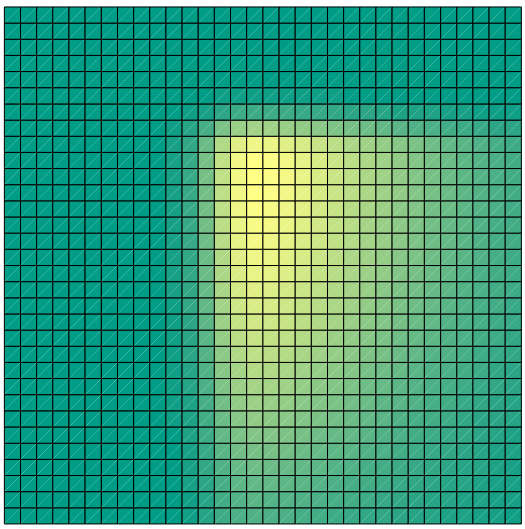
N o r b e r t J a n k o w s k i

Ontogeniczne sieci neuronowe w zastosowaniu do klasyfikacji danych medycznych



P R A C A D O K T O R S K A

POD KIERUNKIEM PROF. WŁODZISŁAWA DUCHA

Katedra Metod Komputerowych
Uniwersytetu Mikołaja Kopernika

Toruń 1999

Nie ma duszy bez pamięci

Jan Lebenstein

Spis treści

Spis rysunków

Spis tabel

Rozdział 1

Wstęp

W stronę uśmiechniętych maszyn[1] kierują się liczne badania ostatnich lat XX wieku, a z pewnością nadchodzący wiek XXI będzie kontynuował te tendencje. Początek istnienia komputerów to czas, w którym można było je znaleźć jedynie na uniwersytetach lub w innych instytucjach naukowo-badawczych. Wraz ze znacznym postępem technologicznym ogromnemu zwiększeniu uległa moc obliczeniowa komputerów, ich cena stała się przystępna dla kieszeni obywateli krajów rozwiniętych, stwarzając tym samym możliwości ich szerokiego zastosowania.

Obecna moc obliczeniowa komputerów pozwala już nie tylko efektywnie rozwiązywać problemy, których złożoność jest wielomianowa, ale również skutecznie próbować rozwiązywać wiele problemów NP-zupełnych, których do niedawna w ogóle nie można było rozwiązywać. Oczywiście rozwiązywanie problemów NP-zupełnych, w większości przypadków, sprowadza się do poszukiwania rozwiązań przybliżonych, ale na tyle dobrych, by były wręcz nieodróżnialne od rozwiązań idealnych, bądź stanowiły rozwiązania satysfakcjonujące, które umożliwią ich użycie w praktyce.

W realnych zastosowaniach na brak trudnych (NP-zupełnych) problemów nie można narzekać. Jest ich wręcz za dużo. Już choćby takie sztandarowe problemy jak szachy, czy problem komiwojażera, są na to dowodem. O więcej przykładów naprawdę nietrudno, wystarczy spojrzeć na typowe problemy w przemyśle, na przykład przeróżne problemy optymalizacyjne, czy niezwykle szeroki wachlarz problemów współczesnej medycyny, których rozwiązanie najczęściej polega na *inteligentnym* przetwarzaniu informacji.

Trzeba pamiętać jednak, iż moc obliczeniowa komputerów to jedynie czynnik niezbędny do rozwiązywania takich problemów. Rozwiązywanie trudnych problemów staje się możliwe przede wszystkim dzięki rozwojowi nowych metod obliczeniowych, które najczęściej stanowią połączenie pewnej wiedzy o problemie z metodami przetwarzania i wykorzystywania tej wiedzy. Taka metodologia postępowania jest dziś

[1] Tytuł książki prof. R. Tadeusiewicza [167].

spotykana w rozmaitych aplikacjach. Wystarczy wspomnieć tomografię komputerową, FMRI, czy scyntygrafię. Ogromną część problemów stanowią różnego typu analizy uprzednio zebranych danych, analizy obrazów, klasyfikacja i rozpoznawanie wzorców, prognozowanie. Różne gałęzie nauki, które zajmują się rozwiązywaniem tego typu problemów, można objąć wspólną nazwą *metod inteligencji obliczeniowej*. Do metod inteligencji obliczeniowej zaliczyć można sztuczne sieci neuronowe, uczenie maszynowe, metody regresji i estymacji, statystykę, teorie filtrów adaptacyjnych, modelowanie Bayesowskie, logikę rozmytą, teorię zbiorów przybliżonych, algorytmy ewolucyjne, metody drążenia danych, modelowanie koneksjonistyczne, neuroinformatykę. Większość modeli wyrosłych z powyższych dziedzin można także scharakteryzować jako *metody uczenia się z danych*[2].

Również i materiał poniższej pracy trudno sklasyfikować tylko do jednej z powyżej wspomnianych gałęzi metod inteligencji obliczeniowej. Choć niewątpliwie większość materiału jest bezpośrednio związana ze sztucznymi sieciami neuronowymi, to nietrudno dopatrzyć się elementów uczenia maszynowego, statystyki, teorii filtrów adaptacyjnych, czy metod wizualizacji.

Drugi rozdział stanowi obszerne omówienie różnych funkcji transferu sztucznych sieci neuronowych. Funkcje transferu mają ogromny wpływ na własności i tym samym możliwości sztucznych sieci neuronowych. Dlatego też w tym rozdziale zebrano informacje o wielu funkcjach transferu. Zaprezentowano również ich nowe, bardziej efektywne wersje, które można zastosować do różnych modeli.

Dokonano systematycznego omówienia funkcji aktywacji, podzielonych na funkcje bazujące na iloczynie skalarnym, mierze odległości (lub podobieństwa) i ich kombinacji. Po funkcjach aktywacji przedstawiono funkcje wyjścia: sigmoidalne, zlokalizowane i semi-centralne. Zaproponowane taksonomie są pierwszą tego typu próbą systematyzacji wiedzy o funkcjach realizowanych przez neuron. Następnie zostały przedstawione funkcje transferu, jako kombinacje różnych funkcji aktywacji z różnymi funkcjami wyjścia. Najpierw przedstawiono funkcje nielokalne, następnie lokalne, semi-lokalne i uniwersalne.

Kolejna część rozdziału obejmuje nowe funkcje transferu, które zostały nazwane funkcjami bicentralnymi. Zostały opisane funkcje bicentralne w formie podstawowej, jak i ich różne ciekawe rozszerzenia, które umożliwiają osiągnięcie jeszcze większej elastyczności poprzez wykorzystanie obrotu w wielowymiarowej przestrzeni, czy delokalizację. W końcowej części rozdziału dokonano tabelarycznego porównania ważnych własności funkcji transferu omówionych w tym rozdziale.

Kolejny rozdział omawia różne aspekty sieci neuronowych z radialnymi funkcjami bazowymi (RBF). Początek rozdziału to omówienie podstaw sieci RBF. Następnie przedstawione zostały różne metody inicjalizacji sieci typu RBF. Potem omówiono standardowe, jak i mniej znane metody uczenia sieci RBF. Zaprezentowane zostały różne człony regularyzacyjne. W końcowej części dokonano porównania sieci typu MLP z sieciami RBF. Pokazano także transformację danych, która umożliwia nielosową inicjalizację sieci typu MLP.

[2] *Uczenie się z danych (ang. Learning from data)* — tytuł książki V. Cherkasskiego i F. Muliera

Rozdział czwarty obejmuje omówienie sieci ontogenicznych i opis ontogenicznej sieci IncNet. Pierwsza część omawia modele, które umożliwiają usuwanie wag lub neuronów ze struktury sieci neuronowej. Druga część rozdziału omawia modele, których struktura rozrasta się podczas procesu adaptacji. Wskazano liczne wady, zalety i ograniczenia przedstawionych modeli ontogenicznych. Omówiona została również sieć z przydziałem zasobów (RAN).

Pozostała część rozdziału to wstęp i omówienie sieci *Incremental Network* (IncNet). Opisano, jak można zastosować filtr EKF do uczenia sieci typu RBF. Zaproponowano także nową odmianę rozszerzonego filtra EKF o mniejszej złożoności obliczeniowej, dzięki której można prowadzić adaptację bardziej złożonych problemów. Zaproponowano nowe, statystyczne metody kontroli złożoności sieci neuronowych. Do zastosowań klasyfikacyjnych została zaprezentowana sieć, która składa się z klastra podsieci IncNet i modułu decyzyjnego.

Następnie opisano możliwości diagnostyczne różnych współczynników, które są wyznaczane przez wspomniany klaster sieci IncNet i moduł decyzyjny, w tym także prawdopodobieństwa przynależności klasyfikowanych wektorów do poszczególnych cech. Opisano także własności różnych innych możliwości kontroli sieci IncNet. W końcowej części rozdziału zaproponowano używanie *przedziałów ufności*, które stanowią bardzo silną alternatywę dla reguł logicznych. Zaproponowano także bardzo ciekawe metody wizualizacji w oparciu o przedziały ufności, jak i ich rozwinięcia, których celem jest wspomaganie procesu diagnozy, szczególnie w medycynie.

Rozdział piąty prezentuje zastosowania sieci IncNet dla realnych i sztucznych danych. W pierwszej części tego rozdziału zebrano i omówiono wiele aspektów wstępnego przetwarzania danych i porównywania modeli. Temat ten jest niemal zawsze przedstawiany szczątkowo przez większość książek, które opisują zagadnienia sztucznych sieci neuronowych. Fakt ten doprowadził do powstania wielu niejednoznaczności i rożnych interpretacji metodologii wstępnego przetwarzania danych i porównywania modeli. W rozdziale omówione zostały metody porównania modeli, różne transformacje danych (standardowe jak i nowe), problemy wartości nietypowych i wartości brakujących, oraz ważniejsze aspekty metod selekcji cech.

Pierwszy przykład zastosowania sieci IncNet, to analiza danych psychometrycznych. Celem jest klasyfikacja pacjentów do odpowiednich typów nozologicznych w oparciu dokonywane testy psychometryczne i w rezultacie poprawienie jakości klasyfikacji dokonywanej obecnie przez psychologów. Dokonano szczegółowej analizy otrzymanych rezultatów dla różnych końcowych sieci IncNet. Kolejne zastosowania sieci IncNet, to problemy klasyfikacji raka piersi, zapalenia wątroby, cukrzycy, zapalenia wyrostka i chorób tarczycy. Wszystkie zastosowania zostały omówione i porównane z innymi, najlepszymi obecnie klasyfikatorami dla danych baz.

Jako uzupełnienie powyżej wspomnianych zastosowań zostały dołączone zastosowania sieci IncNet w problemach aproksymacyjnych. Zastosowano sieć IncNet do aproksymacji czterech różnych funkcji i porównano rezultaty z różnymi modelami.

Rozdział 2

Funkcje transferu

Wybór funkcji transferu ma niezwykle duży wpływ na możliwości działania sieci neuronowych. Chociaż funkcje sigmoidalne jako funkcje transferu są powszechnie stosowane nie ma powodu, aby to one były optymalne we wszystkich przypadkach. Przedstawione zostaną tu zalety i wady wielu różnych funkcji transferu jak i szeregu nowych funkcji transferu posiadających większe możliwości. Przedstawiona zostanie również propozycja taksonomii funkcji aktywacji i funkcji wyjścia. Będą opisane również uniwersalne funkcje, które poprzez zmianę parametrów stają się lokalne lub nielokalne, albo nielokalne w pewnych podprzestrzeniach, a w innych podprzestrzeniach lokalne. Również i inne funkcje zostaną zaprezentowane, włączając w to funkcje bazujące na nieeuklidesowej mierze odległości. Następnie wprowadzone zostaną funkcje bicentralne, które powstają jako liniowy produkt par funkcji sigmoidalnych. Taki produkt składający się z N funkcji bicentralnych w N wymiarowej przestrzeni jest w stanie reprezentować o wiele większą klasę gęstości prawdopodobieństw wejściowej przestrzeni wektorów, niż np. typowa wielowymiarowa funkcja gaussowska. Przedstawione są też różne możliwości rozszerzeń funkcji bicentralnych, które mogłyby stanowić pewien złoty środek pomiędzy złożonością samej sieci, a jej możliwością do uczenia się. Funkcje bicentralne i ich rozszerzenia mogą być z powodzeniem stosowane do różnych sieci neuronowych w szczególności do jakich jak RBFN, RAN, IncNet i FSM. Z kolei, używając takich funkcji i wymuszając ostre granice (duże skosy), podążamy do logicznej interpretacji sieci neuronowej.

Przykłady zastosowań w diagnostyce medycznej jak i aproksymacji funkcji wielowymiarowych ewidentnie pokazują, że użycie funkcji bicentralnych jako funkcji transferu daje lepsze wyniki, niż innych funkcji.

Powstanie sztucznych sieci neuronowych jako systemów adaptacyjnych było początkowo motywowane możliwościami przetwarzania informacji mózgu ludzkiego [85, 10, 154]. Pojedyncze sztuczne neurony, jak i architektury sztucznych sieci neuronowych mają niewiele wspólnego z prawdziwą biologiczno–logiczną budową mózgu. Sztuczne sieci neuronowe są sieciami złożonymi z prostych elementów, nazywanych neuronami, które posiadają parametry adaptacyjne W. Modyfikacje tych parametrów

prowadzą do uczenia się przez sieć odwzorowania wektora X z przestrzeni wejściowej do przestrzeni wyjściowej $Y = A_W(X)$. Ze statystycznego punktu widzenia systemy adaptacyjne powinny być estymatorami rozkładu prawdopodobieństwa $p(X, Y)$ lub chociaż prawdopodobieństwa $p(Y|X)$. Do estymacji granic decyzji rozkładu prawdopodobieństwa konieczna jest *adaptowalność* kształtu powierzchni funkcji transferu i właśnie to stanowi o *sile* adaptacyjnej sieci.

Sztuczne sieci neuronowe są systemami, które posiadają mac obliczeniową komputera uniwersalnego, tj. mogą realizować dowolne odwzorowanie z jednej przestrzeni (wejściowej) do drugiej (wyjściowej). Różnią się pod wieloma względami, lecz wspólną cechą jest obliczanie wartości funkcji transferu przez każdy neuron. Pierwszymi modelami sztucznych sieci były sieci logiczne [128] lub urządzenia progowe, obliczające funkcje krokową. Funkcje krokowe były następnie uogólniane do funkcji o kształcie sigmoidalnym. Pokazano też, że sieć neuronowa z jedną warstwą ukrytą z funkcjami sigmoidalnymi jest uniwersalnym aproksymatorem [33, 89], tj. może aproksymować dowolną ciągłą funkcję z dowolną dokładnością przy wystarczalnej liczbie neuronów. Taką samą własność mają sieci z funkcjami gaussowskimi, użytymi w miejsce funkcji sigmoidalnych [81, 145].

Nowy typ funkcji transferu zwanych *gaussian bars* został zaproponowany przez Hartmana i Keelera[80]. Pao zaprezentował inny typ sieci (*functional link networks*) [144], w którym wykorzystano kombinacje różnych funkcji takich, jak wielomiany, funkcje periodyczne, funkcje sigmoidalne i gaussowskie. Haykin i Leung proponują użycie *rational transfer functions* i prezentują bardzo dobre wyniki przy użyciu tych funkcji transferu [120]. W pracy Dorffnera [36] prezentowane są funkcje stożkowe, które gładko zmieniają się od funkcji o kształcie sigmoidalnym do funkcji zbliżonej do funkcji gaussowskiej. Można też użyć funkcji Lorentzowskiej, jako uproszczenia funkcji gaussowskiej zaproponowanej przez Girauda i in. [73]. Te prace, jak i sporo innych, pokazują, iż wybór funkcji transferu jest istotny i tak samo ważny jak i dobór architektury sieci czy algorytmu uczenia.

Sieci neuronowe są używane do aproksymacji rozkładu prawdopodobieństwa *a priori*, dla klasyfikacji lub do aproksymacji gęstości prawdopodobieństwa zbioru danych treningowych [10, 154]. Żadne z powyżej wspomnianych funkcji nie są wystarczające do reprezentacji rozkładu prawdopodobieństwa wielowymiarowej przestrzeni wejściowej przy użyciu małej liczby parametrów. Problem uczenia, z geometrycznego punktu widzenia, można przestawić jako cel, którym jest wybór takiej przestrzeni funkcji i ich parametrów, które będą jak najbardziej elastyczne przy użyciu jak najmniejszej liczby parametrów adaptacyjnych. Konsekwencje tych faktów wydają się być nadal nieznane niektórym badaczom.

Żadne z powyżej wspomnianych funkcji transferu nie są wystarczająco elastyczne do opisu powierzchni decyzji pewnych danych z wielowymiarowej przestrzeni wejściowej, przy użyciu małej liczby parametrów adaptacyjnych. Do testowania metod adaptacyjnych statystycy preferują sztuczne dane [84, 65]. Jest oczywiste, iż pewne rozkłady danych są łatwo aproksymowane przy użyciu funkcji zlokalizowanych (np. funkcji gaussowskich), a inne rozkłady są prostsze w aproksymacji wykorzystując funkcje nielokalne (np. funkcje sigmoidalna z aktywacją w postaci liniowej kombinacji wejść). W [84] rozważany był problem o N wymiarowej przestrzeni wejściowej, w którym wektory znajdujące się wewnątrz pewnej sfery należą do jednej klasy, a na

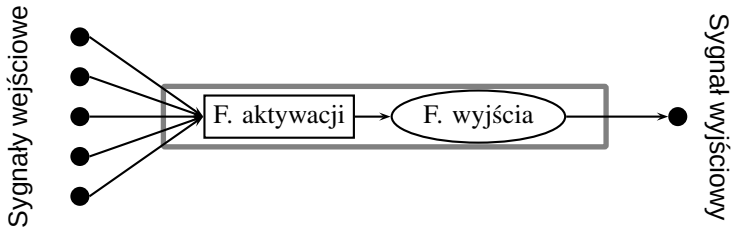
zewnątrz do drugiej. Łatwo zauważyć, iż do rozwiązania takiego problemu wystarczy jedna wielowymiarowa funkcja gaussowska z $2N$ parametrami adaptacyjnymi (na centrum i rozmycia). Jednakże rozwiązanie tego samego problemu wymaga wielu hiperpłaszczyzn tworzonych przez funkcje sigmoidalne. Najprostsza możliwa sieci MLP, która rozwiązała by powyższy problem musi skonstruować sympleks przy użyciu N funkcji sigmoidalnych i jednego dodatkowego neuronu na wygładzenie powierzchni, co stanowi $N^2 + N$ parametrów adaptacyjnych i znacznie komplikuje proces uczenia. Z kolei, w innym problemie, gdy do pierwszej kolasy zakwalifikować punkty z rogu układu współrzędnych, ograniczając obszar płaszczyzną $(1, 1, ..., 1)$, to wystarczy jedna płaszczyzna ($N + 1$ parametrów), aby rozdzielić dwie klasy. Natomiast znacznie trudniej jest rozwiązać problem przy użyciu funkcji gaussowskich. Umieszczając jedną w centrum obszaru i $N + 1$ po rogach wymaga $2N(N + 2)$ parametrów nie rozwiązuje się idealnie problemu, a i znacznie utrudnia się proces adaptacji. Usprawnianie algorytmów uczenia lub struktur sieci nie będą wystarczające, gdy obszary decyzyjne będą produktem funkcji *sferycznych* lub hiperpłaszczyzn.

Poniżej rozważane są różne funkcje transferu dla sztucznych sieci neuronowych. Jednak nie jest celem tego rozdziału przedstawienie wszelkich prac, jakie były prowadzone na ten temat. Anderson [5] uzasadnia użycie funkcji sigmoidalnych dla motoneuronów, lecz przejście od neuronów impulsowych (*ang. spiking neurons*) kory mózgowej (jej asocjacyjnej funkcji) do modelu, w którym używa się ciągłych funkcji transferu, nie jest trywialne (teoretyczne wprowadzenie w modele oparte o neurony impulsowe można znaleźć w [126]). Bardzo ciekawym aspektem jest też budowanie neuronów analogowych lub modeli hardwareowych [133, 179, 91], lecz ten temat również wykracza już po za główny temat pracy. Nie będą też rozważane funkcje używane w modelach asocjacyjnych, takie jak funkcje monotoniczne [111, 136, 184, 180, 181], funkcje periodyczne [182, 182, 109, 138] i neurony chaotyczne [71, 183]. Te ostatnie mogą być bardziej przydatne w neurobiologi i mogą unikać złudnych lokalnych minimów funkcji błędu. Także w rozmytych sieciach neuronowych używa się specjalnych funkcji transferu, te również zostaną pominięte. Pominięty zostanie też model *neuronu złożonego* (por. [165]).

Ciekawą rzeczą okazało się sporządzenie systematycznego przeglądu przeróżnych funkcji transferu dla sieci neuronowych, jak i taksonomii funkcji aktywacji i wyjścia, ponieważ, jak dotąd, informacje te w literaturze były zupełnie rozproszone poza nielicznymi wyjątkami, które prezentują funkcje alternatywne do funkcji sigmoidalnej. Część z funkcji, które zostały zaprezentowane poniżej, nigdy nie były jeszcze użyte.

W poniższym podrozdziale przedstawiono ogólne pojęcia związane z opisywaniem funkcji transferu. W następnym podrozdziale przedstawiono szeroki opis funkcji aktywacji neuronu. Opis obejmuje szeroki wachlarz różnych miar odległości. Kolejny podrozdział przedstawia przeróżne funkcje wyjścia, po czym następuje podrozdział, w którym przedstawiono różne funkcje transferu, podzielone na kilka grup. Porównywanie rezultatów uzyskanych za pomocą różnych funkcji transferu jest przedsięwzięciem bardzo trudnym. Różne funkcje mogą być użyte w bardzo różnych sieciach. Również i sposób inicjalizacji sieci może prowadzić do bardzo zróżnicowanych wyników. Tym samym, nie jest możliwe w pełni obiektywne i jednoznaczne porównanie takich wyników.

2.1. Funkcje realizowane przez neuron



Rysunek 2.1: Model neuronu

Za przetwarzanie sygnału przez każdy neuron odpowiedzialne są dwie funkcje — funkcja aktywacji i funkcja wyjścia. *Funkcja aktywacji* oblicza wartość całkowitego sygnału wejściowego neuronu. W tym podrozdziale będzie to liniowa kombinacja sygnałów wejściowych, choć w podrozdziale 2.2.1 zostaną przedstawione bardzo różne funkcje odległości, które będą mogły zastąpić ową liniową kombinację.

Jeśli neuron i jest połączony z neuronem j (gdzie $j = 1, \ldots, N$) i wysyła sygnał o wartości x_j z *siłą* połączenia równą W_{ij}, to całkowita aktywacja I_i będzie równa:

$$I_i(\boldsymbol{x}; \boldsymbol{W}) = \sum_{j=1}^{N} W_{ij} x_j \tag{2.1}$$

Powyższa liniowa kombinacja wejść jest najczęściej stosowaną funkcją aktywacji używaną w sieciach MLP.

