, tym większe prawdopodobieństwo bycia wybranym.

Drugie zjawisko pojawia się w sytuacji, kiedy zakres sąsiedztwa w początkowej fazie algorytmu jest zbyt duży i niektóre prototypy zbiegają się niemalże w ten sam punkt. Wynika to to najczęściej ze złego doboru przedziału $[\sigma_{\min}, \sigma_{\max}]$, ale może być także powodowane pewną szczególną naturą rozkładu będącego przedmiotem przybliżania. Takie zgrupowane prototypy zwykle „podróżują" odtąd razem w trakcie dalszej pracy algorytmu, nie mogąc oderwać się od siebie. Najprostszym zalecanym krokiem jest wówczas post-processing przeprowadzany po wykonaniu algorytmu, polegający na usunięciu prototypów duplikujących się z innymi (tj. pozostawieniu po jednym z każdej takiej grupy).

2 Aproksymacja za pomocą sieci RBF

2.1 Ogólna idea

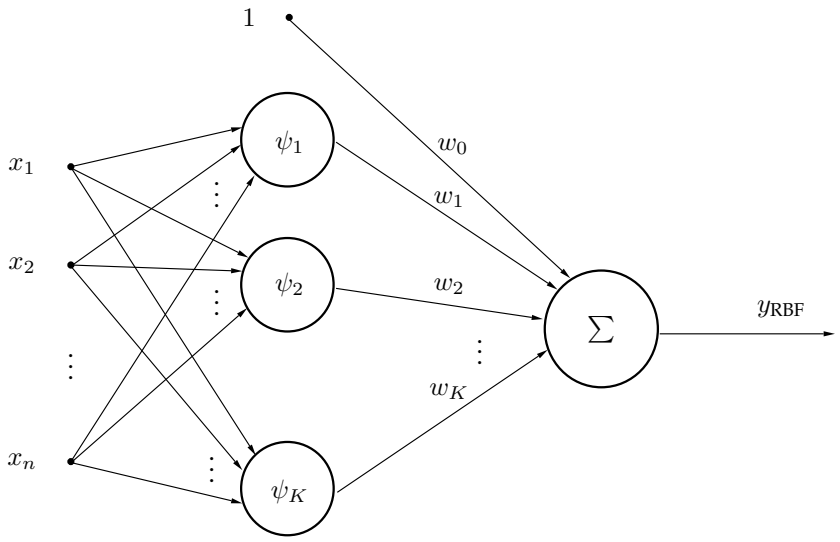
W tym punkcie rozpatrujemy zadanie aproksymacji (podobnie jak to miało miejsce w zadaniu dotyczącym sieci neuronowej MLP) czyli zadanie uczenia z nadzorem. Oznacza to, że w danych istnieje pewna zmienna wyróżniona jako y, w przypadku zadania aproksymacji jest to zmienna rzeczywista, której wartości chcielibyśmy umieć przewidywać za pomocą pozostałych zmiennych. Uściślając, dany jest zbiór par uczących: $D = \left\{(\mathbf{x}_i, y_i)\right\}_{i=1,\ldots,m}$, gdzie $\mathbf{x}_i = (x_{i1}, \ldots, x_{in}) \in \mathbb{R}^n$, $y_i \in \mathbb{R}$.

Sieć neuronowa RBF (ang. *Radial Basis Function*) to aproksymator o tzw. bazach radialnych lub inaczej kołowych. Oznacza, to że funkcja aktywacji neuronów jest funkcją wykazującą symetrię kołową i ma ona lokalny zakres działania. Dla odróżnienia warto przypomnieć, że sieci typu MLP miały neurony o sigmoidalnej funkcji aktywacji, której pobudzenie (wartości bliskie 1) może rozciągać się na dowolnie duże fragmenty dziedziny. W sieci RBF neuron jest pobudzony tylko w otoczeniu pewnego punktu w dziedzinie (którego położenie wynika z parametrów neuronu) i wraz z oddalaniem się od tego punktu pobudzenie neuronu wygasa (zbiega do wartości 0).

Podobnie jak wiele innych sieci neuronowych, sieć RBF można uczyć algorytmami gradientowymi sposobem on-line (poprawki parametrów następują po każdym pokazanym przykładzie). Okazuje się jednak, że istnieje wygodny w praktyce algorytm uczenia off-line dla sieci RBF wykorzystujący jako krok pośredni *klasteryzację*. Taka możliwość wynika z lokalności działania neuronów RBF. Stąd też właśnie samorganizująca się mapa Kohonena, jako przykład algorytmu klasteryzacji, jest przydatnym wstępem do omawiania uczenia sieci RBF.