Drugą funkcją przetwarzaną przez neuron jest *funkcja wyjścia* $o(I)$. Te dwie funkcje razem decydują o wartości wyjściowej neuronu. Całość przetwarzania informacji przez neuron odbywa się w N wymiarowej przestrzeni wejściowej, która jest także nazywana przestrzenią parametrów. Złożenie funkcji aktywacji z funkcją wyjścia nazywa się *funkcją transferu* $o(I(\mathbf{x}))$. Funkcje aktywacji i wyjścia dla warstwy wejściowej i wyjściowej mogą być inne niż dla warstw ukrytych. Zazwyczaj stosowane są funkcje liniowe w warstwie wejściowej i wyjściowej, a dla warstw ukrytych wybiera się nieliniowe funkcje transferu. Pewne funkcje transferu nie mogą być w naturalny sposób podzielone na funkcję aktywacji i funkcje wyjścia. Za lokalną funkcję transferu będzie się przyjmować funkcję, której wartości będą istotnie różne od zera (tj. $|o(I(\mathbf{x}))| > \epsilon$ dla pewnego ϵ) dla wartości $\mathbf{x}$ leżących na skończonym obszarze przestrzeni wejściowej. To oznacza, że lokalny charakter funkcji transferu będzie zależał nie tylko od funkcji wyjścia, ale również od funkcji aktywacji.

Pierwsze modele sieci neuronowych zaproponowane w pracy McCulloch'a i Pitts'a [128] wykorzystywały w przetwarzaniu funkcje logiczne. Funkcja wyjścia w takim modelu była funkcją schodkową $\Theta(I; \theta)$, która przyjmowała wartość 0 poniżej progu

θ i 1 powyżej progu:

$$\Theta(I;\theta) = \begin{cases} 1 & I > \theta \\ 0 & I \leqslant \theta \end{cases} \tag{2.2}$$

Używanie funkcji progowych było motywowane analizą logicznego działania podukładów komputerów, jak i wyobrażaniem sposobu pracy mózgu, jako podobnego do sposobu przetwarzania informacji w strukturach składających się z elementów przełącznikowych (logicznych).

W zasadzie można dokonywać dowolnych obliczeń przy użyciu neuronów logicznych (tj. używających funkcji logicznych). Trzeba wtedy rzeczywiste wartości dyskretyzować i użyć neuronów logicznych do uczenia ich reprezentacji bitowej. Ogromną zaletą korzystania z logicznych elementów jest możliwość szybkiego przetwarzania takiej informacji, jak również możliwość efektywnej realizacji takich funkcji hardwareowo. Granice decyzji, otrzymane w wyniku użycia neuronów logicznych są hiperpłaszczyznami obróconymi przez parametry W_{ij}. Wtedy sieć oparta o takie elementy dzieli przestrzeń wejściową na wielościany (czasem nieskończone).

Funkcje wieloschodkowe stanowią etap pośredni pomiędzy funkcjami schodkowymi, a funkcjami semi-liniowymi. Liczba progów funkcji wieloschodkowej jest określona, a samą funkcję można zdefiniować poprzez:

$$\sigma_m(I) = y_i \quad \text{dla} \quad \theta_i \leqslant I < \theta_{i+1} \tag{2.3}$$

Aby uniknąć konstrukcji warunkowych dla stałych różnic $\theta = \theta_i - \theta_{i+1}$ wieloschodkowe funkcje można implementować efektywnie przy użyciu wektorów schodków $\boldsymbol{\nu}$ i arytmetyki stałopozycyjnej do konwersji przeskalowanych wartości wejściowych do danej przestrzeni wyjściowej: $\boldsymbol{\nu}\,[\Theta\,(1 + \mathrm{Int}[(I - \theta_1)/\theta])]$, gdzie θ_1 jest pierwszym progiem. Zamiast funkcji wieloschodkowej stosuje się funkcje semi-liniowa:

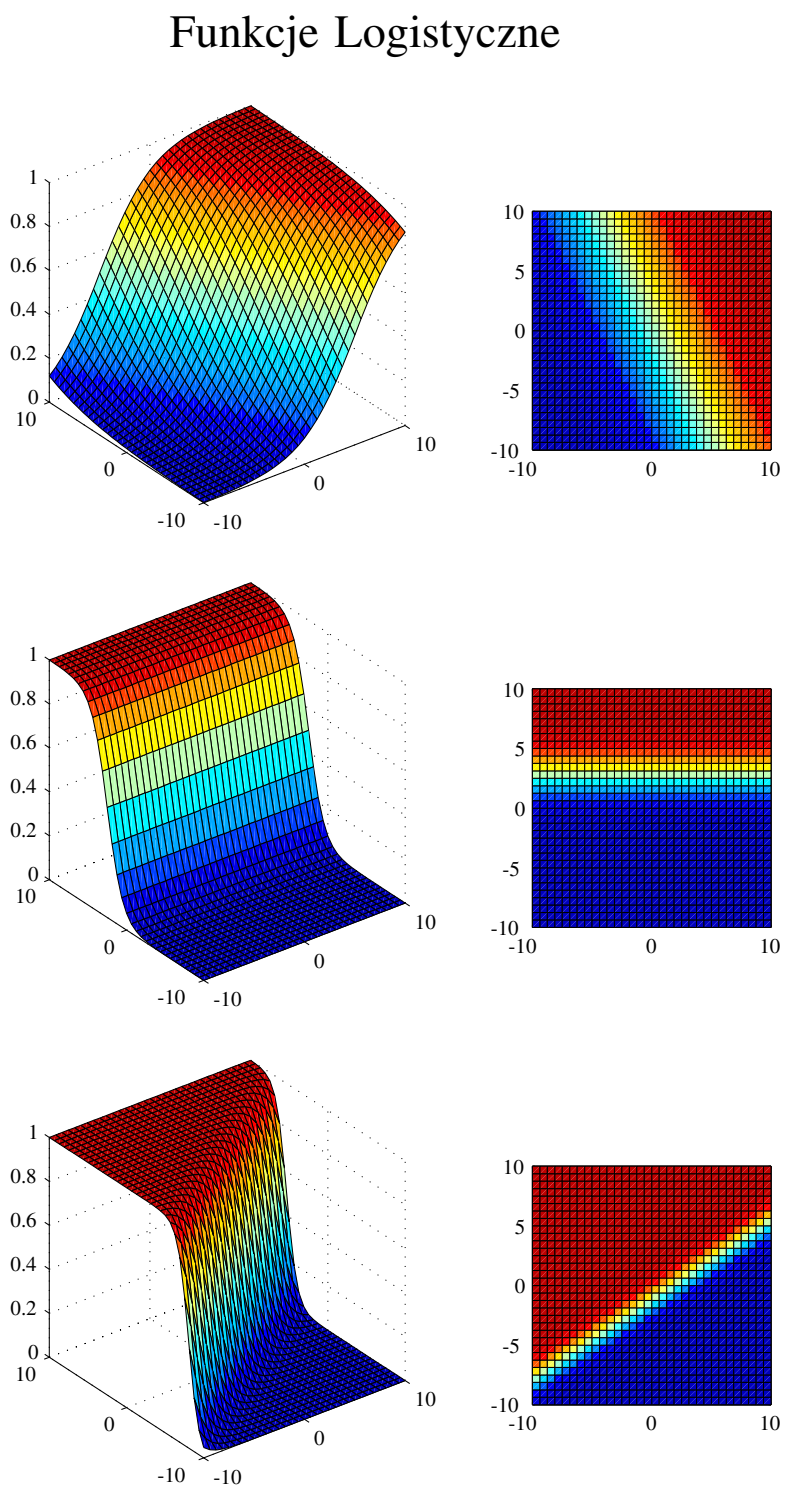
$$s_l(I;\theta_1,\theta_2) = \begin{cases} 0 & I \leqslant \theta_1 \\ (I - \theta_1)/(\theta_2 - \theta_1) & \theta_1 < I \leqslant \theta_2 \\ 1 & I > \theta_2 \end{cases} \tag{2.4}$$

Te funkcje zostały później uogólnione do funkcji *logistycznej*, powszechnie spotykanej w literaturze (patrz rys. 2.2):

$$\sigma(I/s) = \frac{1}{1 + e^{-I/s}} \tag{2.5}$$

Stała s określa skos funkcji logistycznej wokół jej liniowej części. Istnieje cała grupa różnych funkcji o kształcie podobnym do funkcji logistycznej nazwana *funkcjami sigmoidalnymi*. W granicy, gdy skos s dąży do nieskończoności wszystkie funkcje sigmoidalne przechodzą w funkcję schodkową.

Kombinacja liniowej aktywacji, jako kombinacji (2.1) z funkcją logistyczną, daje najbardziej popularną spośród funkcji transferu sieci neuronowych. Kombinacje funkcji



Rysunek 2.2: Funkcje logistyczne

sigmoidalnych z liniową aktywacją dają w rezultacie funkcję nielokalną, choć nic nie stoi na przeszkodzie by sigmoidalnych funkcji wyjściowe użyć w kombinacji z innymi lokalnymi funkcjami aktywacji (por. równania (2.62–2.65)), tworząc lokalną funkcję transferu.

Ciągle panuje powszechne przekonanie, że aktywność neuronów biologicznych ma wiele wspólnego z funkcjami sigmoidalnymi, choć nie jest to powód, dla którego funkcje sigmoidalne są tak popularne. Z wyjątkiem paru neurobiologicznych inspiracji, funkcje sigmoidalne mogą mieć uzasadnienie statystyczne [10, 102].

Rozważmy problem klasyfikacji w N wymiarowej przestrzeni z dwiema klasami o normalnym rozkładzie z równymi macierzami kowariancji

$$p(\mathbf{x}|C_k) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}}_k)^T\Sigma^{-1}(\mathbf{x}-\bar{\mathbf{x}}_k)\right\} \tag{2.6}$$

Korzystając z twierdzenia Bayesa prawdopodobieństwo *a posteriori* dla pierwszej klasy jest określone przez:

$$p(C_1|\mathbf{x}) = \frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_1)P(C_1)+p(\mathbf{x}|C_2)P(C_2)} = \frac{1}{1+\exp(-y(\mathbf{x}))} \tag{2.7}$$

gdzie $P(C_k)$ jest prawdopodobieństwem klas *a priori*, a funkcja $y(\mathbf{x})$ jest zdefiniowana przez:

$$y(\mathbf{x}) = \ln\frac{p(\mathbf{x}|C_1)P(C_1)}{p(\mathbf{x}|C_2)P(C_2)} \tag{2.8}$$

Mamy równość: $p(C_2|\mathbf{x}) = 1-p(C_1|\mathbf{x})$. Prowadzi to do logistycznej funkcji wyjścia z dość skomplikowaną funkcją aktywacji. Takie funkcje są używane w logistycznej analizie dyskryminacyjnej [4]. Dla problemów więcej niż dwuklasowych można użyć znormalizowanej funkcji eksponencjalnej (czasem zwanej *softmax*):

$$p(C_k|\mathbf{x}) = \frac{\exp(y_k(\mathbf{x}))}{\sum_i \exp(y_i(\mathbf{x}))} \tag{2.9}$$

Po takiej normalizacji wartości $p(C_k|\mathbf{x})$ mogą być interpretowane jako prawdopodobieństwa.

Innym uzasadnieniem racjonalności funkcji sigmoidanych [41] może być fakt, iż wartości wejściowe pochodzą zazwyczaj z obserwacji, które nie są całkiem dokładne, dlatego można zamiast wartości $\bar{y}$ użyć wartość rozkładu Gaussa $G_y = G(y;\bar{y},s_y)$ wokół $\bar{y}$ z rozmyciem s_y. Rozkład ten można też traktować jako funkcje przynależności rozmytej liczby G_y [116]. Skumulowana funkcja rozkładu wygląda natomiast tak:

$$p(x-\bar{y}) = \int_{-\infty}^{x} G(y;\bar{y},s_y)dy = \frac{1}{2}\left[1+\mathrm{erf}\left(\frac{x-\bar{y}}{s_y\sqrt{2}}\right)\right] \approx \sigma\left(\frac{x-\bar{y}}{T}\right) \tag{2.10}$$

gdzie erf jest funkcją błędu, a $T = \sqrt{2}s_y/2.4$. Dokładność tej aproksymacji jest nie gorsza niż 0.02. Skumulowany rozkład $p(x - \bar{y})$ może być interpretowany jako prawdopodobieństwo zajścia reguły $R_x(z)$ wtedy i tylko wtedy gdy $z \leqslant x$ jest prawdą, tj. $p(R_x|G_y) = p(x - \bar{y})$.

W następnym podrozdziale przedstawione zostaną różne typy funkcji aktywacji.

2.2. Funkcje aktywacji

Liniowa kombinacja wejść, w literaturze angielskiej zwana *fan-in activation* (2.1), jako aktywacja jest stosowana nie z powodów inspiracji biologicznych, lecz dlatego, że kontury o stałej wartości $I(\mathbf{x}) = const$ formują hiperpłaszczyznę. Metody statystyczne klasyfikacji mogą być podzielone na grupy. Pierwszą grupę stanowią metody bazujące na analizie dyskryminacyjnej, które używają hiperpłaszczyzn lub innych powierzchni do podziału przestrzeni wejściowej. Druga grupa obejmuje metody klasteryzacji i metody oparte na podobieństwie, które korzystają z pewnych miar odległości. Stąd też mamy do czynienia z trzema różnymi typami funkcji aktywacji:

- **Kombinacja liniowa** (iloczyn skalarny) $I(\mathbf{x}; \mathbf{w}) \propto \mathbf{w}^T \cdot \mathbf{x}$ (używana w sieciach MLP).
- **Miary odległości** jako aktywacje, lub ogólniej miary podobieństwa, $D(\mathbf{x}; \mathbf{t}) \propto ||\mathbf{x} - \mathbf{t}||$, wyznaczają podobieństwo wektora $\mathbf{x}$ do wektora $\mathbf{t}$.
- **Kombinacje** dwóch powyższych aktywacji, $A(\mathbf{x}; \mathbf{w}, \mathbf{t}) \propto \alpha\, \mathbf{w}^T \cdot \mathbf{x} + \beta\, ||\mathbf{x} - \mathbf{t}||$,

Taksonomia przeróżnych funkcji aktywacji została zaprezentowana na rysunku 2.3.

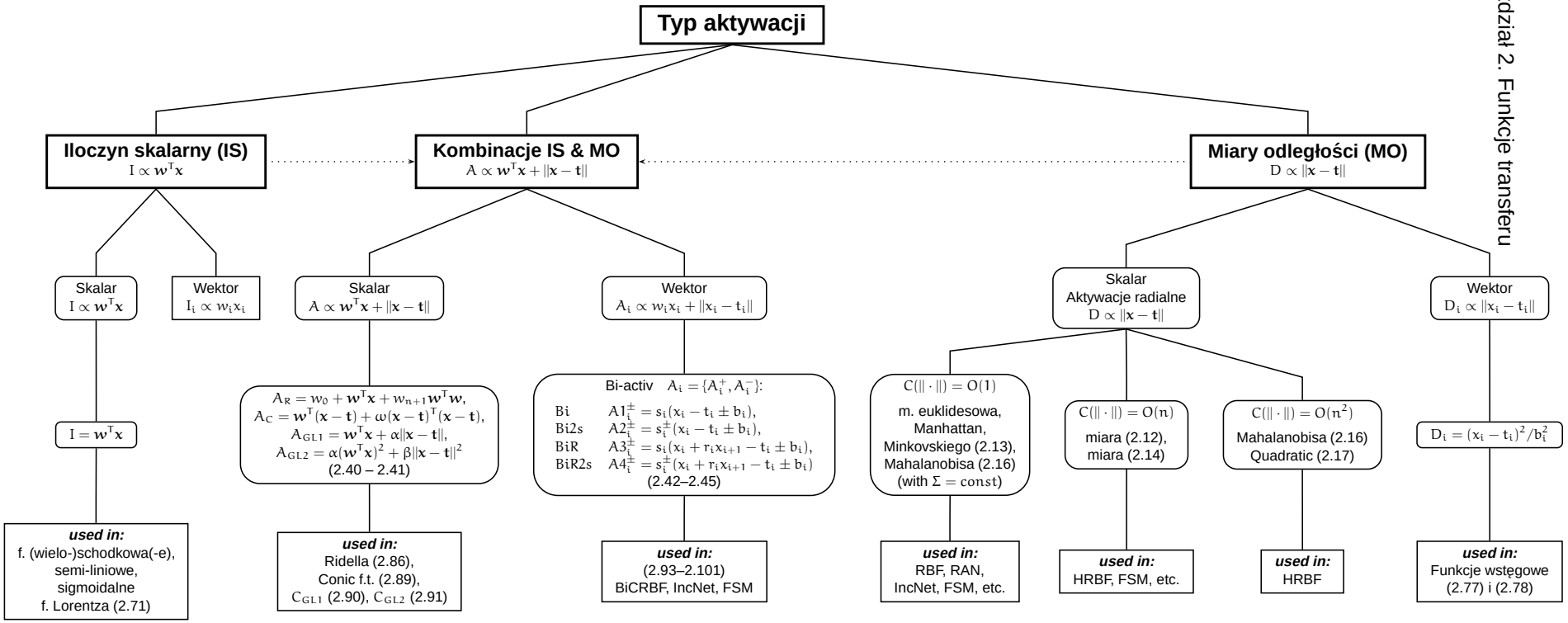
W każdym przypadku końcowa aktywacja składa się ze skalaru lub wektora wartości. Na przykład typowa funkcja odległości $D(\mathbf{x}, \mathbf{t})$ daje jako wynik skalar, choć mogą być używane jako wektor $D_i(x_i, t_i)$, gdzie $D_i(x_i, t_i)$ może być zdefiniowane jako:

$$D_i(x_i, t_i, b_i) = (x_i - t_i)^2/b_i^2 \tag{2.11}$$

Kwadrat funkcji aktywacji jest formą kwadratową. Uznając wszystkie parametry takiej formy za niezależne i przekształcając do formy kanonicznej, mamy:

$$I^2(\mathbf{x}; \mathbf{w}) \sim D^2(\mathbf{x}; \mathbf{t}, \mathbf{a}) = \sum_i^N a_i(x_i' - t_i)^2 \tag{2.12}$$

gdzie zmienne x_i' są liniowymi kombinacjami oryginalnych zmiennych x_i i odpowiadają pseudo-euklidesowej mierze odległości. Jeśli parametry a_i są dodatnie i przyjmiemy $a_i = 1/b_i^2$, to otrzymuje się miarę euklidesową z hiperelipsoidalnymi konturami dla stałych wartości miary. Kwadrat liniowej kombinacji wejść był użyty do *Lorentzowskiej* funkcji transferu (2.71, rys. 2.16). Lorentzowska funkcja nie ma elipsoidalnych konturów, powierzchnie są nielokalne, a kontury tworzą kształty okienkowe (tj. wycinają obszar okienkowy).



Rysunek 2.3: Taksonomia funkcji aktywacji. $C(\|\cdot\|)$ jest liczbą parametrów wolnych normy $\|\cdot\|$.

2.2.1. Miary odległości i podobieństwa jako funkcje aktywacji.

Drugą grupę funkcji aktywacji stanowią aktywacje oparte o podobieństwo wejściowych wektorów do pewnych wektorów prototypowych lub ich uogólnień.

Jednorodne miary odległości.

Jako miary podobieństwa może być używana nie tylko miara euklidesowa, często wykorzystywana w sieciach z radialnymi funkcjami bazowymi, ale również jej naturalne uogólnienie do poniższej miary Minkowskiego, jak i inne miary przedstawione w dalszej części.

$$D_M(\mathbf{x}, \mathbf{y}; \alpha) = \left(\sum_{i=1}^{N} |x_i - y_i|^\alpha \right)^{1/\alpha} \tag{2.13}$$

Miara euklidesowa i Manhattan są oczywiście specjalnymi przypadkami miary Minkowskiego dla $\alpha = 2$ i $\alpha = 1$. Można jeszcze bardziej rozbudować miarę Minkowskiego, wprowadzając czynniki skalujące:

$$D_{Mb}(\mathbf{x}, \mathbf{y}; \mathbf{b})^\alpha = \sum_{i}^{N} d(x_i, y_i)^\alpha / b_i \tag{2.14}$$

Funkcja $d(\cdot)$ jest używana do estymacji podobieństwa dla danego wymiaru, najczęściej stosuje się po prostu: $d(x_i, y_i) = |x_i - y_i|$. Dla $\alpha = 2$ wektor $\|\mathbf{x}\| = 1$ znajduje się na sferze jednostkowej, dla większych wartości α sfera przechodzi w gładki hipersześcian, a dla $\alpha < 1$ przyjmuje kształt hipocykloidy (patrz rys. 2.4).

Podobna funkcja była użyta jako jądro (*ang. kernel function*) w modelu Generalized Memory-Based Learning [29]:

$$C_K(\mathbf{x}, \mathbf{x}', \boldsymbol{\nu}) = \left(\sum_{k=1}^{d} (x_k - x'_k)^2 \nu_k^2 \right)^{-q} \tag{2.15}$$

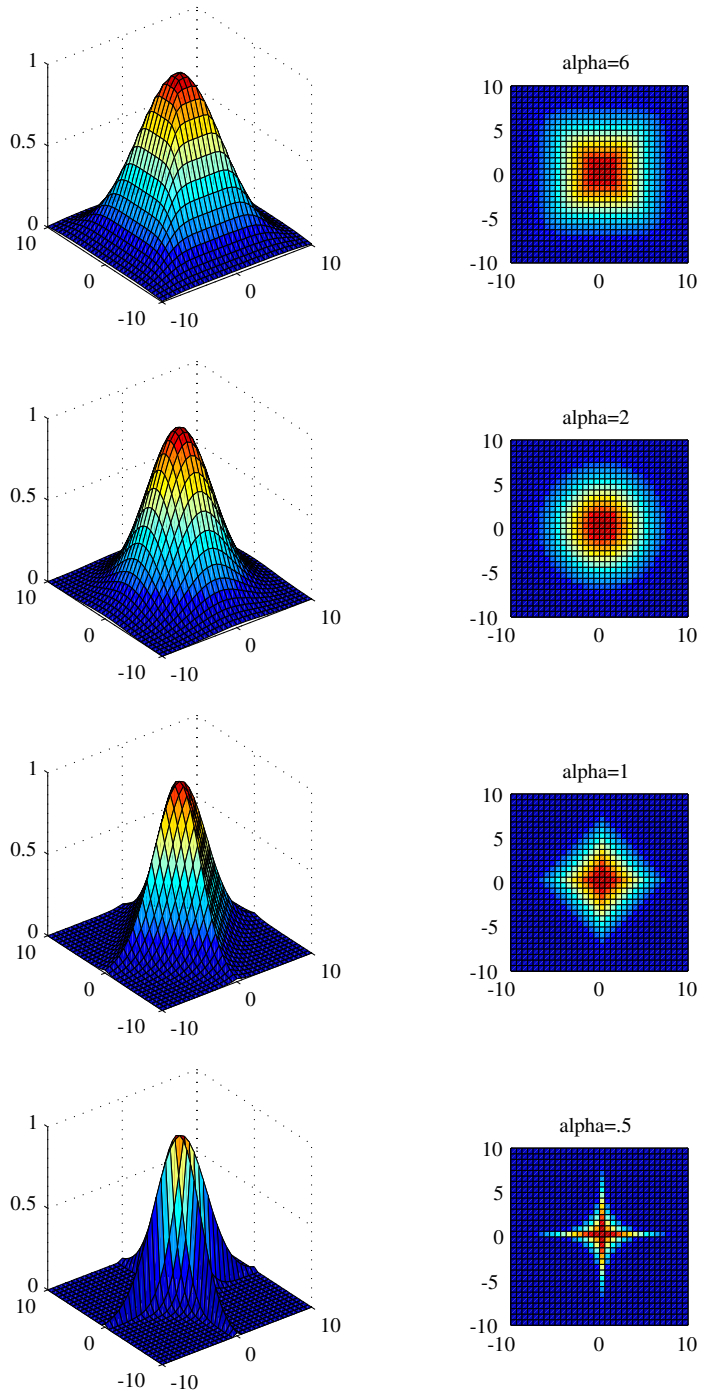
gdzie $q > 0$.

Inną, bardzo dobrą miarą jest miara Mahalanobisa:

$$D_M^2(\mathbf{x}; \mathbf{t}) = \sum_{ij} (x_i - t_i) \Sigma^{-1} (x_i - t_i) \tag{2.16}$$

lub miara odległości o bardziej ogólnej formie kwadratowej z dodatnio określoną macierzą Q ustaloną dla danego problemu:

$$D_Q(\mathbf{x}, \mathbf{y}; Q) = (\mathbf{x} - \mathbf{y})^T Q (\mathbf{x} - \mathbf{y}) \tag{2.17}$$



Rysunek 2.4: Funkcja Gaussa z miarami Minkowskiego o różnych współczynnikach równania 2.13.

Różnego rodzaju czynniki korelacyjne są również pożądane. Na przykład funkcja Camberra:

$$D_{Ca}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} \frac{|x_i - y_i|}{|x_i + y_i|} \tag{2.18}$$

lub Czebyszewa:

$$D_{Ch}(\mathbf{x},\mathbf{y}) = \max_{i=1,\dots,N} |x_i - y_i| \tag{2.19}$$

czy też odległość χ^2:

$$D_{\chi}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} \frac{1}{\mathbf{sum}_i} \left(\frac{x_i}{\mathbf{size}_x} - \frac{y_i}{\mathbf{size}_y} \right)^2 \tag{2.20}$$

gdzie $\mathbf{sum}_i$ jest sumą wszystkich wartości cechy i ze zbioru trenującego, a $\mathbf{size}_x$ i $\mathbf{size}_y$ są sumami wszystkich wartości wektorów x i y.

Korelacyjna miara odległości jest zdefiniowana poprzez:

$$D_{Cd}(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{N}(x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x}_i)^2 \sum_{i=1}^{N}(y_i - \bar{y}_i)^2}} \tag{2.21}$$

gdzie $\bar{x}_i$ i $\bar{y}_i$ są wartościami średnimi cechy i ze zbioru treningowego.

Z kolei funkcję korelacyjną rangową Kendall'a definiuje poniższe wyrażenie:

$$D_{KRC}(\mathbf{x},\mathbf{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{N} \sum_{j=1}^{i-1} \operatorname{sign}(x_i - x_j) \operatorname{sign}(y_i - y_j) \tag{2.22}$$

Wszystkie z powyższych funkcji nadają się na radialne lub do zastąpienia odległości Euklidesowej w wielu funkcjach transferu.

Niejednorodne miary odległości.

Wielowymiarowe miary odległości wcale nie muszą być jednorodne. Można użyć dla cech numerycznych miary Minkowskiego, a dla cech symbolicznych miar statystycznych. W metodach rozumowania opartych na precedensach (*ang. memory-based reasoning*) popularność zyskała miara MVDM (*ang. Modified Value Difference Metric*) [176, 177, 175]. Odległość pomiędzy dwoma N wymiarowymi wektorami $\mathbf{x}$, $\mathbf{y}$ z cechami o wartościach dyskretnych (w tym cechami symbolicznymi). W C-klasowym problemie jest definiowana poprzez prawdopodobieństwa warunkowe jako:

$$D_V^q(x,y) = \sum_{j=1}^{N} \sum_{i=1}^{C} |p(C_i|x_j) - p(C_i|y_j)|^q \tag{2.23}$$

gdzie $p(C_i|x_j)$ jest estymowane przez liczbę $N_i(x_j)$ wystąpień wartości x_j cechy j w wektorach należących do klasy C_i podzielonej przez liczbę $N(x_j)$ wystąpień wartości x_j cechy j w wektorach należących do dowolnej klasy:

$$D_V^q(x,y) = \sum_{j=1}^{N}\sum_{i=1}^{C}\left|\frac{N_i(x_j)}{N(x_j)} - \frac{N_i(y_j)}{N(y_j)}\right|^q \tag{2.24}$$

Różnica wartości dla j-tej cechy jest zdefiniowana jako:

$$d_V^q(x_j,y_j) = \sum_{i}^{C}|(p(C_i|x_j) - p(C_i|y_j))|^q$$

co pozwala policzyć $D_V(\mathbf{x},\mathbf{y})$ przez sumowanie różnic wartości po wszystkich wymiarach. Tak zdefiniowana miara odległości jest zależna od danych (poprzez macierz z liczbą wierszy równą liczbie klas, liczbie kolumn równej liczbie cech). Uogólnienie tej miary na wartości ciągłe, wymaga zbioru funkcji gęstości $p_{ij}(x)$ z $i = 1,\ldots,C$ i $j = 1,\ldots,N$.

Niejednorodna miara HEOM (*ang. Heterogeneous Euclidean-Overlap Metric*) jest pewnym uproszczeniem miary VDM:

$$D_{HEOM}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{j=1}^{N} d_j(x_j,y_j)^2} \tag{2.25}$$

gdzie odległość d_j wyznaczana jest przez:

$$d_j(x_j,y_j) = \begin{cases} 1 & \text{gdy } x_j \text{ lub } y_j \text{ jest nieznany, nieustalony} \\ overlap(x_j,y_j) & \text{gdy atrybut } x_j \text{ jest nominalny} \\ \frac{|x_j-y_j|}{x_j^{max}-x_j^{min}} & \text{wp.p.} \end{cases} \tag{2.26}$$

x_j^{max} i x_j^{min} jest maksymalną i minimalną wartością j-tego atrybutu:

$$x_j^{max} = \max_i x_j^i \qquad x_j^{min} = \min_i x_j^i \tag{2.27}$$

Różnica pomiędzy x_j^{max} i x_j^{min} określa zakres j-tego atrybutu. Funkcja $overlap$ jest zdefiniowana poprzez:

$$overlap(x,y) \quad = \quad \begin{cases} 0 & x = y \\ 1 & x \neq y \end{cases} \tag{2.28}$$

Niejednorodną miarę HVDM (*ang. Heterogeneous Value Difference Metric*) można zdefiniować poprzez:

$$D_{HVDM}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{j=1}^{N}\left(dh_j(x_j,y_j)\right)^2} \tag{2.29}$$

$$\mathrm{dh}_j(x_j, y_j) = \begin{cases} 1 & x \text{ lub } y \text{ jest nieznany} \\ \mathrm{N\_vdm}_j(x_j, y_j) & \text{cecha } j \text{ jest nominalna} \\ \mathrm{N\_dif}_j(x_j, y_j) & \text{cecha } j \text{ jest liniowa} \end{cases} \tag{2.30}$$

a

$$\mathrm{N\_dif}_j(x_j, y_j) = \frac{|x_j - y_j|}{4\sigma_j} \tag{2.31}$$

gdzie σ_j oznacza odchylenie standardowe wartości cechy j. Znormalizowaną odległość VDM można wyznaczyć na kilka sposobów:

$$\begin{aligned} \mathrm{N1\_vdm}(x, y) &= \sum_{i=1}^{C} \left| \frac{N_i(x)}{N(x)} - \frac{N_i(y)}{N(y)} \right| \\ \mathrm{N2\_vdm}(x, y) &= \sqrt{\sum_{i=1}^{C} \left(\frac{N_i(x)}{N(x)} - \frac{N_i(y)}{N(y)} \right)^2} \\ \mathrm{N3\_vdm}(x, y) &= \sqrt{C}\ \ \mathrm{N2\_vdm}(x, y) \end{aligned} \tag{2.32}$$

Dyskretna odmiana miary VDM (*ang. Discrete Value Difference Metric*) może być używana dla ciągłych wartości atrybutów:

$$d_{\mathrm{DVDM}}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N} \mathrm{vdm}_j\big(\mathrm{disc}_j(x_i), \mathrm{disc}_j(y_j)\big)^2 \tag{2.33}$$

gdzie disc jest funkcją dyskretyzacji:

$$\mathrm{disc}_j(x_j) = \begin{cases} \left\lfloor \frac{x - \mathrm{min}_j}{w_j} \right\rfloor + 1 & \text{cecha } j \text{ jest ciągła} \\ x & \text{cecha } j \text{ jest dyskretna} \end{cases} \tag{2.34}$$

w_j są parametrami. Dyskretyzacja umożliwia użycie miary VDM zarówno do nominalnych wartości, jak i do ciągłych. Jeszcze innym sposobem obliczania miary VDM dla cech o ciągłych wartościach jest użycie interpolowanej miary VDM (*ang. Interpolated Value Difference Metric*):

$$d_{\mathrm{IVDM}}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N} \mathrm{ivdm}_j\big(x_j, y_j\big)^2 \tag{2.35}$$

gdzie

$$\mathrm{ivdm}_j(x_j, y_j) = \begin{cases} \mathrm{vdm}_j(x_j, y_j) & \text{cecha } j \text{ jest dyskretna} \\ \sum_{i=1}^{C} \big(p(C_i|x_j) - p(C_i|y_j)\big)^2 & \text{cecha } j \text{ jest ciągła} \end{cases} \tag{2.36}$$

Wyżej użyte prawdopodobieństwa są wyznaczane poprzez interpolację:

$$p(C_i|x_j) = P(C_i|x_j, u) + \frac{x_j - x_{j,u}^{mid}}{x_{j,u+1}^{mid} - x_{j,u}^{mid}} \big(P(C_i|x_j, u+1) - P(C_i|x_j, u)\big) \tag{2.37}$$

gdzie $x_{j,u}^{mid}$ i $x_{j,u+1}^{mid}$ są środkami dwóch następujących zdyskretyzowanych podziałów, spełniających nierówność $x_{j,u}^{mid} \leqslant x_j \leqslant x_{j,u+1}^{mid}$. $P_{j,u,c}$ jest prawdopodobieństwem zdyskretyzowanego podziału u, zdefiniowanego w jego środku. Wartości podziałów u są wyznaczane przez funkcje $\mathtt{disc}_j$: $u = \mathtt{disc}_j(x_j)$.

Miary typu VDM mogą być stosowane w problemach, w których korzysta się z metod gradientowych. W pełni numeryczne wektory wejściowe uzyskuje się, używając ciągłych wymiarów, które zastępują wartości symboliczne i dyskretne poprzez prawdopodobieństwa $p(C_i|x_j)$.

Jak widać możliwości doboru funkcji odległości są całkiem bogate, choć w praktyce rzadko się spotyka odstępstwa od odległości Euklidesowej. Również i sposób, w jaki oddziaływują funkcje odległości z daną metodą uczenia może być dalece inny. Niektóre miary mogą być wręcz równoważne pewnym przekształceniom samych danych wejściowych jeszcze przed procesem uczenia, tym samym miary takie pełnią raczej dość statyczną rolę w procesie uczenia. Trzeba tu zaznaczyć, że głównym celem przekształcenia danych wejściowych powinno być dokonanie takiej transformacji danych, aby model adaptacyjny mógł z nich wyekstrahować jak najwięcej informacji i uzyskać możliwie maksymalną generalizację. Z kolei inne miary nie mogą być zastąpione poprzez transformacje danych przed uczeniem, wtedy też ich charakter podczas procesu uczenia może być dynamiczny poprzez możliwość adaptacji parametrów takiej miary.

2.2.2. Funkcje aktywacji powstające jako kombinacje iloczynu skalarnego i miar podobieństwa

By polepszyć reprezentację bardziej złożonych rejonów decyzyjnych, funkcje transferu wymagają bardziej wyrafinowanych funkcji aktywacji. Dobrym przykładem takiej funkcji aktywacji może być funkcja zaproponowana przez Ridellę *i. in.* [153]:

$$A_R(\mathbf{x}; \boldsymbol{w}) = w_0 + \sum_{i=1}^{N} w_i x_i + w_{N+1} \sum_{i=1}^{N} x_i^2 \tag{2.38}$$

Inną bardzo ciekawą kombinację zaproponował Dorffner [36], tworząc stożkowe funkcje transferu:

$$A_C(\mathbf{x}; \boldsymbol{w}, \mathbf{t}, \omega) = I(\mathbf{x} - \mathbf{t}; \boldsymbol{w}) + \omega D(\mathbf{x} - \mathbf{t}) \tag{2.39}$$

Funkcje transferu C_{GL1} i C_{GL2} opisane wzorami (2.90 i 2.91) używają jeszcze innych kombinacji komponując równie ciekawe funkcje aktywacji:

$$A_{GL1} = \boldsymbol{w}^T\mathbf{x} + \alpha\|\mathbf{x} - \mathbf{t}\| \tag{2.40}$$

$$A_{GL2} = \alpha(\boldsymbol{w}^T\mathbf{x})^2 + \beta\|\mathbf{x} - \mathbf{t}\|^2 \tag{2.41}$$

Powyższe funkcje aktywacji generują jako wartość skalar. Bicentralne funkcje transferu (dokładnie opisane z podrozdziale 2.4.6) korzystają z wektora aktywacji. Co

więcej, bicentralne funkcje transferu korzystają z dwóch wektorów aktywacji lewej i prawej: $A_i = \{A_i^+, A_i^-\}$ i $\mathbf{A} = [A_1, \ldots, A_n]$. Poniżej prezentowane są różne funkcje aktywacji dla różnych funkcji bicentralnych:

$$\begin{aligned}
\text{Bi} \qquad & A1_i^{\pm} = s_i(x_i - t_i \pm b_i), && (2.42)\\
\text{Bi2s} \qquad & A2_i^{\pm} = s_i^{\pm}(x_i - t_i \pm b_i), && (2.43)\\
\text{BiR} \qquad & A3_i^{\pm} = s_i(x_i + r_i x_{i+1} - t_i \pm b_i), && (2.44)\\
\text{BiR2s} \qquad & A4_i^{\pm} = s_i^{\pm}(x_i + r_i x_{i+1} - t_i \pm b_i) && (2.45)
\end{aligned}$$

Przydatność takich funkcji aktywacji okaże się oczywista przy analizie podrozdziału 2.4.6).

2.3. Funkcje wyjścia

Najprostszym przykładem funkcji wyjścia jest oczywiście funkcja tożsamościowa. Pomimo swej prostoty (a może raczej dzięki swej prostocie!) często jest używana w warstwie wejściowej jak i wyjściowej różnych sieci neuronowych. Poza tym używana jest również w sieciach liniowych i warstwie ukrytej sieci RBF, gdzie w połączeniu z miarą odległości tworzy sferyczną funkcję transferu $||\mathbf{x} - \mathbf{t}||$ (warstwa wejściowa i nierzadko wyjścia sieci RBF wykorzystują również funkcję tożsamościową).

Ponieważ zazwyczaj funkcje aktywacji nie są ograniczone, funkcje wyjścia używane są do ograniczania ostatecznych wartości sieci neuronowej. Na trzy główne typy funkcji wyjścia składają się:

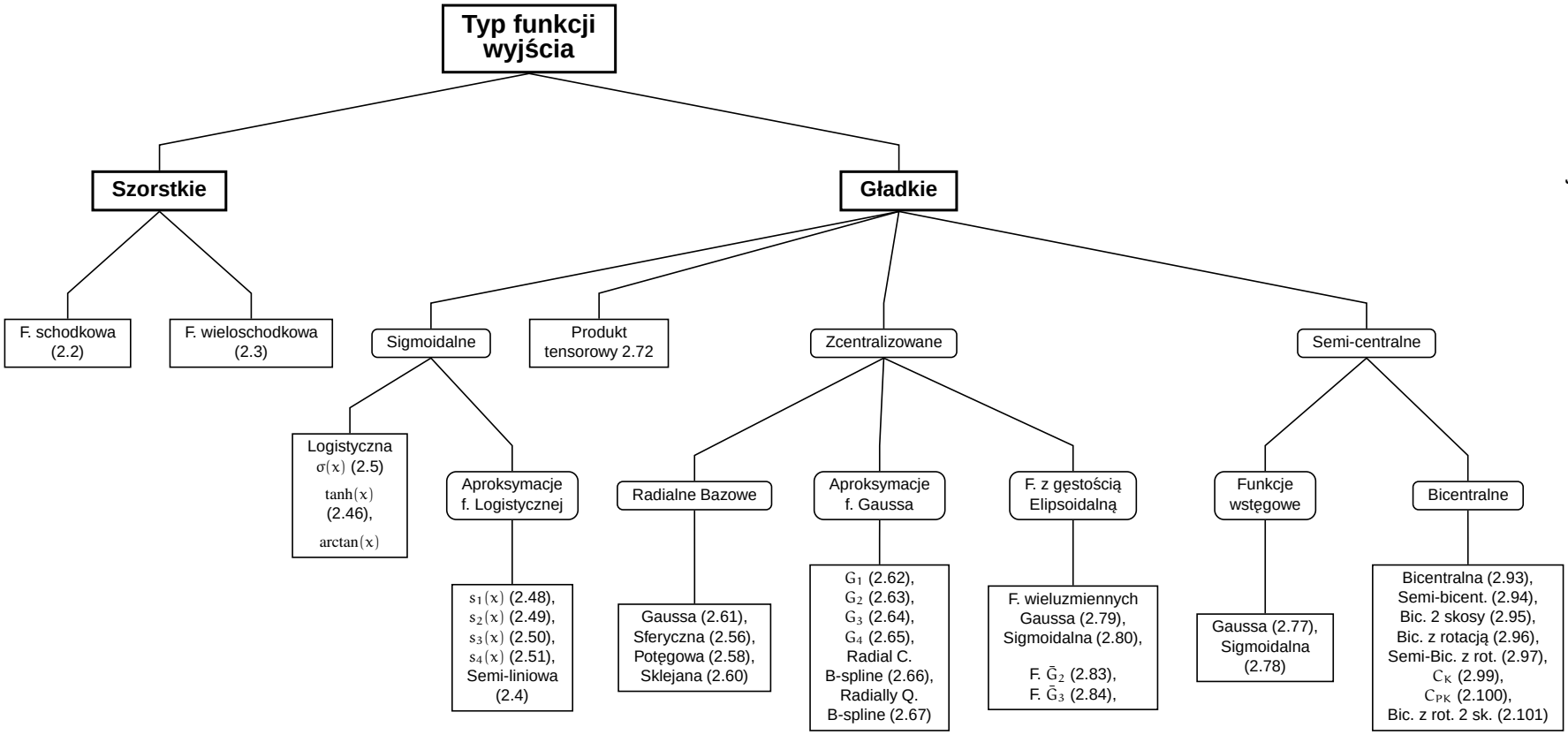
- Funkcje sigmoidalne.
- Funkcje zlokalizowane wokół pewnego centrum.
- Semi-centralne funkcje, które mogą być oparte na kilku centrach, bądź trudnych do określenia centrach.

Na rysunku 2.5 przedstawiona została taksonomia funkcji wyjścia.

2.3.1. Funkcje sigmoidalne

Sigmoidalne funkcje wyjścia, czyli funkcje o kształcie *S* są nie tylko naturalne ze statystycznego punktu widzenia, lecz również w bardzo naturalny sposób umożliwiają ograniczenie nieograniczonych wartości pochodzących z funkcji aktywacji. Funkcje sigmoidalne są funkcjami nielokalnymi — są niezerowe na nieskończonym obszarze.

Ważną własnością jest także gładkość funkcji sigmoidalnych (i łatwość jej regulowania), co jest ważne dla gradientowych metod uczenia. Dla funkcji logistycznej (2.5)



Rysunek 2.5: Taksonomia funkcji wyjścia.

pochodna jest łatwa do wyznaczenia i jest ciągła $\sigma(I)' = \sigma(I)(1-\sigma(I))$. Funkcja logistyczna może być zastąpiona funkcją błędu (erf), funkcją arcus tangens lub też funkcją tangensa hiperbolicznego:

$$\tanh(I/s) = \frac{1-e^{-I/s}}{1+e^{-I/s}} \tag{2.46}$$

$$\tanh'(I/s) = \operatorname{sech}^2(I/s)/s = \frac{4}{s(e^{-I/s}+e^{+I/s})^2} \tag{2.47}$$

Ponieważ obliczanie funkcji eksponencjalnej jest istotnie wolniejsze od obliczania podstawowych operacji arytmetycznych, możliwe jest wykorzystanie innych funkcji o kształcie sigmoidalnym tak, aby przyśpieszyć obliczenia:

$$s_1(I;s) = \Theta(I)\frac{I}{I+s} - \Theta(-I)\frac{I}{I-s} = I\frac{\operatorname{sign}(I)I-s}{I^2-s^2} \tag{2.48}$$

$$s_2(I;s) = \frac{sI}{1+\sqrt{1+s^2I^2}} = \frac{sI}{1+q} \tag{2.49}$$

$$s_3(I;s) = \frac{sI}{1+|sI|} \tag{2.50}$$

$$s_4(I;s) = \frac{sI}{\sqrt{1+s^2I^2}} \tag{2.51}$$

gdzie $\Theta(I)$ jest funkcją schodkową, a $q = \sqrt{1+s^2I^2}$. Pochodne powyższych funkcji daje się łatwo wyznaczyć i szybko obliczyć:

$$s_1'(I;s) = \frac{s}{(I+s)^2}\Theta(I) + \frac{s}{(I-s)^2}\Theta(-I) = \frac{s}{(I+\operatorname{sign}(I)s)^2} \tag{2.52}$$

$$s_2'(I;s) = \frac{s}{q(1+q)} \tag{2.53}$$

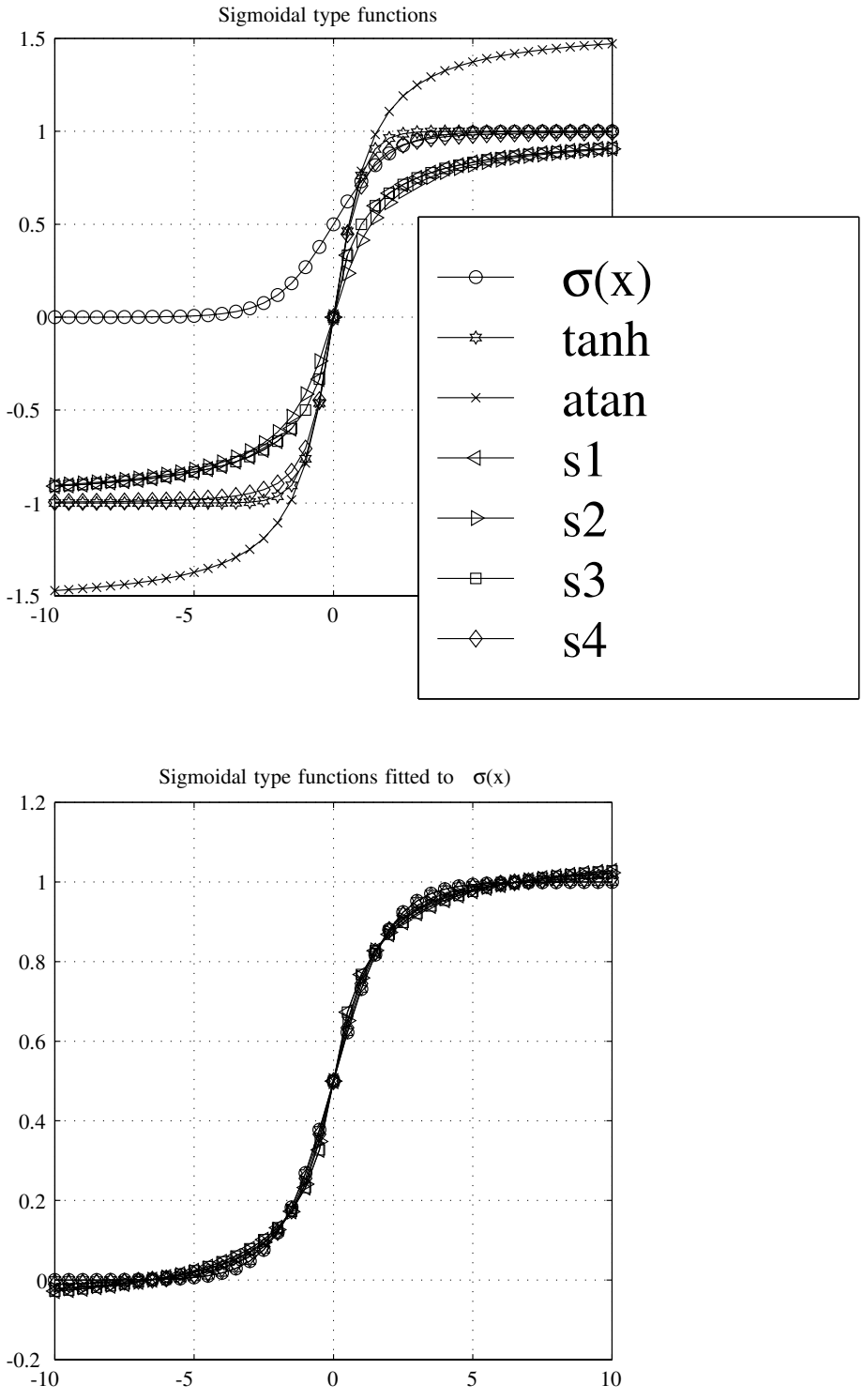
$$s_3'(I;s) = -\operatorname{sign}(I)\frac{s^2I}{(1+|sI|)^2} + \frac{s}{1+|sI|} \tag{2.54}$$

$$s_4'(I;s) = -\frac{s^3I^2}{(1+x^2)^{3/2}} + \frac{s}{\sqrt{1+x^2}} \tag{2.55}$$

Kształty tych funkcji[1] można porównać na rysunku 2.6. Funkcja sigmoidalna i tangens hiperboliczny są trudne do rozróżnienia, natomiast arcus tangens i funkcje s_1, s_2 osiągają nasycenie wolniej dla większych wartości aktywacji. Wszystkie funkcje wygładzają jednak podobnie i dlatego można zaproponować korzystanie z funkcji s_1 lub s_2 aby zaoszczędzić na czasie obliczeń – w praktyce oznacza to oszczędności 2-3 krotne. Innym sposobem zaoszczędzenia czasu na liczeniu funkcji eksponencjalnej jest skorzystanie z pomysłu zamieszczonego w pracy [159], w której funkcja eksponencjalna jest wyznaczana jeszcze szybciej (wykorzystuje się reprezentacje bitową), ale z ok. 2% błędem, co czasem nie jest istotne.