2.2 Obliczenia i uczenie w sieci RBF

Schemat sieci RBF przedstawiono na Rys. 7. W sieci występuje jedna warstwa neuronów nieliniowych oraz druga warstwa z jednym neuronem wykonującym sumę ważoną wyjść neuronów z warstwy poprzedniej.



Rysunek 7: Schemat sieci RBF.

Odpowiedź sieci RBF można zapisać wzorem:

$$y_{\text{RBF}}(\mathbf{x}) = w_0 + \sum_{k=1}^{K} w_k \psi_k(\mathbf{x}), \tag{11}$$

gdzie jako funkcję radialną przyjmuje się zwykle funkcję Gaussa:

$$\psi_k(\mathbf{x}) = \exp\left(-\frac{(x_1 - c_{k,1})^2 + \cdots + (x_n - c_{k,n})^2}{2\sigma_k^2}\right). \tag{12}$$

W sieci RBF parametry $\mathbf{c}_k = (c_{k,1}, c_{k,2}, \ldots, c_{k,n})$ nazywane są zwyczajowo *centrami* neuronów (punkty centralne dzwonów gaussowskich), zaś parametry σ_k stanowią *szerokości* neuronów (szerokość dzwonu na wysokości $e^{-1/2}$). Myśląc geometrycznie, zadaniem sieci RBF jest przybliżenie pewnej spróbkowanej funkcji za pomocą liniowej kombinacji kształtów dzwonowych. Dla odróżnienia, sieć MLP przybliżała za pomocą liniowej kombinacji sigmoid.

Jak wspomniano w praktyce stosuje się off-line'owy algorytm uczenia sieci RBF oparty o klasteryzację. Warto zaznaczyć, że algorytm ten należy zaliczyć do grupy algorytmów heurystycznych, jako że po pierwsze zawiera on pewne elementy dobieralne przez użytkownika, po drugie nie daje gwarancji znalezienia rozwiązania optymalnego[5]. Mimo to, algorytm ten prowadzi do szybszego uczenia niż algorytmy gradientowe on-line i ma naturalną geometryczną motywację przemawiającą za jego użyciem.

Szkic postępowania można przedstawić następująco.

1. Zadaj liczbę K — liczbę neuronów radialnych.
2. Wykonaj klasteryzację (np. za pomocą algorytmu SOM Kohonena) zbioru: $\{\mathbf{x}_i\}_{i=1,\ldots,m}$ (ignorując zmienną y) — uzyskując w ten sposób K prototypów jako centra dla przyszłych neuronów radialnych: $\mathbf{c}_1, \ldots, \mathbf{c}_K$.
3. Dobierz szerokości $\sigma_1, \ldots, \sigma_K$ neuronów, biorąc pod uwagę ich odległości do najbliższych neuronów-sąsiadów.
4. Mając już ustalone parametry: $K, \mathbf{c}_1, \ldots, \mathbf{c}_K, \sigma_1, \ldots, \sigma_K$, znajdź wagi $w_0, \ldots, w_K$ na podstawie kryterium najmniejszych kwadratówww (czyli rozwiązując liniowy problem optymalizacji).

Elementy heurystyczne, którymi mogą różnić się poszczególne konkretne algorytmy, działające wg powyższego schematu, to sposób znajdowania parametrów $\mathbf{c}_k$ i σ_k.

[5]Dla uzupełnienia: algorytmy gradientowe też nie są w ogólności optymalne.

W kroku trzecim dla każdego z neuronów znajdujemy odległość do najbliższego z pozostałych neuronów, tj.:

$$d_k = \min_{\substack{j=1,\dots,K \\ j\neq k}} \|\mathbf{c}_k - \mathbf{c}_j\|, \tag{13}$$

a następnie ustawiamy

$$\sigma_k = \beta d_k, \tag{14}$$

gdzie $\beta \in [1,3]$ jest parametrem dobieranym eksperymentalnie, wpływającym na to, jak bardzo neurony nachodzą na siebie.