Sieci neuronowe oparte na funkcjach sigmoidalnych mają całkiem dobre właściwości matematyczne. Jedną z nich jest fakt, iż sieć korzystająca z takich funkcji, składająca

[1] Wszystkie te funkcje zostały przeskalowane liniowo tak aby ich wartości znajdowały się w przedziale od -1 do 1, skosy poszczególnych funkcji zostały tak dobrane, aby uzyskać jak największe dopasowanie tych funkcji do funkcji sigmoidalnej.



Rysunek 2.6: Porównanie sigmoidalnych funkcji transferu.

się z jednej warstwy neuronów, jest uniwersalnym aproksymatorem [33, 89]. Nie jest to wielką niespodzianką, ponieważ wiele funkcji może być użytych jako funkcje transferu dla uniwersalnego aproksymatora.

Interesujące jest to, że udało się ocenić zbieżność estymacji przy użyciu funkcji sigmoidalnych. Dla sieci o jednej warstwie z n neuronami i przy pewnych (nie bardzo istotnych) założeniach o aproksymowanej funkcji, współczynnik zbieżności zmienia się z $O(n^{-\frac{1}{2}})$, tj. nie zależy to od rozmiaru przestrzeni wejściowej [6, 118, 119]. Dla funkcji wielomianowych współczynnik zależy od rozmiaru przestrzeni wejściowej d i wynosi $O(n^{-\frac{1}{2d}})$, co dla wielowymiarowych przestrzeni prowadzi do zbyt wolnej zbieżności. Z tego też powodu należy być sceptycznym w wyborze wielomianowych funkcji ortogonalnych na funkcje transferu [152, 26]. Stosowanie niewielomianowych funkcji, w tym funkcji periodycznych czy funkcji lokalnych sprawia, że współczynnik zbieżności nie zależy od rozmiaru przestrzeni wejściowej problemu [90].

2.3.2. Funkcje zlokalizowane wokół jednego centrum

Inną silną klasę funkcji stanowią **radialne funkcje bazowe**, wywodzące się głównie z teorii aproksymacji [151, 52, 60], a później także z metod rozpoznawania wzorców, choć występowały tam pod inna nazwą, jako funkcje potencjałowe [70]. Bardzo dobrym wprowadzeniem w świat radialnych funkcji bazowych, jak i sieci pod kątem teorii regularyzacji, jest praca napisana przez Poggio i Girosiego [150]. Dużo ciekawych informacji na temat funkcji RBF można znaleźć w [85, 19, 36, 122, 123, 124, 8, 145, 10].

Radialne funkcje bazowe zawsze wykorzystują jako funkcje aktywacji odległość od centrum funkcji bazowej $r = \|\mathbf{x} - \mathbf{t}\|/b^2$. W tym rozdziale opisane zostaną radialne funkcje wyjścia oparte o radialną aktywację: $o(r) = o(I)$. Niektóre z radialnych funkcji wyjścia są nielokalne, a inne są lokalne. Najprostszą funkcję radialną uzyskuje się poprzez kombinacje radialnej aktywacji z funkcją tożsamościową, jako funkcją wyjścia, tworząc w ten sposób funkcję sferyczną (patrz rys. 2.7). Funkcja ta nie jest lokalna i określona jest poprzez:

$$h(r) = r = \|\mathbf{x} - \mathbf{t}\|/b^2 \tag{2.56}$$

Allison [2] proponuje użycie prostej funkcji potęgowej (*ang. simple multiquadratic*):

$$s_m(I;\Delta) = \sqrt{I^2 + \Delta^2}; \qquad s'_m(I;\Delta) = \frac{I}{s_m(I;\Delta)} \tag{2.57}$$

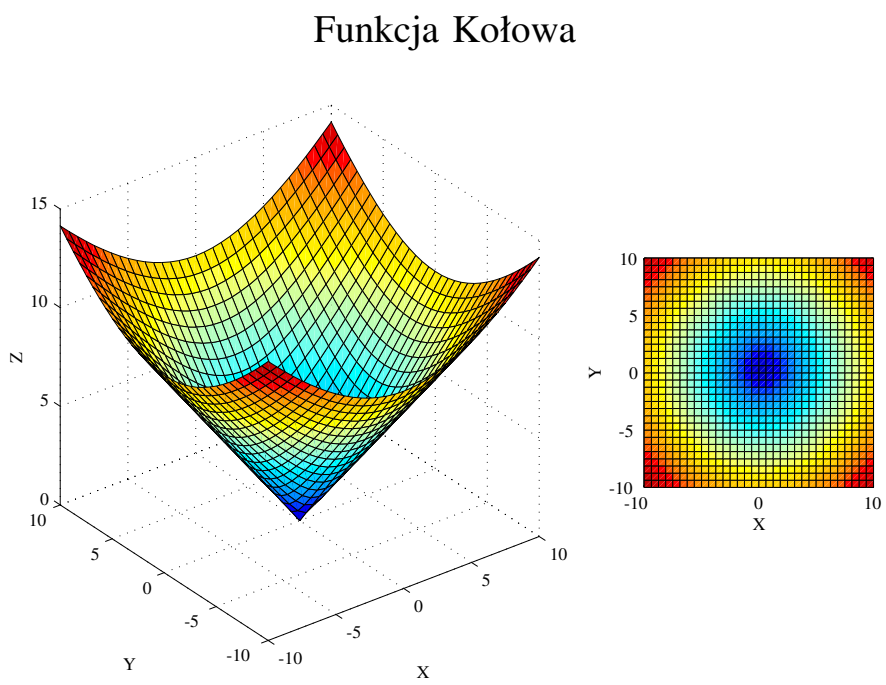
gdzie Δ określa gładkość funkcji.

Innymi przykładami funkcji radialnych jest ogólniejsza wersja funkcji potęgowej (*ang. general multiquadratics*) i funkcja sklejana (rys. 2.10, 2.8, 2.9, 2.7):

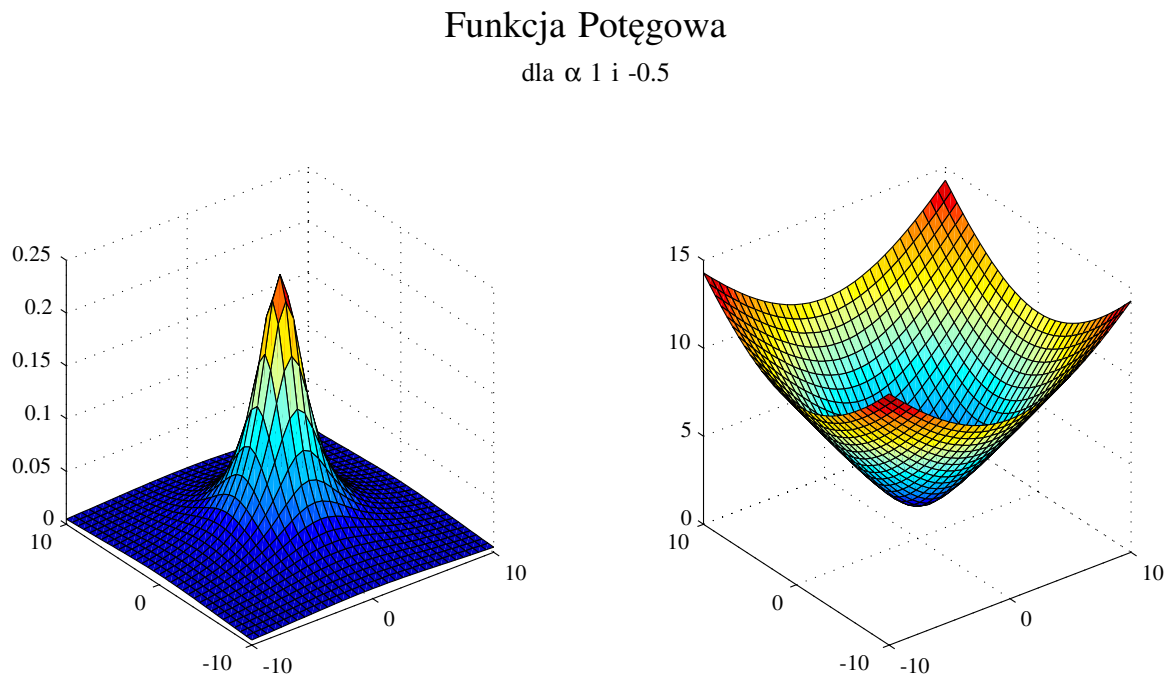
$$h_1(\mathbf{x};\mathbf{t},b,\alpha) = (b^2 + \|\mathbf{x} - \mathbf{t}\|^2)^{-\alpha}, \quad \alpha > 0 \tag{2.58}$$

$$h_2(\mathbf{x};\mathbf{t},b,\beta) = (b^2 + \|\mathbf{x} - \mathbf{t}\|^2)^{\beta}, \quad 0 < \beta < 1 \tag{2.59}$$

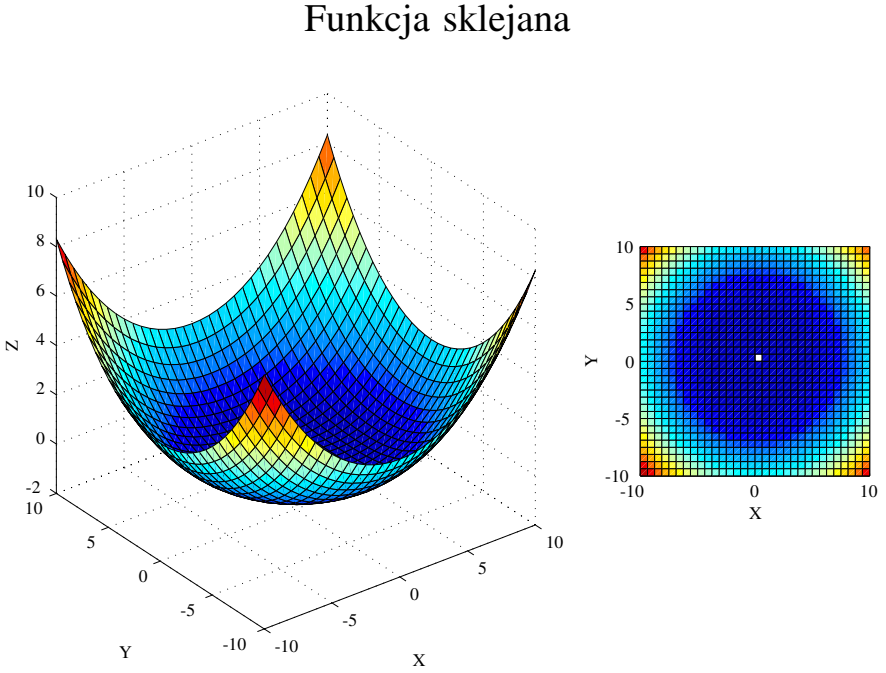
$$h_3(\mathbf{x};\mathbf{t},b) = (b\|\mathbf{x} - \mathbf{t}\|)^2 \ln(b\|\mathbf{x} - \mathbf{t}\|) \tag{2.60}$$



Rysunek 2.7: Funkcja sferyczna (2.56).



Rysunek 2.8: Funkcja potęgowa h_1 i h_2 (2.58).



Rysunek 2.9: Funkcja sklejana h_3 (2.60).

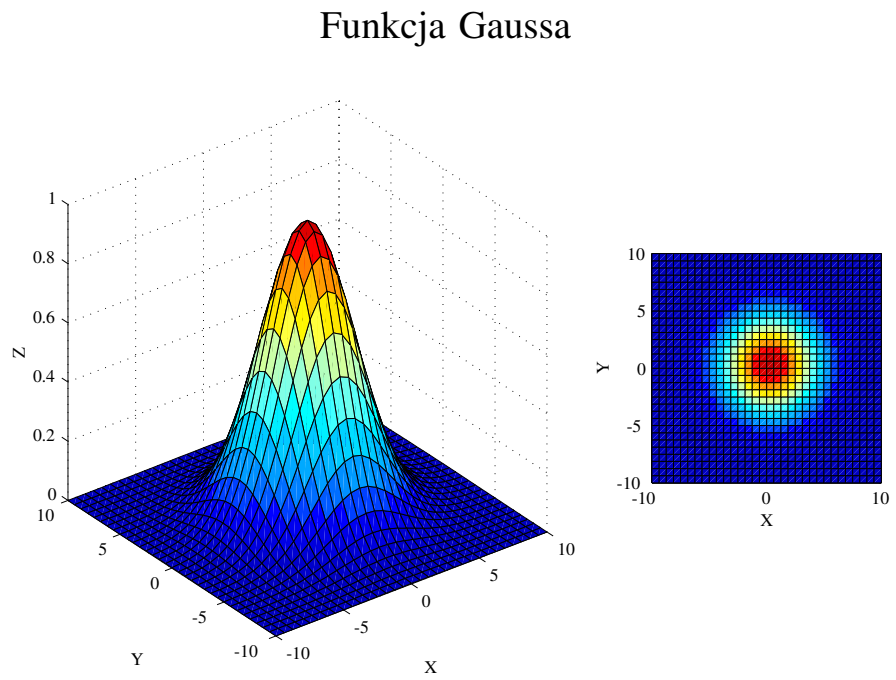
Istnieją różne typy lokalnych radialnych funkcji bazowych. Niewątpliwie najczęściej wykorzystywaną spośród nich jest funkcja gaussowska (patrz rys. 2.10). Funkcja ta najczęściej występuje w kombinacji z odległością euklidesową, choć może być używaną równie dobrze z każdą inna miarą, która może być zapisana jako suma niezależnych komponentów, dzięki czemu funkcja gaussowska jest separowalna[2]. Funkcja gaussowska jest zdefiniowana poniżej:

$$G(r, b) = e^{-r^2/b^2} \tag{2.61}$$

Większość funkcji bicentralnych również posiada własność separowalności (por. podrozdział 2.4.6).

Choć moc przetwarzania sieci z nielokalnymi neuronami nie jest bardzo uzależniona od wyboru niewielomianowej funkcji, to w przypadku korzystania z funkcji lokalnych jest inaczej. Funkcja gaussowska $e^{-D(x)^2}$ jest prawdopodobnie najprostszą postacią funkcji lokalnej, chociaż nie najtańszej w sensie obliczeniowym. Funkcja logistyczna, tanh lub funkcje drugiego lub czwartego stopnia ze zlokalizowaną funkcją aktywacji

[2]Dzięki separowalności można w dowolny sposób wybierać dowolną podprzestrzeń przestrzeni wejściowej i poddawać ją przeróżnym analizom. Jest to niemal koniecznym warunkiem w niektórych zastosowaniach, dla przykładu w wyciąganiu reguł logicznych z sieci typu RBF czy FSM, jak i w zastosowaniu do baz danych, w których napotyka się na wartości brakujące.



Rysunek 2.10: Funkcja gaussowska (2.61).

aproksymują kształt funkcji Gaussa:

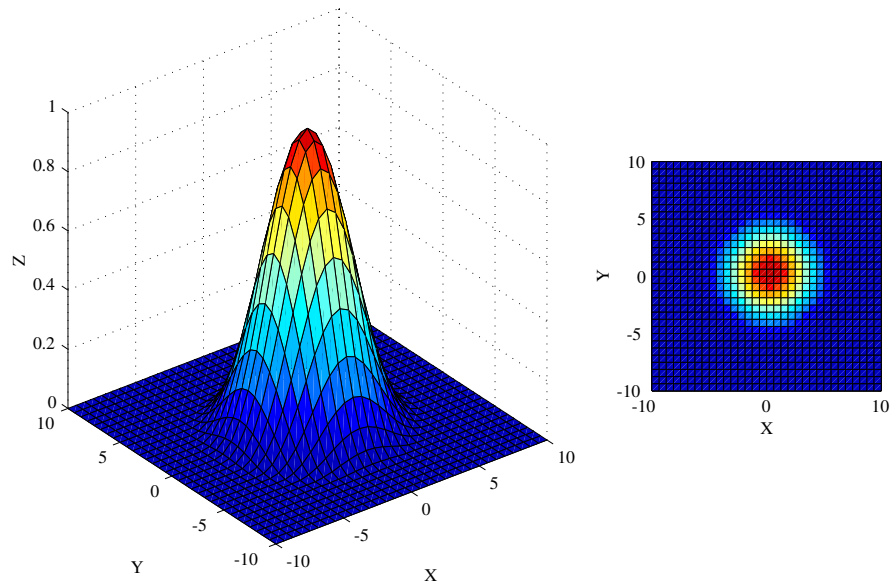
$$G_1(r) = 2 - 2\sigma(r^2) \tag{2.62}$$
$$G_2(r) = 1 - \tanh(r^2) \tag{2.63}$$
$$G_3(r) = \frac{1}{1+r^2}; \quad G_3'(r) = -2rG_3^2(r); \tag{2.64}$$
$$G_4(r) = \frac{1}{1+r^4}; \quad G_4'(r) = -4r^3G_4^2(r) \tag{2.65}$$

W [156] została zaproponowana kołowa funkcja sklejana trzeciego stopnia (*ang. radial cubic B-spline function*). Jest ona zdefiniowana poprzez:

$$\mathrm{RCBSpline}(r) = \frac{1}{4h^2}\begin{cases} h^3 + 3h^2(h-r) + 3h(h-r)^2 + 3(h-r)^3 & r \leqslant h \\ (2h-r)^3 & h < r \leqslant 2h \\ 0 & 2h < r \end{cases} \tag{2.66}$$

gdzie $r = \|\mathbf{x} - \mathbf{t}_i\|^2$, a $\mathbf{t}_i$ to centrum. Rysunek 2.11 pokazuje przykład takiej funkcji.

Saranli i Baykal [156] opisują też kołową funkcję sklejaną stopnia czwartego (*ang.*



Rysunek 2.11: Kołowa funkcja sklejana trzeciego stopnia (2.66).

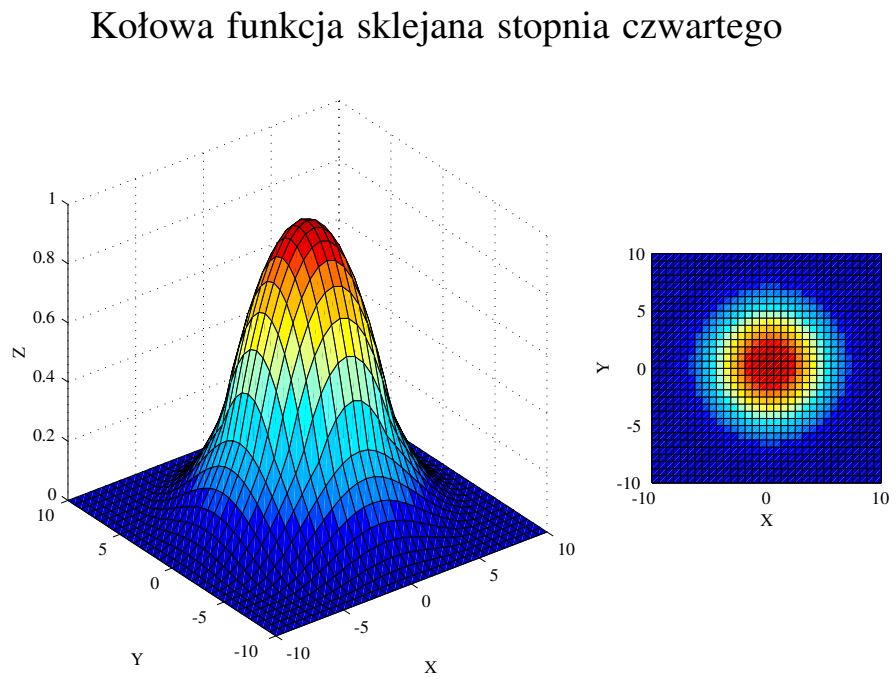
radially quadratic B-spline function) zdefiniowaną przez poniższe równanie:

$$\mathrm{RQBSpline}(r) = \frac{1}{2h^2}\begin{cases} -2r^2 + 3h^2 & r \leqslant h \\ (2h - r)^2 & h < r \leqslant 2h \\ 0 & 2h < r \end{cases} \tag{2.67}$$

gdzie $r = \|\mathbf{x} - \mathbf{t}_i\|$ i $\mathbf{t}_i$ jest i-tym centrum (patrz rys. 2.12, w artykule Saranli i Baykala [156] były błędy w definicjach powyższych dwóch funkcji, tutaj są one zaprezentowane już w poprawnej formie).

Porównanie różnych funkcji zbliżonych kształtem do funkcji gaussowskiej przedstawiono na rys. 2.13.

Szybkość zbieżności radialnych funkcji bazowych ze stałym rozmyciem została opisana przez Niyogiego i Girosiego [141]. Ponieważ prawdziwa funkcja jest nieznana, błąd może być mierzony jedynie względem najlepszej możliwej estymacji tej funkcji, zwanej funkcją regresji $f_0(X)$. Różnice pomiędzy funkcją regresji i funkcją realizowaną poprzez sieć złożoną z radialnych funkcji bazowych z n neuronami, o d wymiarowej przestrzeni wejściowej i k wzorcach uczących, estymuje z poziomem



Rysunek 2.12: Kołowa funkcja sklejana czwartego stopnia (2.67).

zaufania $1-\delta$ poniższe wyrażenie:

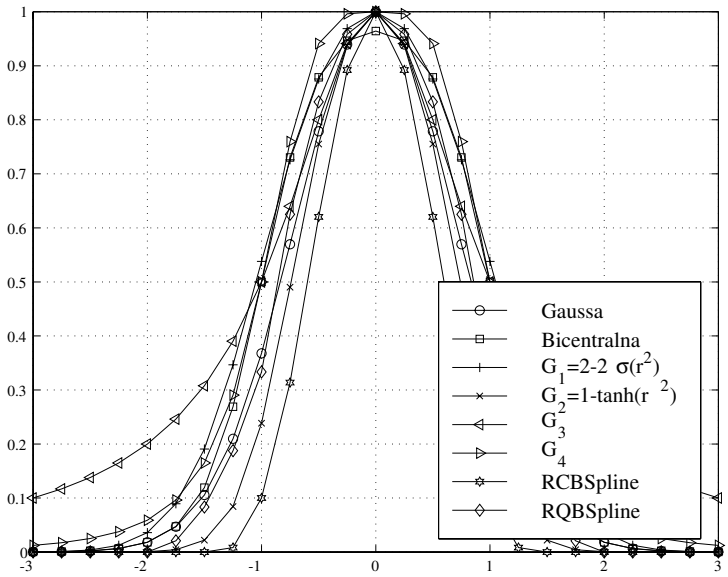
$$\begin{aligned} E\left[(f_0(\boldsymbol{x})-F_{n,k}(\boldsymbol{x}))^2\right] &= \int_{\boldsymbol{x}} d\boldsymbol{x}\, P(\boldsymbol{x})\,(f_0(\boldsymbol{x})-F_{n,k}(\boldsymbol{x}))^2 \\ &\leqslant O\left(\frac{1}{n}\right)+O\left(\sqrt{\frac{nd\ln(nk)-\ln\delta}{k}}\right) \end{aligned} \tag{2.68}$$

Pierwsza część wynika z teorii aproksymacji, $O(1/n)$, natomiast druga, $O\left(((nd\ln(nk)-\ln\delta)/k)^{1/2}\right)$, wynika ze statystyki. Błąd zanika tylko gdy złożoność sieci (sieć opisana jest poprzez n neuronów) jest nieporównanie mała względem liczby wzorców uczących k. Dla każdego wybranego zbioru wzorców istnieje pewna optymalna liczba neuronów, która minimalizuje błąd generalizacji.

2.3.3. Funkcje semi-centralne

Semi-centralne funkcje wyjścia wykorzystują funkcję aktywacji w postaci wektorowej. Dla przykładu wstęgowa funkcja Gaussa przyjmuje postać:

$$\bar{G}(\boldsymbol{r},\boldsymbol{b},\boldsymbol{\nu})=\sum_{i=1}^{N}\nu_i e^{-r_i^2/b_i^2};\quad r_i=(x_i-t_i) \tag{2.69}$$



Rysunek 2.13: Porównanie lokalnych funkcji wyjścia (patrz równania (2.61, 2.93, 2.62–2.66)).

Natomiast rodzinę funkcji bicentralnych można określić wzorem:

$$\mathrm{Bi}(\mathbf{r};\mathbf{b},\mathbf{s}) = \prod_{i=1}^{N} \sigma(I^{+})(1-\sigma(I^{-})) \tag{2.70}$$

gdzie I^{+} i I^{-} są zdefiniowane przez równania (2.42–2.45), tj. wykorzystują dwa wektory aktywacji. Więcej informacji można znaleźć w podrozdziale 2.4.6.

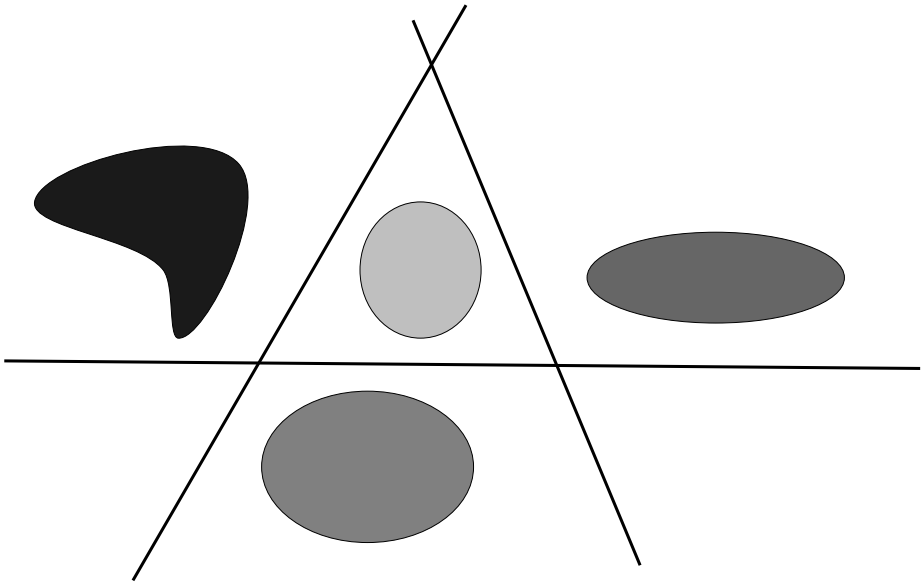
2.4. Funkcje transferu

W tym podrozdziale omawiane są różne kombinacje funkcji aktywacji z funkcjami wyjścia, tworzące przeróżne funkcje transferu. Funkcje transferu zostały podzielone na lokalne i nielokalne. Wyodrębnione zostały też grupy funkcji o hiperelipsoidalnych konturach gęstości funkcji jak i te, które mogą być lokalne i nielokalne, w zależności od doboru parametrów wolnych funkcji, funkcje te będą nazywane *funkcjami uniwersalnymi*. Ostatnią grupę funkcji stanowi rodzina uniwersalnych funkcji bicentralnych.

2.4.1. Nielokalne funkcje transferu

Nielokalne funkcje transferu używane w sieciach neuronowych dzielą wielowymiarową przestrzeń wejściową, dokonując jej podziału na regiony odpowiadające różnym

klasom lub różnym wartościom wektorów wyjściowych. Należy zwrócić uwagę, iż zmiana pojedynczego parametru może powodować istotne zmiany wartości wyjściowej całej sieci dla wszystkich punktów przestrzeni wejściowej. Dlatego też algorytm uczenia musi dokonywać zmian wszystkich parametrów adaptacyjnych w odpowiedniej kolejności. Typowe sigmoidalne funkcje transferu (2.5, 2.46–2.48) oparte o liniową kombinację wejść (2.1), jako aktywację, używane w sieciach MLP do klasyfikacji, jak i aproksymacji, zostały już opisane w podrozdziale 2.3.1.



Rysunek 2.14: Podział na regiony decyzji uformowane przy użyciu funkcji sigmoidalnych z aktywacją zdefiniowaną przez (2.1).

Obszary decyzji jakie powstają przy użyciu takich funkcji w klasyfikacji danych są tworzone przez przecięcia wielowymiarowej przestrzeni wejściowej hiperpłaszczyznami (patrz rysunek 2.14). Taki model *udaje*, że wie wszystko, a to może być niepożądane, głównie w tych miejscach przestrzeni wejściowej, w których nie było żadnych danych uczących. W takich obszarach hiperpłaszczyzny rozciągają się w nieskończoność i dochodzi do arbitralnej klasyfikacji. Funkcje sigmoidalne wygładzają wiele płytkich lokalnych minimów w całkowitej funkcji wyjściowej sieci. Dla problemów klasyfikacyjnych może to być nawet przydatne, lecz ogólnie, w uczeniu odwzorowań (w mapowaniu) może to ograniczać precyzję modelu adaptacyjnego.

Radialne funkcje bazowe (RBF) są wykorzystywane w wielu różnych symulatorach sieci neuronowych, lecz w większości symulatorów jest dostępna tylko funkcja gaussowska, która jest lokalna. Jak już zostało wspomniane, do nielokalnych funkcji bazowych należą funkcje potęgowe (2.58), funkcje sklejane (2.60). Korzystają one z radialnej aktywacji opartej o normę Mikowskiego lub inną miarę odległości (por. rysunki 2.8 i 2.9).

Funkcje te dają gęstości elipsoidalne. Giraud (i in.) [73] użyli liniowej kombinacji

wejść do stworzenia Lorentzowskiej funkcji odpowiedzi (patrz rys. 2.16):

$$L(\boldsymbol{x}; \boldsymbol{w}, \theta) = \frac{1}{1 + I^2(\boldsymbol{x}; \boldsymbol{w}, \theta)} = \frac{1}{1 + \left(\sum_{i=1}^{N} w_i x_i - \theta\right)^2} \tag{2.71}$$

Funkcja Lorentzowska nie ma gęstości elipsoidalnych czy kolistych tak jak radialne funkcje bazowe. Powierzchnia stałej gęstości jest w tym przypadku typu okna funkcji nielokalnej, którego połowa szerokości jest równa $1/\sqrt{\sum_i w_i^2}$. Nielokalne funkcje typu *okna* mogą być uzyskane z wielu lokalnych i semi-lokalnych funkcji opisanych poniżej, gdy tylko produkt poszczególnych składowych tych funkcji nie obejmuje wszystkich wymiarów.

Potęgowy iloczyn tensorowy został wprowadzany do algorytmu MARS przez Friedman [64] (patrz rys. 2.15). Zdefiniowany jest poprzez

$$T(\boldsymbol{x}; \mathbf{t}, \mathbf{s}, \mathcal{I}) = \prod_{i \in \mathcal{I}} [s_i(x_i - t_i)]_+^q \tag{2.72}$$

gdzie $\mathcal{I}$ jest pewnym podzbiorem cech wejściowych, q jest stałą ustaloną przez użytkownika, wektor $\mathbf{t}$ definiuje położenie centrum, a s_i definiuje kierunek i może być równe 1 lub -1, z kolei $[\,\cdot\,]_+$ oznacza, że wartości mniejsze od zera są zastępowane zerami.

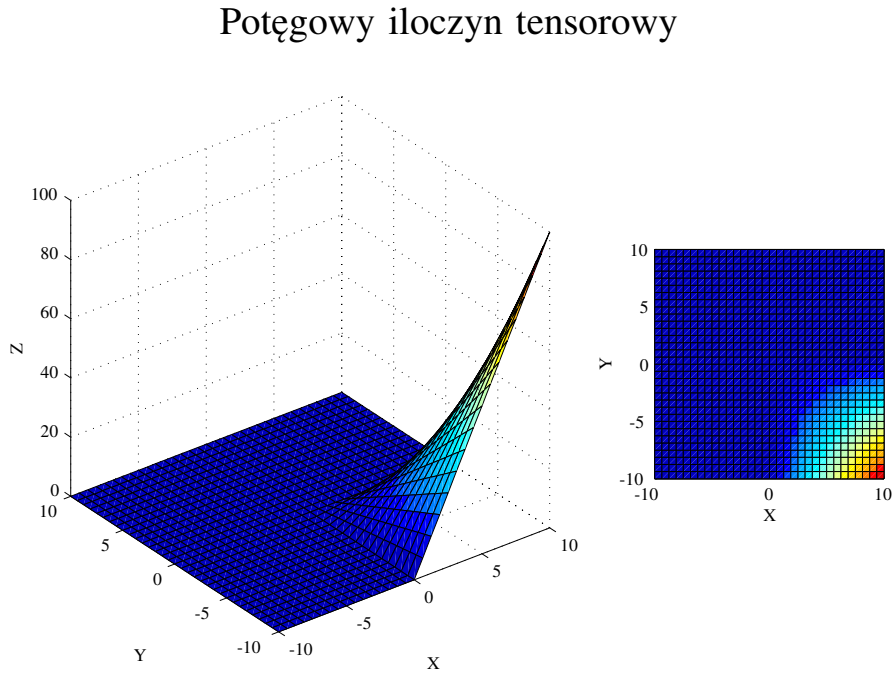
Iloczyn tensorowy w algorytmie MARS został użyty jako komponent do budowy aproksymatora jako jego liniowej kombinacji:

$$MARS(\boldsymbol{x}; \boldsymbol{w}, \mathbf{T}, \mathbf{S}, \mathcal{I}) = \sum_{i=1}^{N} w_i \; T(\boldsymbol{x}; \mathbf{t}_i, \mathbf{s}_i, \mathcal{I}_i) + w_0 \tag{2.73}$$

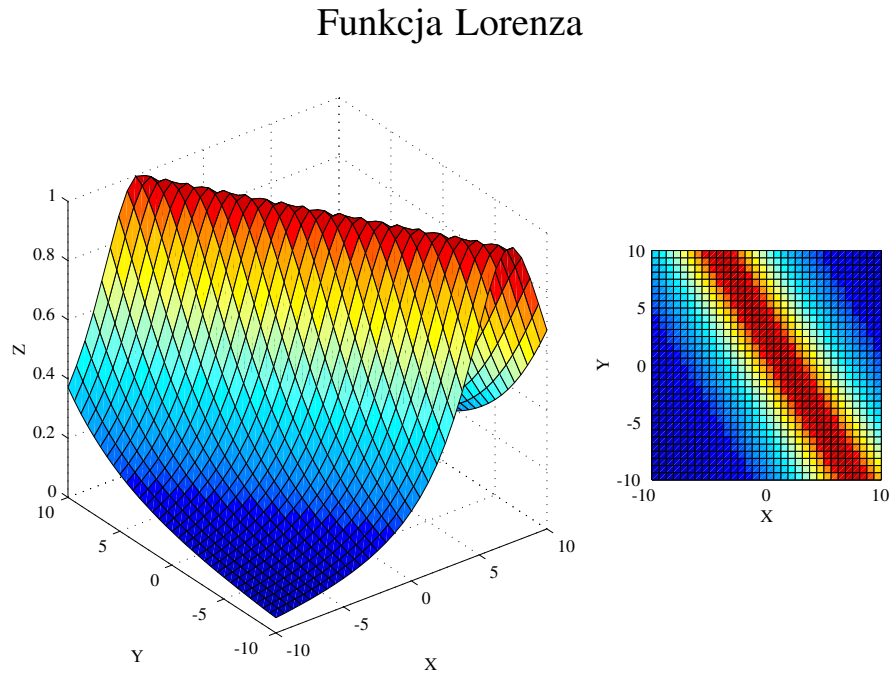
2.4.2. Lokalne i semi-lokalne funkcje transferu

Lokalizowane funkcje transferu używają parametrów, które mają lokalny wpływ na zmiany wartości sieci, czyli zmiany wartości parametrów funkcji. Powodują zmiany wartości całkowitego wyjścia sieci tylko dla wektorów położonych w lokalnym rejonie oddziaływania przestrzeni wejściowej. Były czynione różne próby używania funkcji zlokalizowanych wśród wczesnych modeli sieci neuronowych. Niektóre z nich dotyczyły prac w rozpoznawaniu wzorców [70]. Moody i Darken [135] używali zlokalizowanych neuronów do uczenia odwzorowań rzeczywistych danych i do klasyfikacji, wykorzystując metody samoorganizacji i uczenia z nadzorem. Wybór zlokalizowanych neuronów miał na celu przyśpieszenie procesu uczenia w sieciach wstecznej propagacji. Bottou i Vapnik [15] pokazali siłę metod opartych o zlokalizowane uczenie w ogólności. Z kolei Kadirkamanathan i Niranjan [107] pokazali, że dla konstruktywistycznych metod warunek gładkości dla dodawania nowych neuronów jest spełniony tylko dla mocno zlokalizowanych neuronów.

Najczęściej wykorzystywanym typem aktywacji funkcji RBF jest euklidesowa miara odległości, choć prosto można ją uogólnić, czy zmienić, do dowolnej innej miary



Rysunek 2.15: Funkcja bazowa z potęgowego iloczynu tensorowego.



Rysunek 2.16: Funkcja Lorentzowska (2.71).

$D(\mathbf{x};\mathbf{t})$, gdzie $\mathbf{t}$ jest centrum funkcji bazowej. Funkcja aktywacji minimum osiąga dla centrum: $\mathbf{x} = \mathbf{t}$ i monotonicznie rośnie wraz z oddalaniem się od centrum wektora $\mathbf{x}$. Odległość Hamminga jest często wykorzystywana dla wejść binarnych. Można też wzbogacić miarę aktywacji o czynniki skalujące (por. 2.14), co prowadzi do dodania nowych N parametrów adaptacyjnych.

Najprostszym sposobem na zbudowanie sieci neuronowej z radialnymi funkcjami bazowymi jest zgrupowanie pewnej liczby n funkcji radialnych $G_i(\mathbf{x})$ z uprzednio wyznaczonymi parametrami b i centrami $\mathbf{t}$ i wyznaczenie współczynników w_i. Pozycje centrów mogą być wyznaczone przy użyciu klasteryzacji k-średnich (*ang.* k-*means clastering*) i ustaleniu rozmyć b na dwukrotną wartość najmniejszej odległości pomiędzy funkcjami bazowymi [117]. Wtedy sieć RBF można przedstawić w poniższej postaci:

$$f(\mathbf{x};\mathbf{w},\mathbf{T},\mathbf{b}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x},\mathbf{t}_i,b_i) = \sum_{i=1}^{M} w_i e^{-||\mathbf{x}-\mathbf{t}_i||^2/b_i^2} \tag{2.74}$$

W uogólnionej sieci RBF z regularyzacją również centra funkcji bazowych $\mathbf{t}_i$ podlegają adaptacji [150], pozwalając w ten sposób zredukować liczbę funkcji bazowych adekwatnie do szumu, jaki może się znajdować w danych (to właśnie składa się na regularyzację aproksymowanej funkcji). Szczegółowe informacje o różnych algorytmach uczenia sieci RBF można znaleźć w rozdziale 3.

W N wymiarowej przestrzeni centrum i-tej funkcji bazowej jest opisane przez N współczynników wektora $\mathbf{t}_i$ i jeden parametr b_i, który wyznacza rozmycie funkcji. W prosty sposób można dokonać uogólnienia funkcji gaussowskiej z odległością euklidesową umożliwiając adaptacje rozmycia dla każdego z wymiarów przestrzeni wejściowej niezależnie, co daje w rezultacie $2N$ parametrów adaptacyjnych na jeden neuron.

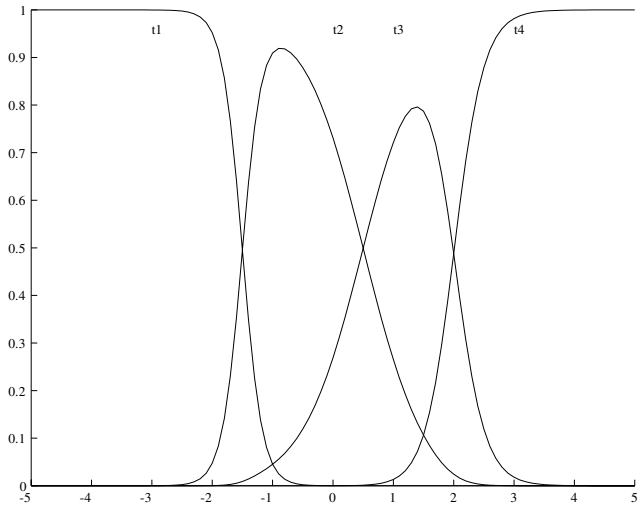
Moody i Darken [135] zaproponowali używanie znormalizowanej wersji funkcji Gaussa:

$$GN_i(\mathbf{x};\mathbf{t}_i,b_i) = \frac{e^{-||\mathbf{x}-\mathbf{t}_i||^2/b_i^2}}{\sum_{j=1}^{M} e^{-||\mathbf{x}-\mathbf{t}_j||^2/b_j^2}} \tag{2.75}$$

Interesującą właściwością tej funkcji jest, że dla każdego wektora $\mathbf{x}$ z przestrzeni wejściowej, suma wartości funkcji w tym punkcie jest równa 1:

$$\sum_i GN_i(\mathbf{x};\mathbf{t}_i,b_i) = 1 \tag{2.76}$$

Dzięki temu, wyjście każdej z tak określonych funkcji można interpretować jako prawdopodobieństwo wyznaczane przez i-ty neuron. Na rysunku 2.17 są pokazane wyjścia (kompletu) czterech znormalizowanych funkcji Gaussa. Bridle [18] zaproponował nazywanie tych funkcji funkcjami *soft-max*, co było motywowane wyżej przedstawionymi właściwościami tych funkcji.



Rysunek 2.17: Znormalizowana funkcja Gaussa — softmax.

2.4.3. Gaussowska i sigmoidalna funkcja wstęgowa

Na wejścia sieci neuronowych nie zawsze są podawane istotne informacje. Mamy wtedy do czynienia ze zbędnymi cechami, czy też cechami o zmniejszonym udziale w procesie adaptacji. Typowe funkcje radialne nie są zdolne do automatycznego wyeliminowania takich cech. Problem ten był rozpatrywany przez Hartmana i Keelera [81, 80] także przez Parka i Sandberga [145]. W przeciwieństwie do wielowymiarowej funkcji gaussowskiej autorzy tworzą funkcje transferu przez kombinacje jednowymiarowych ważonych funkcji gaussowskich (rys. 2.18):

$$G_b(\mathbf{x};\mathbf{t},\mathbf{b},\boldsymbol{\nu}) = \sum_{i=1}^{N} \nu_i e^{-(x_i - t_i)^2 / b_i^2} \tag{2.77}$$

W tym przypadku funkcje aktywacji i wyjścia nie są separowalne. Funkcja wstęgowa ma 3N parametrów adaptacyjnych na jeden neuron. Oryginalna angielska nazwa funkcji *Gaussian bar functions* pochodzi od jej charakteru w wielowymiarowej przestrzeni. Dla przestrzeni wielowymiarowych N wartości poszczególnych wag wstęg ν_i mogą być znacznie mniejsze niż suma wszystkich N wag ν_i wokół $\mathbf{t}$. W warstwie wyjściowej stosuje się funkcje sigmoidalną dla wygładzenia funkcji sieci i zredukowania liczby małych maksimów.

Funkcje wstęgowe pozwalają na eliminację nieistotnych wymiarów wejściowych prowadząc do redukcji wymiarów i to w prostszy sposób niż np. w przypadku wielowymiarowej funkcji gaussowskiej wielowymiarowej. Rozmycia w poszczególnych wymiarach również prowadzą do redukcji wymiarów (por. przykład odwzorowania logistycznego kwadratowego, Moody i Darken [135]). Inna zaleta użycia funkcji wstęgowej płynie z samej istoty istnienia wstęg, które podlegają sumowaniu. Pojedyncze maksimum lub kilka odseparowanych maksimów mogą być opisane przez małą liczbę

funkcji gaussowskich (z $N+1$ parametrami na funkcję) a także przez taką samą liczbę funkcji wstęgowych z niemal trzykrotnie większą liczbą parametrów. Lecz gdy na wejściu znajduje się k klastrów w regularnych odstępach w każdym z N wymiarów i formują w ten sposób k^N klastrów w N wymiarowym hiperześcianie. Taki obraz wejścia wymaga tyle funkcji gaussowskich wielu zmiennych, co klastrów. Natomiast funkcji wstęgowych wystarczy Nk.