Znając już centra i szerokości neuronów, czyli wszystkie parametry dla nieliniowej warstwy sieci, można wyznaczyć teraz wagi $w_0,\dots,w_K$ rozwiązując liniowy problem optymalizacji, tj. minimalizując sumaryczny błąd kwadratowy (uczenie off-line)

$$\frac{1}{2}E^2 = \sum_{i=1}^{m} \frac{1}{2}\Big(y_{\text{RBF}}(\mathbf{x}_i) - y_i\Big)^2. \tag{15}$$

Z warunku koniecznego istnienia ekstremum

$$\forall k = 0,\dots,K \quad \frac{\partial \frac{1}{2}E^2}{\partial w_k} = 0, \tag{16}$$

otrzymujemy $K+1$ równań liniowych z $K+1$ niewiadomymi:

$$\begin{aligned} \frac{\partial \frac{1}{2}E^2}{\partial w_0} = 0 &\Leftrightarrow \sum_{i=1}^{m}\Big(y_{\text{RBF}}(\mathbf{x}_i) - y_i\Big) = 0; \\ \forall k = 1,\dots,K \quad \frac{\partial \frac{1}{2}E^2}{\partial w_k} = 0 &\Leftrightarrow \sum_{i=1}^{m}\Big(y_{\text{RBF}}(\mathbf{x}_i) - y_i\Big)\psi_k(\mathbf{x}_i) = 0. \end{aligned} \tag{17}$$

Zapisujemy powyższy układ równań macierzowo:

$$\underbrace{\sum_{i=1}^{m}\begin{pmatrix} 1 & \psi_1(\mathbf{x}_i) & \cdots & \psi_K(\mathbf{x}_i) \\ \psi_1(\mathbf{x}_i) & \psi_1^2(\mathbf{x}_i) & \cdots & \psi_K(\mathbf{x}_i)\psi_1(\mathbf{x}_i) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_K(\mathbf{x}_i) & \psi_1(\mathbf{x}_i)\psi_K(\mathbf{x}_i) & \cdots & \psi_K^2(\mathbf{x}_i) \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_K \end{pmatrix}}_{\mathbf{w}} = \underbrace{\sum_{i=1}^{m}\begin{pmatrix} y_i \\ y_i\psi_1(\mathbf{x}_i) \\ \vdots \\ y_i\psi_K(\mathbf{x}_i) \end{pmatrix}}_{\mathbf{b}}, \tag{18}$$

i jeżeli tylko macierz $\mathbf{A}$ jest odwracalna, tj. jeżeli $\det \mathbf{A} \neq 0$, to rozwiązaniem jest

$$\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}. \tag{19}$$

Tym sposobem wszystkie parametry sieci RBF o postaci (11) zostały znalezione.

2.3 Alternatywne przedstawienie rozwiązania

Powszechnie stosuje się nieco inny ale równoważny sposób przedstawienia macierzy $\mathbf{A}$ i $\mathbf{b}$ z poprzedniego punktu. Utwórzmy następującą macierz o wymiarach $(K+1)\times m$:

$$\mathbf{G} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \psi_1(\mathbf{x}_1) & \psi_1(\mathbf{x}_2) & \cdots & \psi_1(\mathbf{x}_m) \\ \psi_2(\mathbf{x}_1) & \psi_2(\mathbf{x}_2) & \cdots & \psi_2(\mathbf{x}_m) \\ \vdots & \dots & \ddots & \vdots \\ \psi_K(\mathbf{x}_1) & \psi_K(\mathbf{x}_2) & \cdots & \psi_K(\mathbf{x}_m) \end{pmatrix}. \tag{20}$$

Jest to tzw. *macierz baz* — macierz wartości funkcji bazowych ψ_k dla poszczególnych punktów danych. Numery baz zmieniają się wierszami, zaś numery przykładów kolumnami. Należy wyjaśnić, że pierwszy wiersz wypełniony jest jedynkami w związku z istnieniem wyrazu wolnego w_0, patrz schemat sieci

na Rys. 7 oraz wzór (11). Mówiąc inaczej, można umownie potraktować jedynki z pierwszego wiersza jako wartości nieistniejącej bazy ψ_0.