Można utworzyć podobną kombinację funkcji wstęgowej sigmoidalnej (*ang. sigmoidal bar function*), korzystając z funkcji sigmoidalnych (patrz rys. 2.19):

$$\sigma_b(\mathbf{x};\mathbf{t},\mathbf{b},\boldsymbol{\nu}) = \sum_{i=1}^{N} \frac{\nu_i}{1+e^{-(x_i-t_i)^2/b_i^2}} \tag{2.78}$$

Ta funkcja, podobnie jak i gaussowska funkcja wstęgowa, daje kontury o kontrastowych gęstościach, których obrócenie nie jest proste, co też jest wadą tych funkcji. Sigmoidalna funkcja wstęgowa nie powinna być używana do reprezentacji klastra danych wokół kilku punktów, ponieważ każdy klaster będzie potrzebował $2N$ funkcji sigmoidalnych, podczas gdy wystarczyłaby jedna funkcja gaussowska. Jednak gdy klastry są rozłożone regularnie na kwadratowej siatce, to k^2 klastrów może być reprezentowanych przez tyle samo funkcji gaussowskich lub jedynie $2k$ sigmoidalnych funkcji wstęgowych, czy k gaussowskich funkcji wstęgowych. Z kolei, aby te same klastry przestrzeni wejściowej opisać przy użyciu hiperpłaszczyzn lub funkcji sigmoidalnych, potrzeba by ich $2k-2$.

2.4.4. Funkcje o gęstościach elipsoidalnych

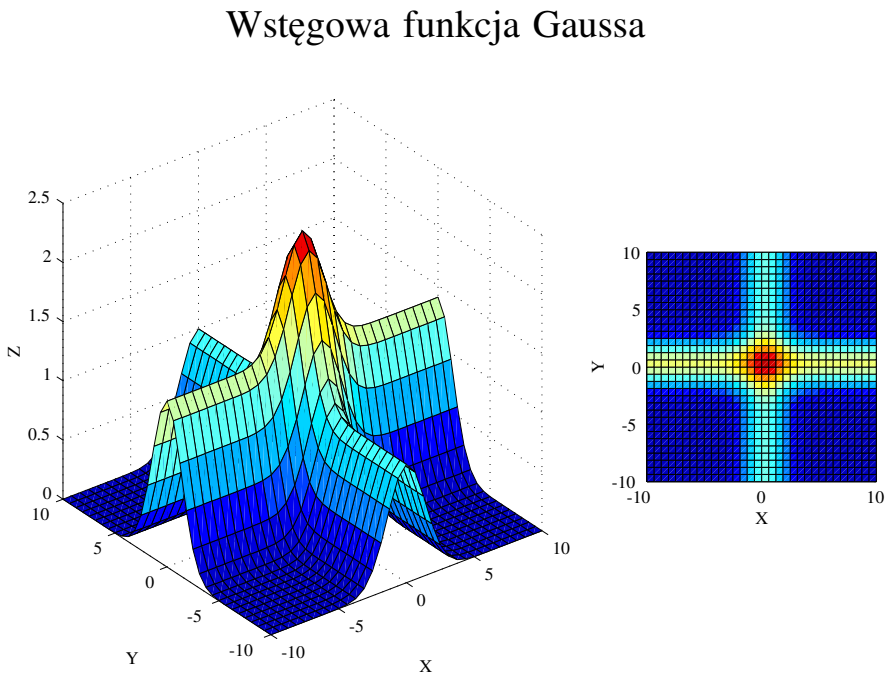
Wielowymiarowa funkcja gaussowska jest przykładem funkcji o gęstości hiperelipsoidalnej (patrz rys. 2.20):

$$G_g(\mathbf{x};\mathbf{t},\mathbf{b}) = e^{-D^2(\mathbf{x};\mathbf{t},\mathbf{b})} = \prod_{i=1}^{N} e^{-(x_i-t_i)^2/b_i^2} \tag{2.79}$$

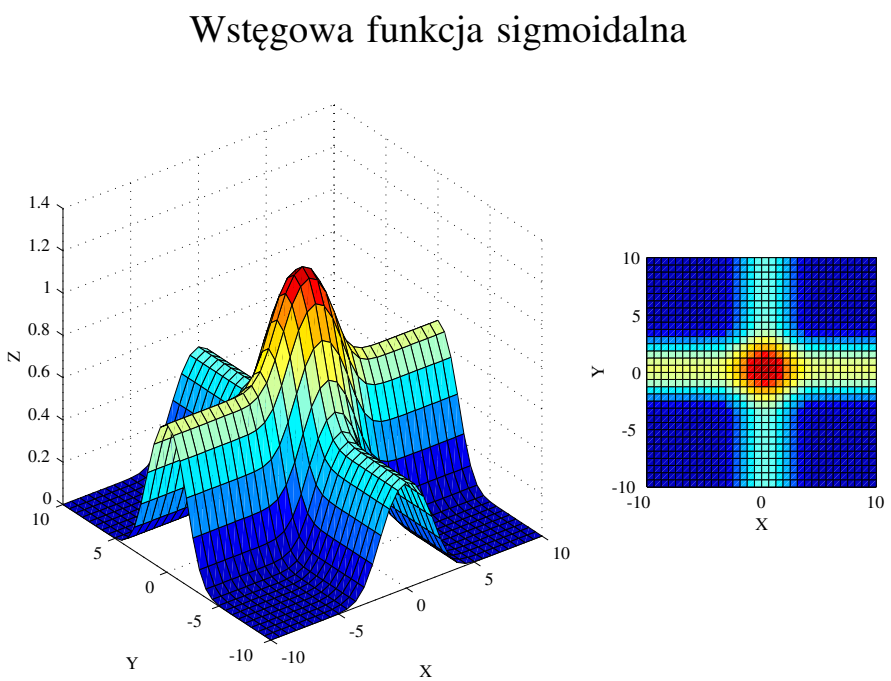
Rozmycia b_i mogą być rozumiane jako współczynniki skalujące w funkcji odległości D^b (2.12). Podobny rezultat uzyskuje się przy użyciu kombinacji funkcji logistycznych (lub innych funkcji sigmoidalnych) z kwadratem funkcji odległości, na przykład (patrz rys. 2.21):

$$\begin{aligned} G_S(\mathbf{x};\mathbf{t},\mathbf{b}) &= 1-\sigma(D^2(\mathbf{x};\mathbf{t},\mathbf{b})) && (2.80)\\ &= \frac{1}{1+\prod_{i=1}^{N} e^{(x_i-t_i)^2/b_i^2}} = \frac{1}{1+e^{D^b(\mathbf{x};\mathbf{t},\mathbf{b})}} && (2.81)\end{aligned}$$

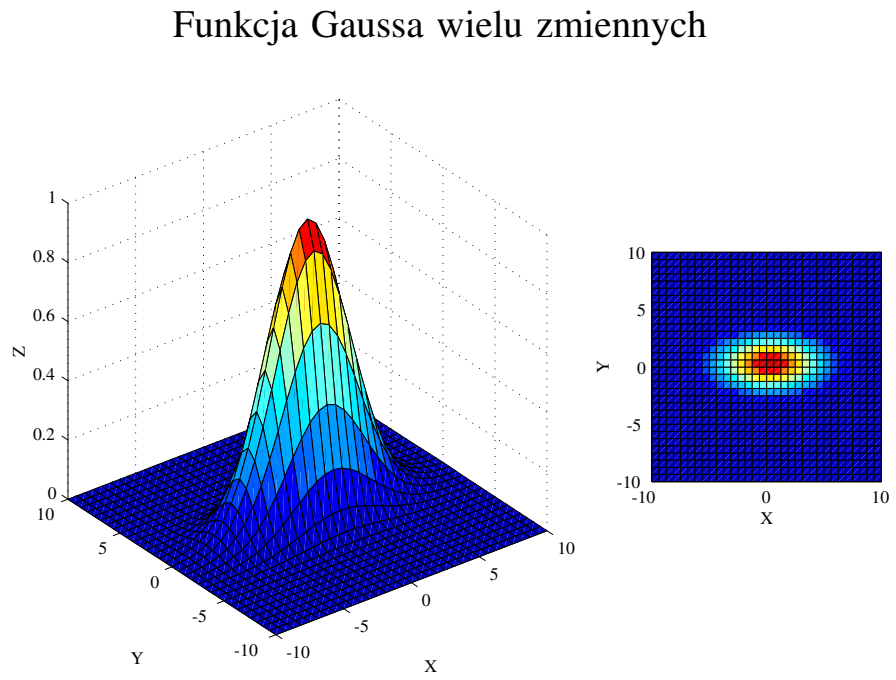
W N wymiarowej przestrzeni wejściowej każda funkcja elipsoidalna używa $2N$ parametrów adaptacyjnych. Używając odległości Mahalanobisa $D_M(\mathbf{x};\mathbf{t})$ (por. 2.16), z symetryczną macierzą kowariancji Σ, dostajemy możliwość dowolnej rotacji hiperelipsoidnych gęstości w wielowymiarowej przestrzeni. Gdy potraktować elementy



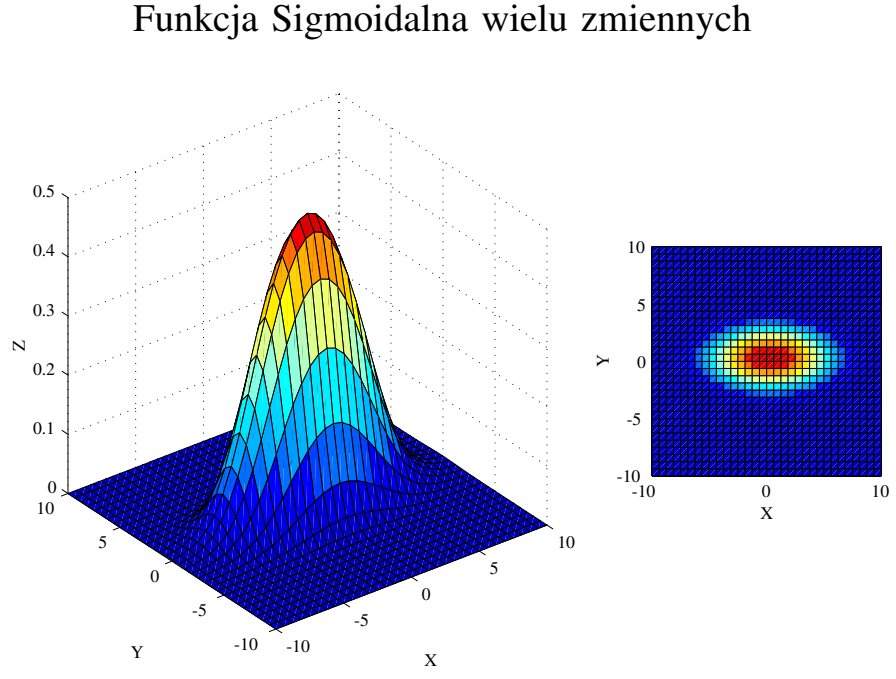
Rysunek 2.18: Wstęgowa funkcja Gaussa (2.77).



Rysunek 2.19: Sigmoidalna funkcja wstęgowa (2.78).



Rysunek 2.20: Funkcja Gaussa wielu zmiennych (2.79).



Rysunek 2.21: Funkcja sigmoidalna wielu zmiennych (2.80).

macierzy kowariancji jako parametry adaptacyjne, będzie to równoważne metryce tensorowej z funkcją odległości:

$$D_Q^2(\mathbf{x};\mathbf{Q};\mathbf{t}) = \sum_{i\geqslant j} Q_{ij}(x_i - t_i)(x_j - t_j) \tag{2.82}$$

Całkowita liczba parametrów adaptacyjnych na jeden neuron wynosi $N(N+3)/2$. Powierzchnie funkcji są dane w ogólnej formie kwadratowej i tym samym mogą być elipsoidalne, paraboliczne czy hiperboliczne.

Pojedynczy neuron może być jeszcze bardziej złożony jeśli zostałaby zastosowana ogólniejsza funkcja odległości. Należy jednak unikać zbyt wielkiej liczby nieliniowych parametrów na jeden neuron. Można też podać jeszcze prostszą postać funkcji o gęstości elipsoidalnej (patrz rys. 2.22):

$$\bar{G}_2(\mathbf{x};\mathbf{t},\mathbf{b}) = \prod_{i=1}^{N} \frac{1}{1+(x_i - t_i)^2/b_i^2} \tag{2.83}$$

Tego wzoru nie można jednak przedstawić w podobnej formie jak powyższe funkcje, uzależniając go od pewnej funkcji odległości.

Przez liniową aproksymację funkcji G_S (pomijając wielowymiarowy iloczyn) otrzymujemy kwadratową funkcję odległości w mianowniku (patrz rys. 2.23):

$$\bar{G}_3(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{1}{1+\sum_{i=1}^{N}(x_i - t_i)^2/b_i^2} = \frac{1}{1+D^{\mathbf{b}}(\mathbf{x};\mathbf{t},\mathbf{b})} \tag{2.84}$$

Funkcje te również dają gęstości hiperelipsoidalne.

Udało się też zaobserwować ciekawą własność funkcji gaussowskiej G_g (2.79) dzięki całkiem prostej renormalizacji[3]:

$$G_R(\mathbf{x};\mathbf{t},\mathbf{b}) = \frac{G_g(\mathbf{x};\mathbf{t},\mathbf{b})}{G_g(\mathbf{x};\mathbf{t},\mathbf{b}) + G_g(\mathbf{x};-\mathbf{t},\mathbf{b})} = \frac{1}{1+e^{-4\sum_{i=1}^{N} x_i t_i/b_i^2}} \tag{2.85}$$

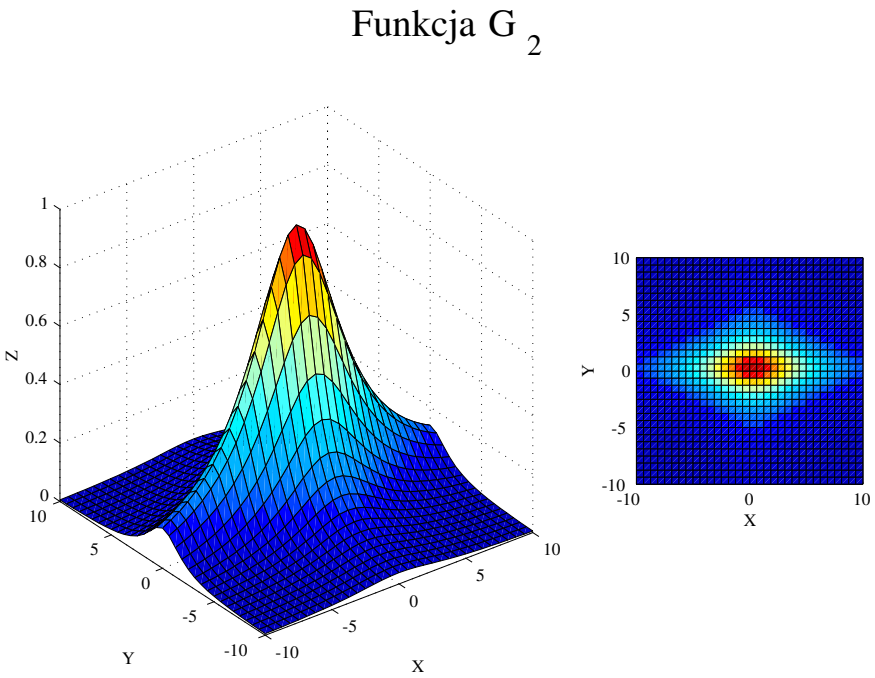
funkcja G_R staje się nielokalna i jest równoważna funkcji sigmoidalnej $\sigma(\mathbf{x};\mathbf{p})$ dla $p_i = b_i^2/4t_i$.

Tym samym można używać sieci RBFN zamiast MLP i vice versa, a prosta transformacja wejścia umożliwia użycie sieci MLP ze zlokalizowanymi funkcjami transferu [37, 38].

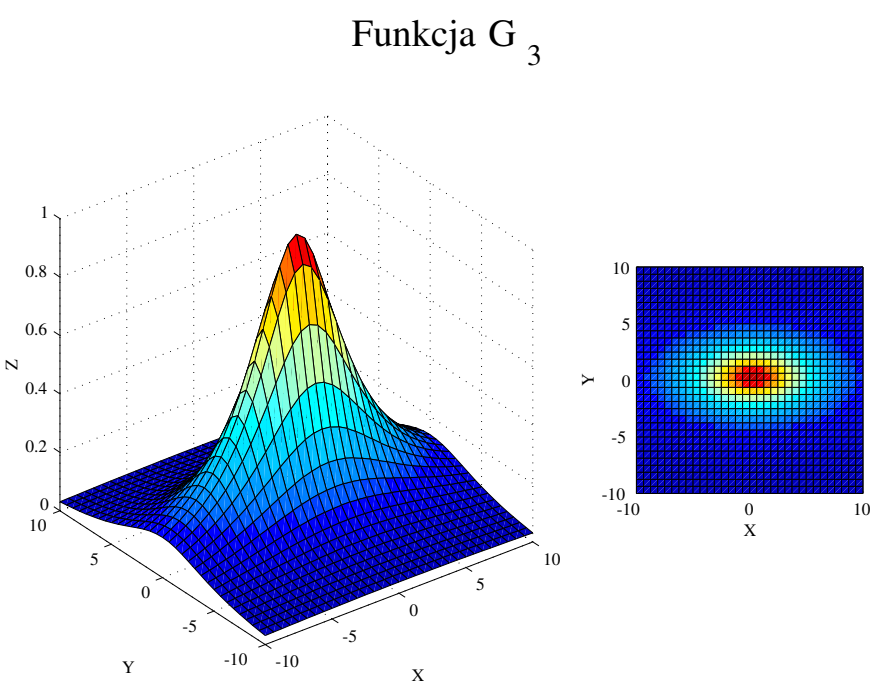
2.4.5. Uniwersalne funkcje transferu

Połączenie liniowego czynnika we wzorze aktywacji $I(\mathbf{x};\boldsymbol{W},\theta)$ i kwadratowego czynnika używanego w liczeniu odległości Euklidesowej, daje w rezultacie funkcję, która

[3] Własność tę pokazał Igor Grabiec z Uniwersytetu Liubljanie Włodzisławowi Duchowi w prywatnej dyskusji.



Rysunek 2.22: Funkcja $\bar{G}_2$ (2.83).



Rysunek 2.23: Funkcja $\bar{G}_3$ (2.84).

dla pewnych parametrów staje się lokalna, a dla innych nielokalna (dalej takie funkcje będą też nazywane funkcjami semi-lokalnymi). Kilka takich funkcji zostało zaprezentowanych niedawno. Ridella i in. [153] używali neuronów kołowych w ich kołowej sieci wstecznej propagacji. Funkcją wyjścia jest standardowa funkcja sigmoidalna lecz funkcja aktywacji składa się z dodatkowego członu (patrz rys. 2.24):

$$A_R(\mathbf{x};\mathbf{w}) = w_0 + \sum_{i=1}^{N} w_i x_i + w_{N+1} \sum_{i=1}^{N} x_i^2 \tag{2.86}$$

I_c może też być przedstawione w formie odległości::

$$\begin{aligned} A_R(\mathbf{x};\mathbf{w}) &= d_c(\mathbf{x};\mathbf{c}) = (\|\mathbf{x}-\mathbf{c}\|^2 - \theta) w_{N+1}; \\ c_i &= -w_i/2w_{N+1}; \quad \theta = \frac{1}{w_{N+1}} \left(\sum_{i=1}^{N} \frac{w_i^2}{4w_{N+1}^2} - w_0 \right) \end{aligned} \tag{2.87}$$

Ridella i in. [153] uzyskali bardzo dobre rezultaty przy użyciu tych neuronów w sieci ze wsteczną propagacją. Udowodnili też, iż w wielu przypadkach neurony kołowe dają optymalne rozwiązanie dla problemów klasyfikacyjnych. Inny typ neuronu kołowego został zaproponowany przez Kirbiego i Mirande [110]. W ich rozwiązaniu dwa neurony sigmoidalne współdziałają razem, a ich wspólne wyjście jest ograniczane tak, aby leżało w kole jednostkowym.