Łatwo sprawdzić, że macierze z poprzedniego punktu dają się przedstawić jako:

$$\mathbf{A} = \mathbf{G}\mathbf{G}^T, \tag{21}$$

$$\mathbf{b} = \mathbf{G}\mathbf{Y}, \tag{22}$$

gdzie $\mathbf{Y} = (y_1, y_2, \ldots, y_m)$ oznacza kolumnę wartości zmiennej y wziętą ze zbioru uczącego. Jak widać sumowanie obecne w poprzedniej reprezentacji macierzy $\mathbf{A}$ jest teraz realizowane poprzez iloczyn macierzy $\mathbf{G}$ oraz jej wersji transponowanej $\mathbf{G}^T$.

Stąd też poszukiwane rozwiązanie — wektor współczynników wagowych dla warstwy liniowej — można przedstawić jako:

$$\mathbf{w} = (\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{G}\mathbf{Y}. \tag{23}$$

2.4 Przykłady do ćwiczeń domowych

W ramach zadania domowego należy przeprowadzić aproksymację funkcji dwóch zmiennych

$$f(x_1, x_2) = \cos(x_1 x_2)\cos(2x_1) \tag{24}$$

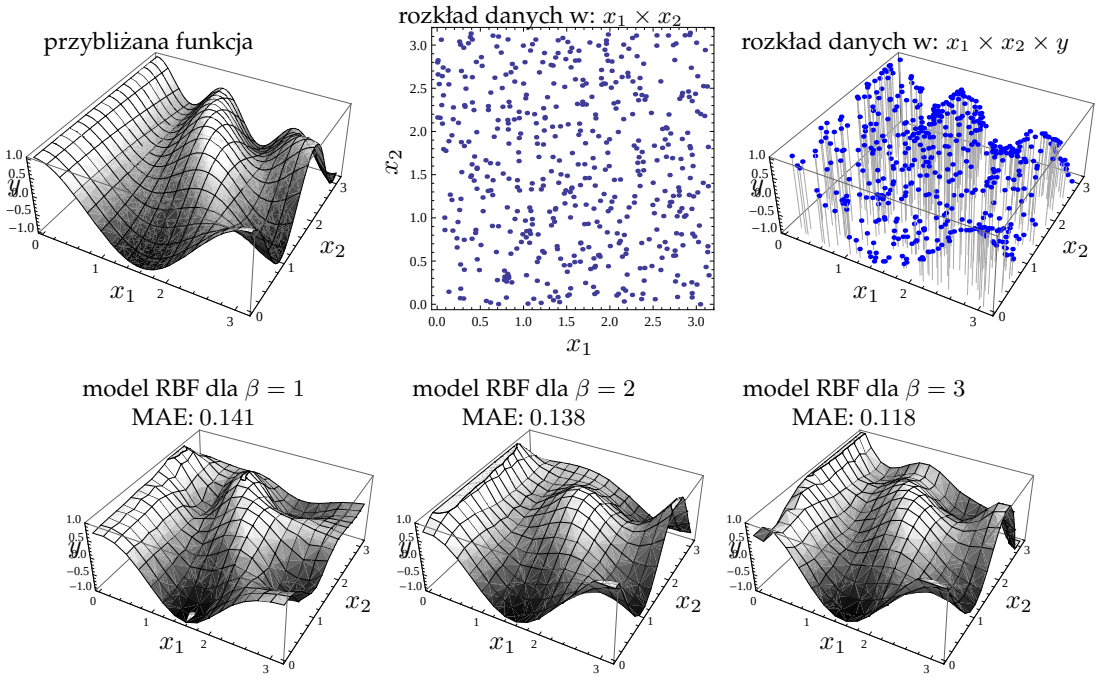
na podstawie zbioru zaczerpniętych z niej próbek (przykładów). Rozpatrywaną dziedzinę tej funkcji ograniczymy do kwadratu $[0, \pi] \times [0, \pi]$, przy czym eksperymenty przeprowadzone zostaną na trzech różnych rozkładach punktów $\mathbf{x}_i$ w ramach tego kwadratu (wskazówki na zajęciach):

1. rozkład jednostajny nad całym kwadratem $[0, \pi] \times [0, \pi]$,
2. rozkład nad pasem spiralnym wpisanym w kwadrat $[0, \pi] \times [0, \pi]$,
3. rozkład nad szachownicą wpisaną w kwadrat $[0, \pi] \times [0, \pi]$.

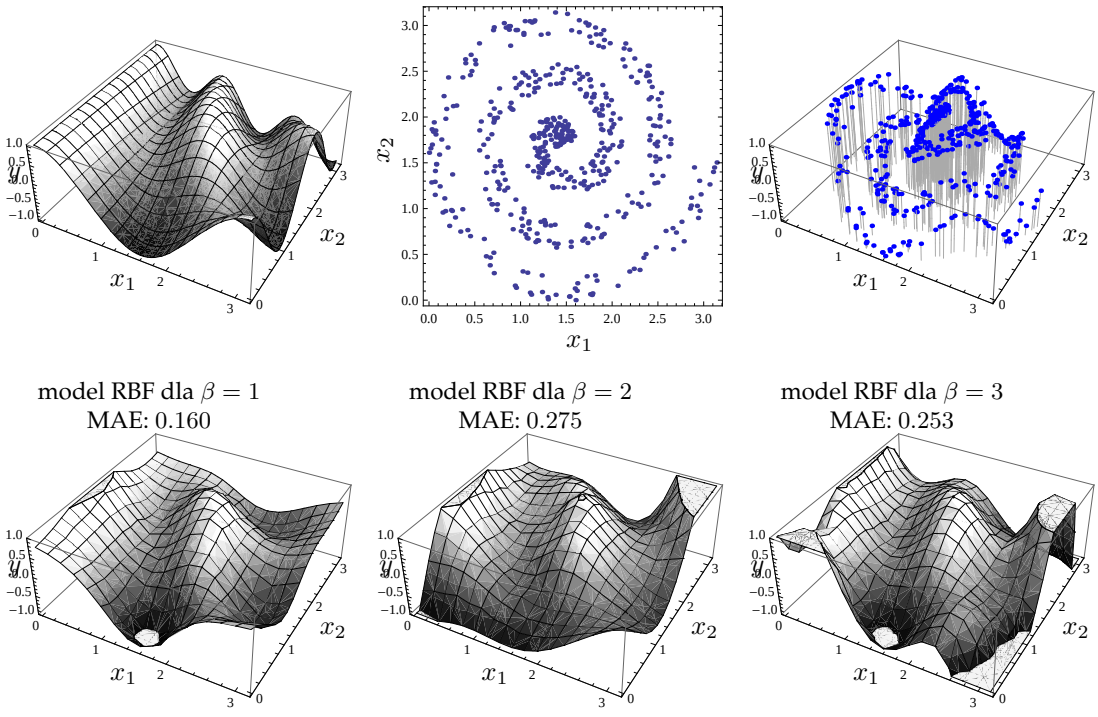
We wszystkich trzech przypadkach należy wygenerować zbioru o rozmiarze $m = 500$.

Przykładowe oczekiwane rozwiązania przedstawiono na rysunkach: 8, 11, 10. W górnym wierszu każdego z rysunków wykreślono dla przypomnienia przybliżaną funkcję oraz zaczerpnięty z niej zbiór danych (w rzucie na dziedzinę oraz w trzech wymiarach). W dolnym wierszu przedstawiono uzyskane modele RBF (przybliżenia) dla różnie dobranych wartości parametru β (stopień nachodzenia na siebie neuronów). Wykresy powierzchniowe modeli są celowo ograniczone do przeciwdziedziny $[-1, 1]$ dla ujednolicenia widoku z funkcją przybliżaną. Stąd też możliwe jest zaobserowanie w niektórych miejscach obcięć powierzchni modeli, gdy te wychodzą poza przedział $[-1, 1]$ (jest to tylko efekt wizualny, powierzchnie modeli są ciągłe). Dodatkowo dla informacji nad modelami podano wartości średniego błędu bezwzględnego MAE (ang. *Mean Absolute Error*) pomiędzy funkcją przybliżaną a przybliżeniem. Wartości te obliczono całkując numerycznie wg wzoru:

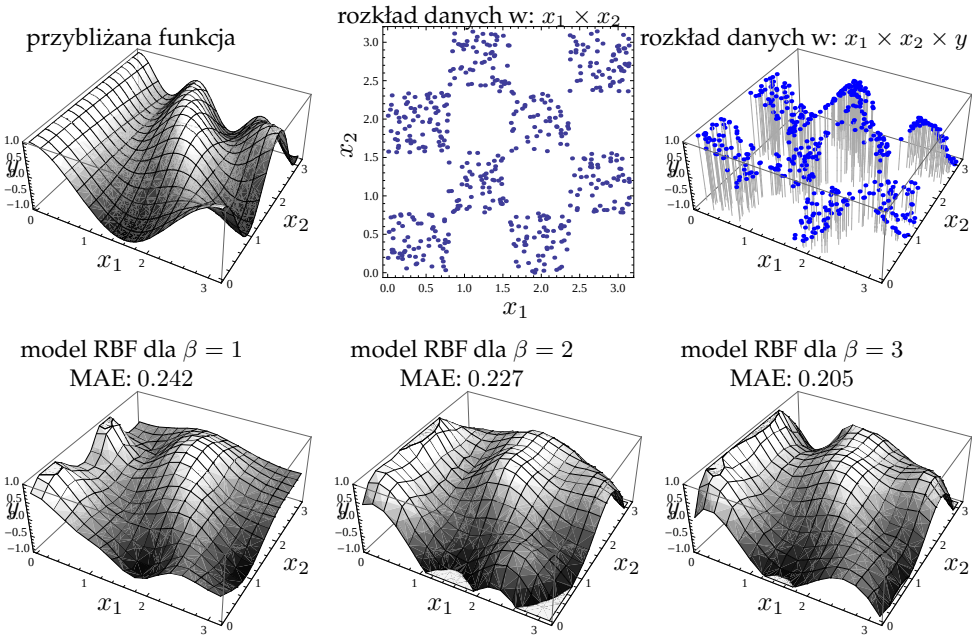
$$\mathrm{MAE} = \frac{1}{\pi^2}\int_0^\pi \int_0^\pi |f(x_1, x_2) - y_{\mathrm{RBF}}(x_1, x_2)| dx_1\, dx_2. \tag{25}$$



Rysunek 8: Dane dla rozkładu jednostajnego i modele RBF dla różnych wartości parametru β.

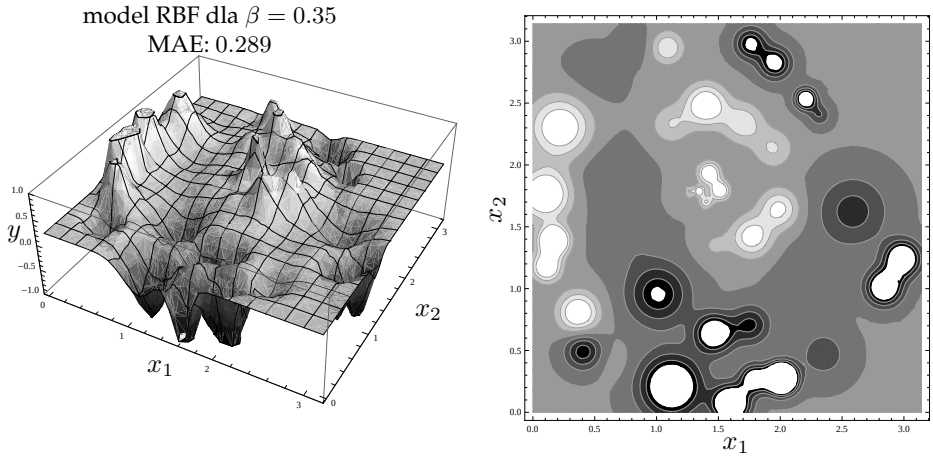


Rysunek 9: Dane dla rozkładu „spirala” i modele RBF dla różnych wartości parametru β.



Rysunek 10: Dane dla rozkładu „szachownica” i modele RBF dla różnych wartości parametru β.

Dla zobrazowania sposobu rozkładania się neuronów radialnych w dziedzinie i ich wpływu na wynikową powierzchnię modelu sporządzono dodatkowo Rys. 11 (poza celem zadania domowego). Dotyczy on przypadku, w którym punktu danych były rozłożone w formie pasa spiralnego. Model RBF przedstawiony na tym rysunku ma nadmiernie małą wartość parametru $\beta = 0.35$ — co prowadzi do sytuacji, że neurony mają bardzo wąski zakres działania (wystające górki i doliny), a poza ich położeniem model ma powierzchnię zbliżoną do funkcji stałej. Prawa strona Rys. 11 przedstawia warstwicowy wykres tego modelu pozwalający zaobserwować spiralne rozłożenie neuronów, jako wynik klasteryzacji przykładów w dziedzinie.



Rysunek 11: Model RBF (wraz z wykresem warstwicowym) dla małej wartości $\beta = 0.35$ nad rozkładem „spirala”.