Dorffner [36] zaproponował użycie stożkowej funkcji transferu jako funkcji uniwersalnej dla sieci MLP i RBFN. Linie proste czy elipsy to tylko specjalne przypadki funkcji stożkowej. Z geometrycznych rozważań, Dorffner proponuje połączenie dwóch funkcji aktywacji: liniowej kombinacji wejść i funkcji odległości (patrz rys. 2.25), co razem daje aktywacje:

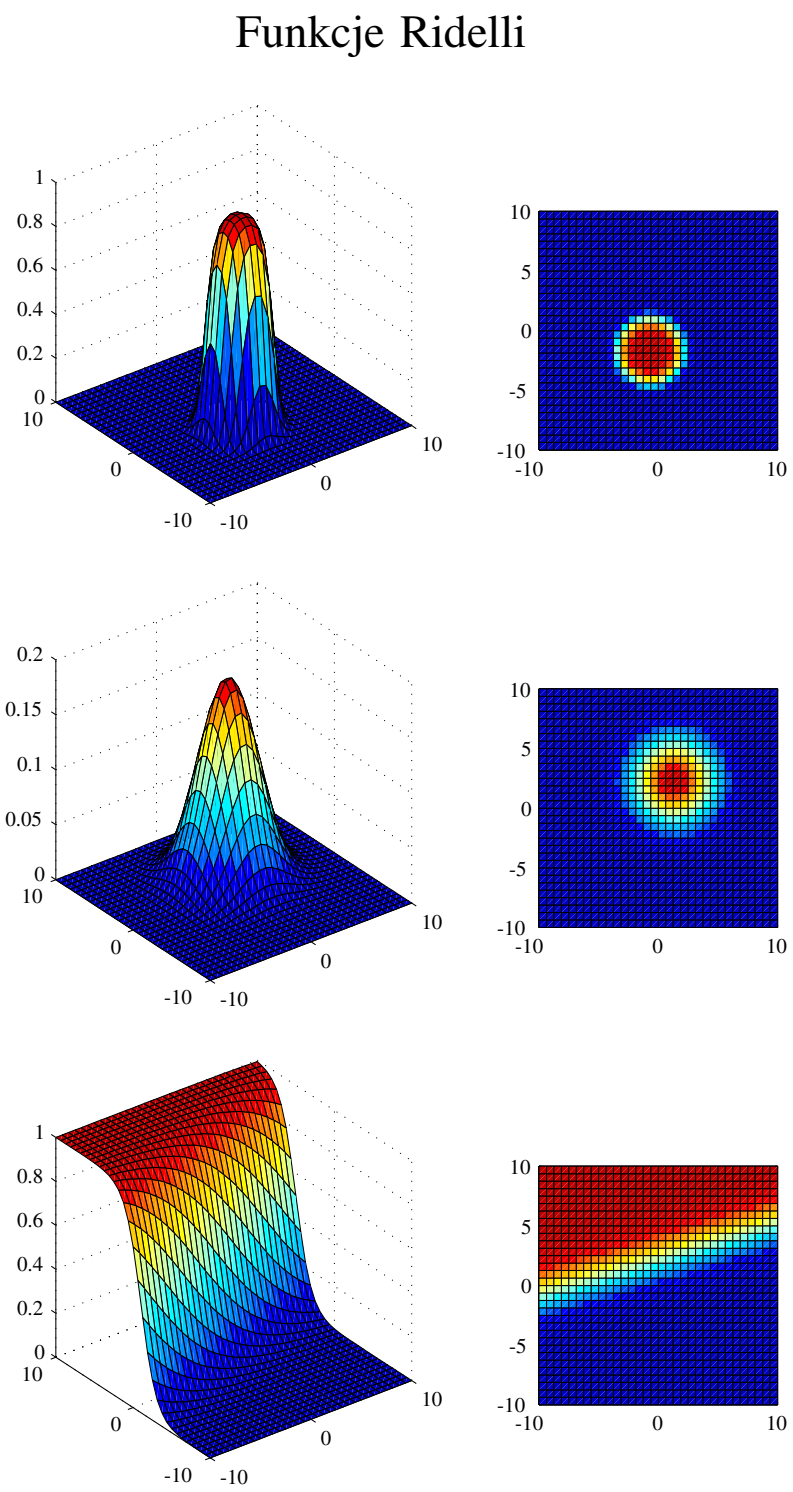
$$\begin{aligned} A_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega) &= I(\mathbf{x}-\mathbf{t};\mathbf{w}) + \omega D(\mathbf{x}-\mathbf{t}) \\ &= \sum_{i=1}^{N+1} w_i(x_i - t_i) + \omega \sqrt{\sum_{i=1}^{N+1} (x_i - t_i)^2} \end{aligned} \tag{2.88}$$

Aktywacja ta jest później przekształcana przez funkcje sigmoidalną by uzyskać końcową stożkową funkcję transferu:

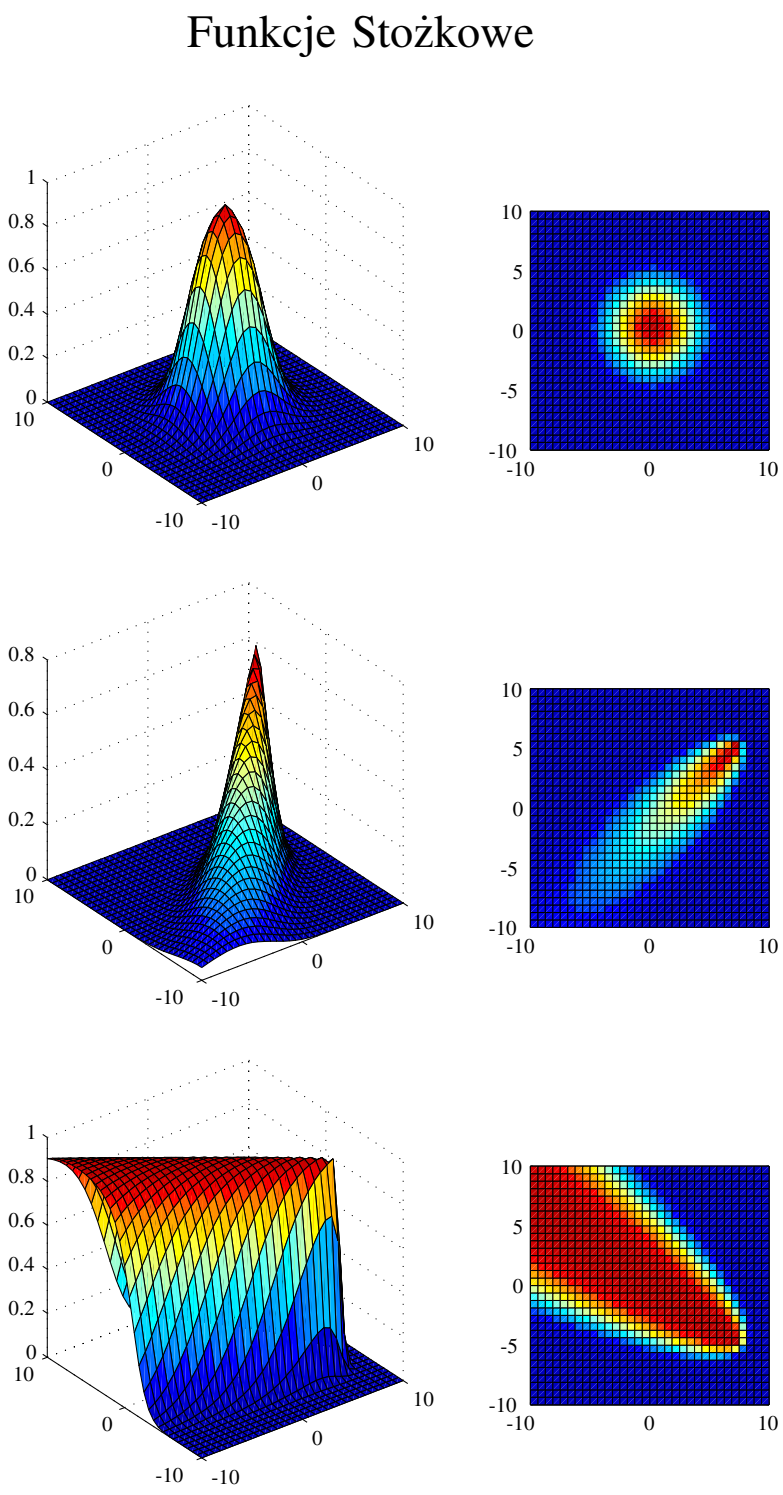
$$C_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega) = 1 - \sigma(A_C(\mathbf{x};\mathbf{w},\mathbf{t},\omega)) \tag{2.89}$$

Z powyższych rozważań można łatwo wywnioskować, że daje się skonstruować i inne funkcje uniwersalne w oparciu o liniową kombinację wejść i funkcje odległości. Na przykład $\exp(\alpha I^2 - \beta D^2)$ lub aproksymacja funkcji gaussowskiej połączona z funkcją Lorentza, dają ciekawe funkcje uniwersalne (patrz rys. 2.26):

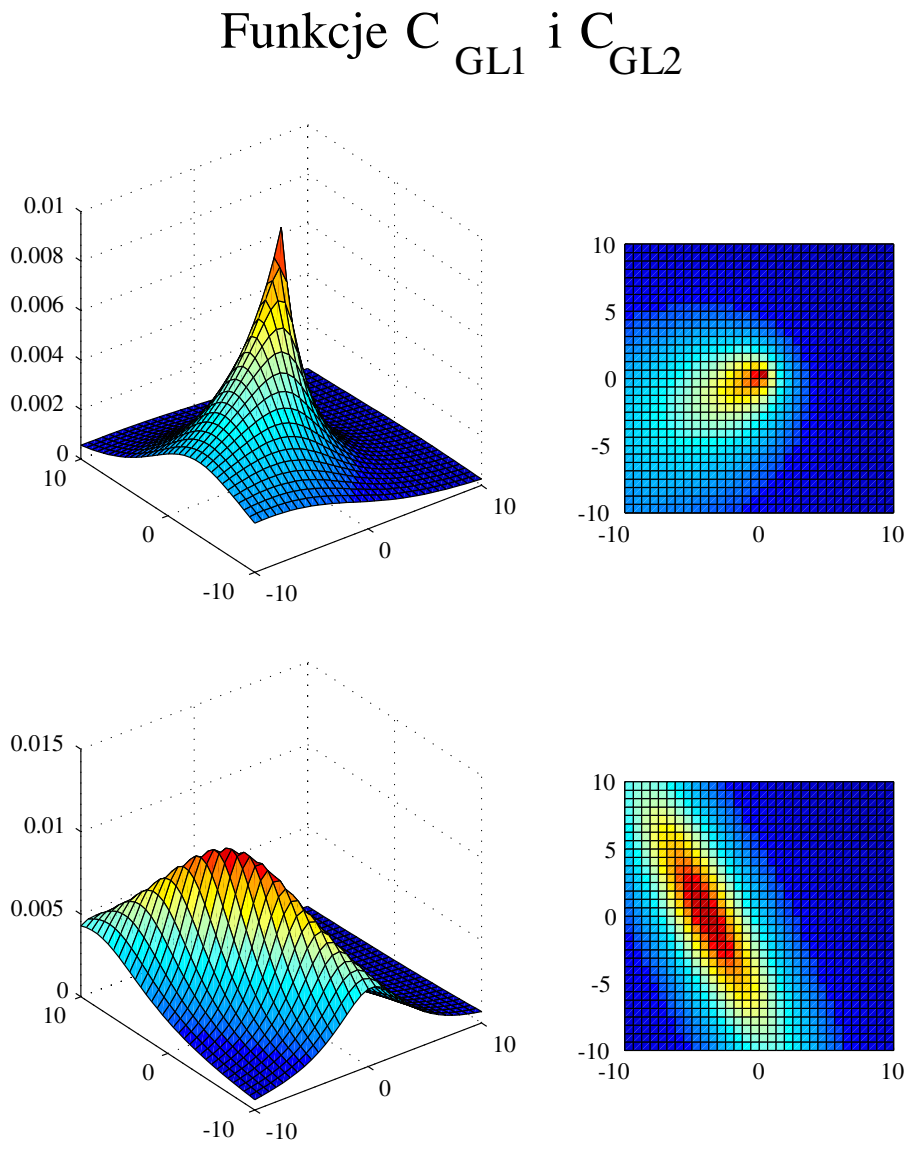
$$C_{GL1}(\mathbf{x};\mathbf{W},\mathbf{t},\alpha,\theta) = \frac{1}{1 + (I(\mathbf{x};\mathbf{W},\theta) + \alpha D(\mathbf{x};\mathbf{t}))^2} \tag{2.90}$$



Rysunek 2.24: Funkcja kołowa Riddelli (2.86).



Rysunek 2.25: Funkcje stożkowe (2.89).



Rysunek 2.26: Kombinacja aproksymacji funkcji gaussowskiej z funkcją Lorentza (2.90 i 2.91).

lub

$$C_{GL2}(\mathbf{x}; \boldsymbol{W}, \mathbf{t}, \alpha, \theta) = \frac{1}{1 + \alpha I^2(\mathbf{x}; \boldsymbol{W}, \theta) + \beta D^2(\mathbf{x}; \mathbf{t})} \tag{2.91}$$

Dla uproszczenia można przyjąć iż $\beta = 1 - \alpha$. Parametr α waży względny udział liniowej i nieliniowej części w całości funkcji. W tym przypadku liczba adaptacyjnych parametrów wynosi $2N + 1$ — gdy nie ma odrębnego parametru skalującego przy części z funkcją odległości; lub $3N + 1$ — gdy są odrębne parametry skalujące przy liniowej i nie liniowej części. Niestety funkcje uniwersalne są nieseparowalne (por. z podrozdział 2.4.7, w którym również będą przedstawione funkcje semi-lokalne).

2.4.6. Funkcje bicentralne

Kombinacja dwóch funkcji sigmoidalnych daje możliwość utworzenia zlokalizowanej funkcji typu okno na kilka różnych sposobów. Dwoma najprostszymi sposobami są: różnica dwóch funkcji sigmoidalnych $\sigma(x) - \sigma(x - \theta)$ i iloczyn $\sigma(x)(1 - \sigma(x))$. Po renormalizacji obie formy stają się identyczne:

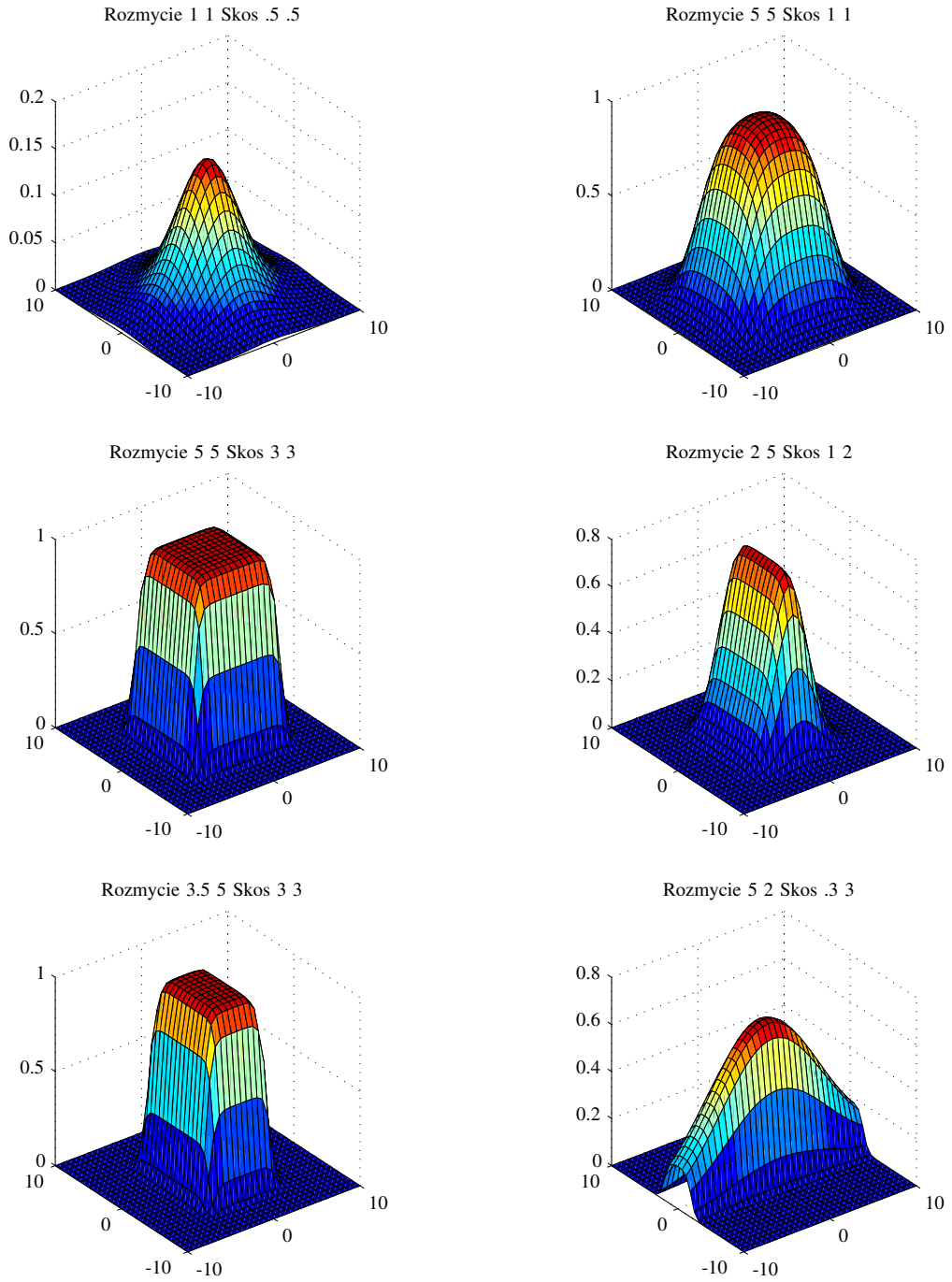
$$\frac{\sigma(x + b)(1 - \sigma(x - b))}{\sigma(b)(1 - \sigma(-b))} = \frac{\sigma(x + b) - \sigma(x - b)}{\sigma(b) - \sigma(-b)} \tag{2.92}$$

Gdy wziąć iloczyn takich kombinacji funkcji w każdym wymiarze, otrzymujemy wielowymiarową funkcję, która jest bardzo elastyczna, produkuje wypukłe regiony decyzyjne, wygodne dla klasyfikacji. Ogólna postać produktu w N wymiarowej przestrzeni wygląda tak (por. rys. 2.4.6):

$$\begin{aligned} Bi(\mathbf{x}; \mathbf{t}, \mathbf{b}, \mathbf{s}) &= \prod_{i=1}^{N} \sigma(A1_i^+)\ (1 - \sigma(A1_i^-)) \\ &= \prod_{i=1}^{N} \sigma(e^{s_i} \cdot (x_i - t_i + e^{b_i}))\ (1 - \sigma(e^{s_i} \cdot (x_i - t_i - e^{b_i}))) \end{aligned} \tag{2.93}$$

gdzie $\sigma(x) = 1/(1 + e^{-x})$, $\mathbf{t}$ to centrum, $\mathbf{b}$ rozmycie, a $\mathbf{s}$ skos. Pierwszy człon z funkcją sigmoidalną rośnie gdy rosną wartości x_i, drugi człon będzie wtedy malał. To właśnie lokalizuje funkcję wokół punktu t_i w każdym z wymiarów. Kształt bicentralnej funkcji gęstości $Bi(\mathbf{x}; \mathbf{t}, \mathbf{b}, \mathbf{s})$ można adaptować poprzez przemieszczanie punktu centrum $\mathbf{t}$, zmianę rozmycia $\mathbf{b}$ i ustawianie skosu $\mathbf{s}$. Dzięki niezależności parametrów rozmyć i skosów, funkcje bicentralne mogą estymować bardziej skomplikowane rozkłady gęstości niż na przykład wielowymiarowe funkcje gaussowskie. Radialne funkcje bazowe są zdefiniowane względem tylko jednego centrum $\|x - t\|$. Tu są używane dwa centra $t_i + e^{b_i}$ i $t_i - e^{b_i}$. Stąd pochodzi nazwa funkcji bicentralnych. Produkt funkcji daje z kolei elastyczne wypukłe gęstości. Zostały też zastosowane funkcje eksponencjalne do rozmycia e^{b_i} i skosu e^{b_i} zamiast s_i i b_i. Zmniejszyło to oscylacje funkcji błędu podczas uczenia, co stabilizuje proces uczenia.

Liczba parametrów adaptacyjnych na jeden neuron wynosi $3N$. Jest też możliwa redukcja wymiarów, podobnie, jak w przypadku wstęgowych funkcji gaussowskich.



Rysunek 2.27: Kilka przykładów gęstości funkcji bicentralnych (2.93).

Lecz i w tym przypadku funkcje bicentralne są bardziej elastyczne, a wykorzystują tyle samo parametrów co funkcje wstęgowe.

Funkcje bicentralne mogą zostać w prosty sposób rozszerzone do semi-lokalnych poprzez dodanie dwóch (lub jednego) parametrów:

$$\begin{aligned} \mathrm{SBi}(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) &= \prod_{i=1}^{N}(\alpha+\sigma(A1_i^+))\ (1-\beta\sigma(A1_i^-)) \\ &= \prod_{i=1}^{N}(\alpha+\sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i})))\ (1-\beta\sigma(e^{s_i}\cdot(x_i-t_i-e^{b_i}))) \end{aligned} \tag{2.94}$$

Funkcja nie zanika (jej wartości są różne od zera) dla dużych wartości $|x|$ (gdy tylko $\alpha \neq 0$ i $\beta \neq 0$). Dla $\alpha = 0$, $\beta = 0$ funkcja semi-bicentralna jest równoważna funkcji bicentralnej. Liczba parametrów adaptacyjnych funkcji semi-bicentralnej jest równa $3N + 2$ jeśli parametry α i β są takie same dla wszystkich wymiarów przestrzeni wejściowej lub $5N$ w przeciwnym wypadku.

2.4.7. Rozszerzenia funkcji bicentralnych

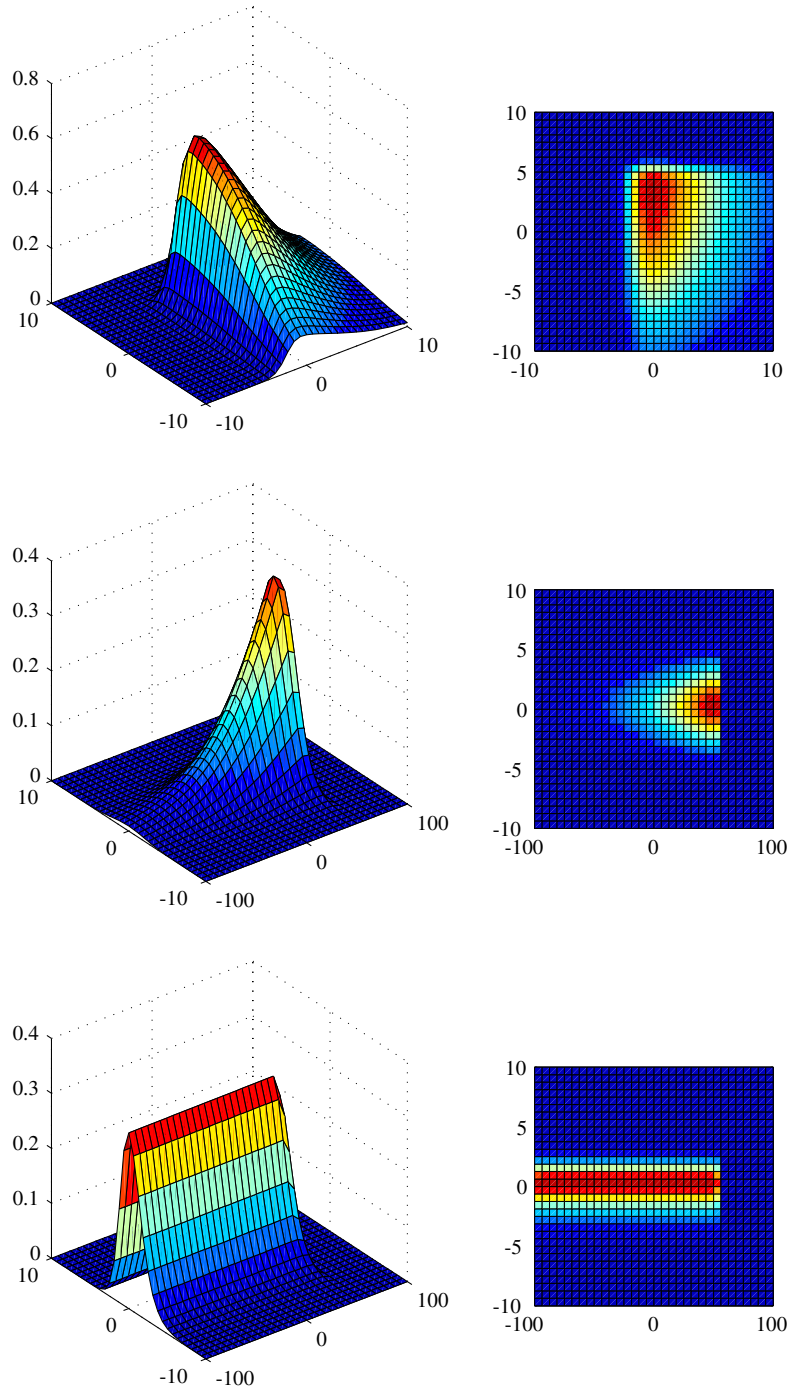
Funkcje bicentralne z niezależnymi skosami.

Innym sposobem kontrolowania gęstości funkcji estymującej jest użycie niezależnych skosów dla kombinacji funkcji sigmoidalnych (patrz rys. 2.28):

$$\begin{aligned} \mathrm{Bi2s}(\mathbf{x};\mathbf{t},\mathbf{b},\mathbf{s}) &= \prod_{i=1}^{N}\sigma(A2_i^+)\ (1-\sigma(A2_i^-)) \\ &= \prod_{i=1}^{N}\sigma(e^{s_i}\cdot(x_i-t_i+e^{b_i}))\ (1-\sigma(e^{s_i'}\cdot(x_i-t_i-e^{b_i}))) \end{aligned} \tag{2.95}$$

Ustawiając skos s_i lub s_i' na małą wartość, funkcja bicentralna może się delokalizować lub też rozciągać na lewo i/lub prawo od punktu centrum t, w każdym z wymiarów niezależnie. To pozwala na uzyskanie takich powierzchni gęstości, jak półhiperwalec, półhiperelipsoida, *miękki* trójkąt i inne (patrz rys. 2.28). Choć liczba parametrów trochę wzrosła, i wynosi teraz $4N$ parametrów na jeden neuron, to jednak elastyczność adaptacji estymowanej funkcji jest widocznie większa.

Należy zwrócić uwagę, iż podobnie, jak funkcje bicentralne ta funkcja jest również separowalna co umożliwia różnym metodom zagłębianie się w rozmaite podprzestrzenie przestrzeni wejściowej. Można również, na przykład manipulując funkcją błędu, dodatkowo wymuszać preferowanie małych, łagodnych skosów. Prowadzi to do redukcji niezależnie w poszczególnych wymiarach poszczególnych neuronów. Można



Rysunek 2.28: Przykłady funkcji bicentralnych z niezależnymi skosami (2.95).

też wymuszać preferencje dużych skosów i/lub rozciągać rozmycia, co przyczynia się bezpośrednio do możliwości interpretacji sieci jako reguł logicznych i również redukcji wymiarów. Metody interpretacji sieci opartych głównie o funkcje bicentralne były opisane w [47, 1, 44].

Funkcje bicentralne z rotacją.

Funkcja bicentralna opisana w podrozdziale 2.4.6, posiadająca 3N parametrów adaptacyjnych na neuron, jest wystarczająca do reprezentowania wielu różnych klas gęstości. Z kolei funkcja semi-bicentralna czy bicentralna z niezależnymi skosami może opisywać zlokalizowane, jak i niezlokalizowane funkcje. Następnym krokiem w kierunku zwiększenia elastyczności funkcji transferu jest dołączenie możliwości rotacji gęstości poszczególnych neuronów w wielowymiarowej przestrzeni wejściowej [48, 47]. Oczywiście można użyć pełnej macierzy przekształcenia w przestrzeni wejściowej na wektorze, $\mathbf{R}\mathbf{x}$, lecz w praktyce macierz taka jest bardzo trudna do adaptacji (zawiera $N \times N$ parametrów), poza tym jest w niej jedynie $N-1$ niezależnych kątów obrotu (na przykład eulerowskich). Dużych kłopotów nastręczałoby też wyznaczanie pochodnych dla algorytmów uczenia opartych na metodzie spadku gradientu. Można zaproponować dwie drogi rozwiązania tego problemu. W obu będzie możliwy obrót we wszystkich wymiarach przestrzeni przy użyciu jedynie $N-1$ lub N parametrów obrotu. W pierwszym przypadku jest to produkt kombinacji sigmoid opartych o transformacje na wektorze wejściowym (patrz rys. 2.29):

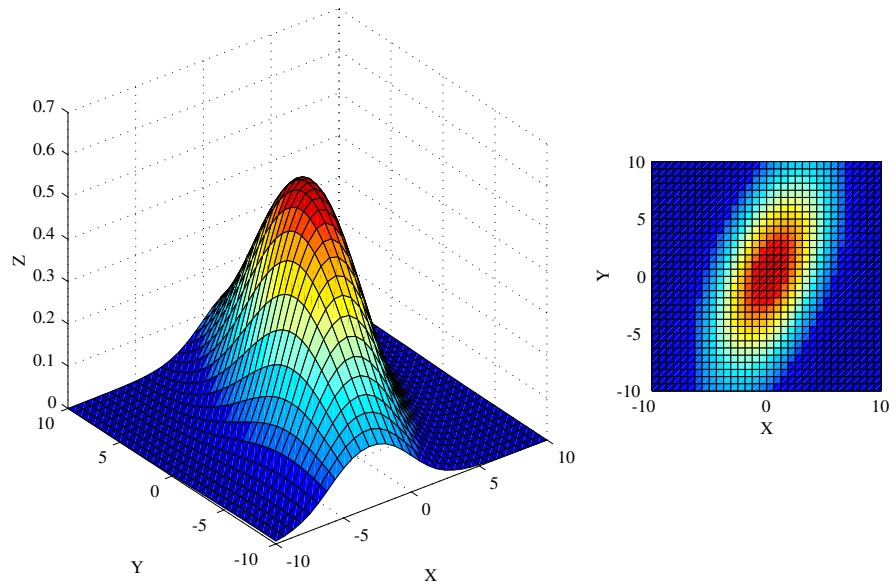
$$\begin{aligned} C_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{R}) &= \prod_i^N \left(\sigma(A3_i^+) - \sigma(A3_i^-)\right) \qquad (2.96)\\ &= \prod_i^N \left(\sigma(\mathbf{R}_i\mathbf{x}+t_i) - \sigma(\mathbf{R}_i\mathbf{x}+t_i')\right) \\ SC_P(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{p},\mathbf{r},\mathbf{R}) &= \prod_i^N \left(p_i\cdot\sigma(A3_i^+) + r_i\cdot\sigma(A3_i^-)\right) \qquad (2.97)\\ &= \prod_i^N \left(p_i\cdot\sigma(\mathbf{R}_i\mathbf{x}+t_i) + r_i\cdot\sigma(\mathbf{R}_i\mathbf{x}+t_i')\right) \end{aligned}$$

gdzie $\mathbf{R}_i$ jest i-tym wierszem macierzy rotacji $\mathbf{R}$ o następującej strukturze:

$$\mathbf{R} = \begin{bmatrix} s_1 & \alpha_1 & 0 & & \cdots & 0 \\ 0 & s_2 & \alpha_2 & 0 & & \\ \vdots & & & \ddots & & \vdots \\ & & & & s_{N-1} & \alpha_{N-1} \\ 0 & \cdots & & & 0 & s_N \end{bmatrix} \qquad (2.98)$$

gdzie s_i to parametry opisujące skosy, a parametry α_i opisują obrót funkcji względem jej centrum $\mathbf{t}-\mathbf{t}'$.

Gdy $p_i = 1$ i $r_i = -1$ funkcja SC_P staje się funkcją lokalną, równoważną funkcji



Rysunek 2.29: Funkcje bicentralne z rotacją (2.96).

C_P i podobną do funkcji bicentralnej, pomijając możliwość rotacji. Przypisując inne wartości parametrom p_i i r_i funkcja SC_P delokalizuje się.

Drugi sposób uzyskania funkcji z możliwością obrotu polega na utworzeniu sumy kombinacji funkcji sigmoidalnych typu okna $L(x;t,t') = \sigma(x+t) - \sigma(x+t')$ w $N-1$ wymiarach i kombinacji obróconej przez wektor $\mathbf{K}$:

$$C_K(\mathbf{x};\mathbf{t},\mathbf{t}',\boldsymbol{W},\mathbf{K}) = \sum_{i=1}^{N-1} W_i L(x_i,t_i,t_i') + W_N L(\mathbf{Kx},t,t') \tag{2.99}$$

Ta gęstość funkcji jest prostopadła do wektora $\mathbf{K}$. Czyniąc funkcje $C_K(\cdot)$ funkcją aktywacji, a funkcje sigmoidalną funkcją wyjścia z odpowiednim progiem, na wyjściu otrzymamy gęstości prostopadłe do $\mathbf{K}$. Alternatywnie można użyć też iloczynu kombinacji a nie sumy:

$$C_{PK}(\mathbf{x};\mathbf{t},\mathbf{t}',\mathbf{K}) = L(\mathbf{Kx},t,t') \prod_{i=1}^{N-1} L(x_i,t_i,t_i') \tag{2.100}$$

w tym przypadku nie musimy używać funkcji sigmoidalnej jako funkcji wyjścia.

Rotacja w tych funkcjach dodaje jedynie $N-1$ parametrów dla funkcji $C_P(\cdot)$ i N parametrów dla funkcji $C_K(\cdot)$.

Rotacje (które mają możliwość adaptacji podczas procesu uczenia) zostały, jak dotąd zaimplementowane jedynie w dwóch typach sieci neuronowych, w Feature Space Mapping [45, 1, 44] i IncNet [104, 100, 101] opisanej w tej pracy.

Funkcje bicentralne z rotacją i niezależnymi skosami.

Można również połączyć możliwości funkcji bicentralnych z obrotem i niezależnymi skosami, tworząc jeszcze elastyczniejszą funkcję transferu (patrz rys. 2.30):

$$\mathrm{BiR2s}(\mathbf{x};\mathbf{t},\mathbf{t}',\alpha) = \prod_i^N \sigma(A4_i^+)(1-\sigma(A4_i^-)) \tag{2.101}$$

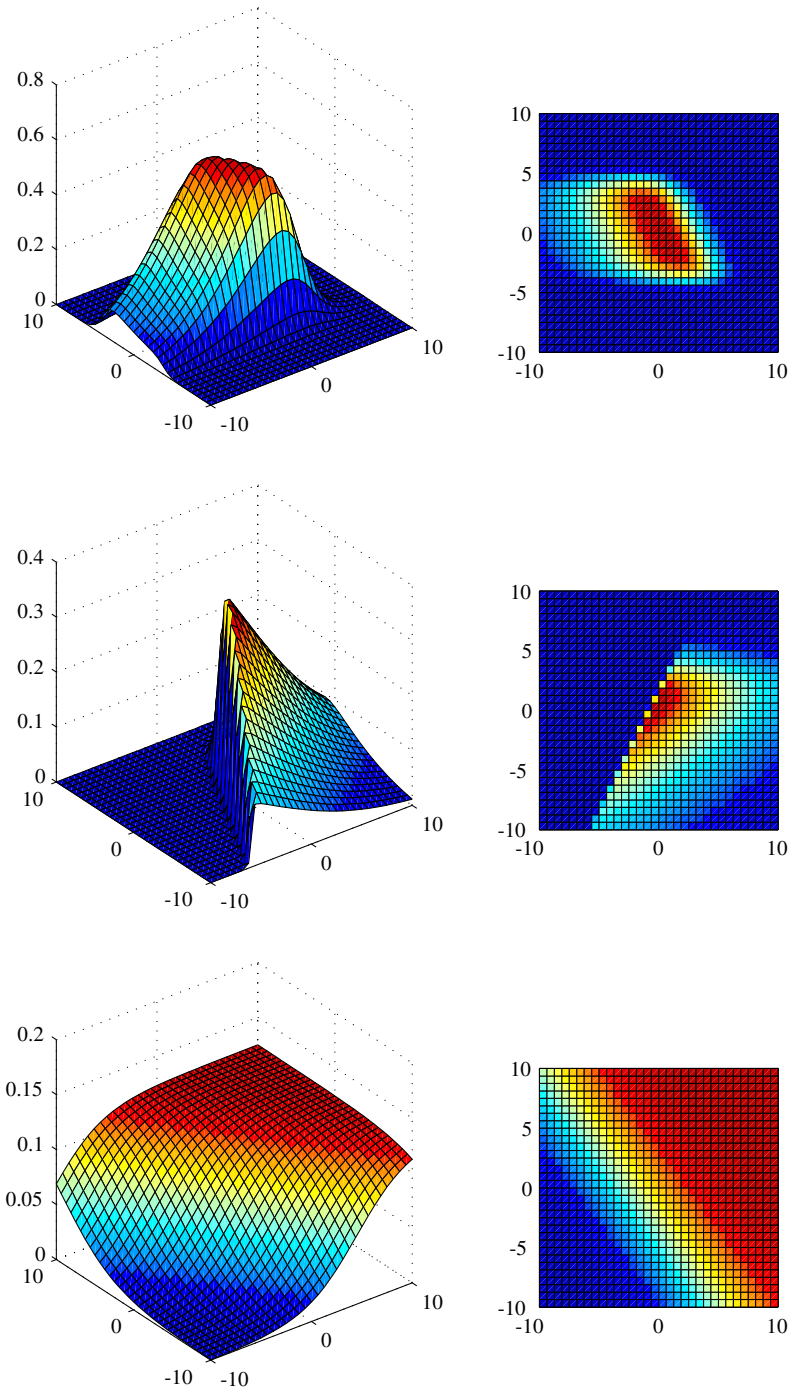
$$= \prod_i^N \sigma(s_i(x_i+\alpha_i x_{i+1}-t_i+b_i))(1-\sigma(s_i'(x_i+\alpha_i x_{i+1}-t_i-b_i)))$$

gdzie $\alpha_1,\ldots,\alpha_{N-1}$ definiują obrót, przyjmuje się iż $x_{N+1}=0$ i $\alpha_N=0$. Gęstości tej funkcji mogą być obrócone, funkcja może być lokalna lub semi-lokalna, jak i niesymetryczna w poszczególnych wymiarach. Liczba parametrów adaptacyjnych wynosi $5N$ i tym samym stanowi silną alternatywę dla funkcji SBi (2.94).

Bardzo ważną i zarazem pożyteczną cechą funkcji bicentralnych jest separowalność. Typowe sigmoidalne funkcje transferu nie są separowalne, a pośród typowych funkcji radialnych tylko funkcja Gaussa jest funkcją separowalną. Funkcje bicentralne, choć są skonstruowane za pomocą funkcji sigmoidalnych, są separowalne dzięki iloczynowi par sigmoid po wszystkich wymiarach wejściowych. Dzięki temu zawsze można dokonać opuszczenia pewnego wymiaru czy też kilku wymiarów i proces analizy danych modelu może być z powodzeniem kontynuowany. Właściwości te można wykorzystywać gdy mamy do czynienia z niepełnymi (brakującymi) danymi, wyciąganiem reguł logicznych z sieci neuronowych czy implementacją pamięci asocjacyjnej za pomocą sieci neuronowych [45, 1, 39, 40].

2.4.8. Końcowe porównanie lokalnych i nielokalnych funkcji transferu

Porównanie różnych funkcji transferu zostało przedstawione w tabeli 2.1. W pierwszej kolumnie znajduje się nazwa funkcji transferu; druga kolumna pokazuje numer równania definiującego funkcje; trzecia kolumna pokazuje użytą funkcję aktywacji. Kolejna kolumna pokazuje liczbę adaptacyjnych parametrów dla d wymiarowej przestrzeni wejściowej. Dla funkcji wielostopniowej k jest liczbą stopni funkcji (zazwyczaj nie podlegają one jednak adaptacji). W piątej kolumnie mamy informację, która określa czy funkcja transferu ma lokalny, czy nielokalny charakter (niektóre funkcje mogą być i lokalne i nielokalne, w zależności od stanu parametrów. Kolejna kolumna pokazuje typ(-y) parametrów adaptacyjnych: $\boldsymbol{w}$ – liniowe parametry wag, θ – progi, $\mathbf{t}$ – centra, $\mathbf{b}$ i b są parametrami odpowiadającymi dyspersją lub parametrami odpowiadającymi za skalowanie cech, $\mathbf{s}$ określaj skosy, $\mathbf{R}$ parametry rotacji, o i O



Rysunek 2.30: Funkcje bicentralne z rotacją i niezależnymi skosami (2.101).

oznaczają inne parametry adaptacyjne. Ostatnie dwie kolumny opisują, czy funkcja jest separowalna (Y), symetryczna (S), asymetryczna (A) lub niesymetryczna (N).

Zaprezentowane funkcje transferu, wyjścia i aktywacji prezentują szeroką gamę przeróżnych właściwości i możliwości, które niewątpliwie są bardzo przydatne w rozwiązywaniu złożonych problemów. Innym sposobem zwiększenia możliwości modeli adaptacyjnych jest dodanie nowych cech wejściowych, stworzonych poprzez różne transformacje nieliniowe cech pierwotnych. Więcej informacji można znaleźć w pracy Ducha i Jankowskiego [48], jak i [144, 139, 169, 168, 158].

Funkcja	Nr	Aktywacja	Liczba parametrów	Lokalna Nielokal.	Typ parametrów	Separowalność	Symetria
Schodkowa	(2.2)	I	$d+1$	NL	$\boldsymbol{w}, \theta$		A
Wieloschodkowa	(2.3)	I	$d+k$	NL	$\boldsymbol{w}, \Theta$		A
Semi-liniowa	(2.4)	I	$d+2$	NL	$\boldsymbol{w}, \theta$		A
Logistyczna	(2.5)	I	$d+1$	NL	$\boldsymbol{w}, \theta$		A
tanh, arctan	(2.46)	I	$d+1$	NL	$\boldsymbol{w}, \theta$		A
s_1	(2.48)	I	$d+1$	NL	$\boldsymbol{w}, \theta$		A
s_2	(2.49)	I	$d+1$	NL	$\boldsymbol{w}, \theta$		A
s_3	(2.50)	I	$d+1$	NL	$\boldsymbol{w}, \theta$		A
s_4	(2.51)	I	$d+1$	NL	$\boldsymbol{w}, \theta$		A
Sferyczna	(2.56)	D	d	NL	$\mathbf{t}, b$		S
Potęgowa	(2.58)	D	$d+2$	L+NL	$\mathbf{t}, b, o$		S
Sklejana	(2.60)	D	$d+1$	NL	$\mathbf{t}, b$		S
Gaussa	(2.61)	D	$d+1$	L	$\mathbf{t}, b$	Y	S
$G_1 = 2 - 2\sigma(r^2)$	(2.62)	D	$d+1$	L	$\mathbf{t}, b$		S
$G_2 = \tanh(r^2)$	(2.63)	D	$d+1$	L	$\mathbf{t}, b$		S
G_3	(2.64)	D	$d+1$	L	$\mathbf{t}, b$		S
G_4	(2.65)	D	$d+1$	L	$\mathbf{t}, b$		S
RCBSpline	(2.66)	D	$d+1$	L	$\mathbf{t}, b$		S
RQBSpline	(2.67)	D	$d+1$	L	$\mathbf{t}, b$		S
Wstęgowa Gaussa	(2.77)	D_i	$3d$	L	$\mathbf{t}, \mathbf{b}, O$	Y	S
Wstęgowa sigmoidalna	(2.78)	D_i	$3d$	L	$\mathbf{t}, \mathbf{b}, O$	Y	S
Gaussa wiel. zm.	(2.79)	D	$2d$	L	$\mathbf{t}, \mathbf{b}$	Y	S
Sigmoidalna w. zm.	(2.80)	D	$2d$	L	$\mathbf{t}, \mathbf{b}$		S
$\bar{G}_2$	(2.83)	D_i	$2d$	L	$\mathbf{t}, \mathbf{b}$	Y	S
$\bar{G}_3$	(2.84)	D	$2d$	L	$\mathbf{t}, \mathbf{b}$		S
Lorentza	(2.71)	I	$d+1$	NL	$\mathbf{t}, \mathbf{s}$		N
Iloczyn tensorowy	(2.72)	D_i	$2d+1$	NL	$\boldsymbol{w}, \theta$		N
G_R	(2.85)	D	$2d$	NL	$\mathbf{t}, \mathbf{b}$		N
Ridelli	(2.86)	A_R	$d+3$	L+NL	$\boldsymbol{w}, \theta$		S+N
Stożkowa	(2.89)	A_C	$2d+2$	L+NL	$\mathbf{t}, \boldsymbol{w}, \theta, o$		S+N
C_{GL1}	(2.90)	A_{GL1}	$2d+3$	L+NL	$\mathbf{t}, \boldsymbol{w}, \theta, o$		S+N
C_{GL2}	(2.91)	A_{GL2}	$2d+4$	L+NL	$\mathbf{t}, \boldsymbol{w}, \theta, o$		S+N
Bicentralna	(2.93)	A1	$3d$	L	$\mathbf{t}, \mathbf{b}, \mathbf{s}$	Y	S
Semi-bicentralna	(2.94)	A2	$5d$	L+NL	$\mathbf{t}, \mathbf{b}, \mathbf{s}, O$	Y	S+N
Bicentral z 2 sk.	(2.95)	A2	$4d$	L+NL	$\mathbf{t}, \mathbf{b}, \mathbf{s}$	Y	S+N
Bicentral z rot.	(2.96)	A3	$4d-1$	L	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R$	Y/N	S
Semi-bic. z rot.	(2.97)	A3	$6d-1$	L+NL	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R, O$	Y/N	S+N
C_K z rotacją	(2.99)	A1, A1($\mathbf{kx}$)	$4d$	L	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R$	Y/N	S
C_{PK} z rotacją	(2.100)	A1, A1($\mathbf{kx}$)	$4d$	L	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R$	Y/N	S
Bicentral, rot., 2 sk.	(2.101)	A4	$5d-1$	L+NL	$\mathbf{t}, \mathbf{b}, \mathbf{s}, R$	Y/N	S+N

Tabela 2.1: Porównanie różnych funkcji transferu. Symbole użyte w tabeli zostały wyjaśnione w tekście.

Rozdział 3

Sieci z radialnymi funkcjami bazowymi

W bieżącym rozdziale będzie można prześledzić wiele przeróżnych podejść do sieci z radialnymi funkcjami bazowymi (RBF), różne metody uczenia z nadzorem i bez nadzoru, metody inicjalizacji i przeróżne własności rozmaitych modeli sieci. Należy także wspomnieć o szerokiej gamie zastosowań. Jednym pierwszych ciekawych zastosowań było użycie sieci RBF do przetwarzania obrazów [148]. Szczególnie ciekawe są ostatnie rezultaty Poggia [147]. Ciekawe są rezultaty w rozpoznawaniu wzorców [70]. Interesujące są również rezultaty uzyskane w rozpoznawaniu i przetwarzaniu mowy [140, 106]. Dobre rezultaty uzyskano też stosując sieci RBF do analizy szeregów czasowych (w tym również finansowych) [108, 135, 19, 86, 92, 93]. Jeszcze inne przykłady obejmują zastosowanie sieci RBF do różnych zagadnień medycznych (patrz rozdział 5). Interesujące było także zastosowanie sieci RBF do analizy chromosomów myszy [137].

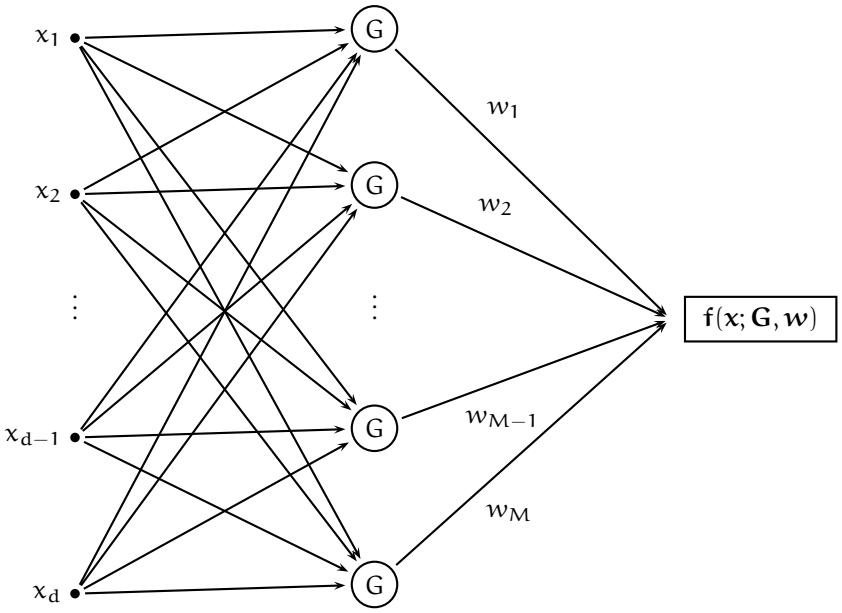
3.1. Sieci z radialnymi funkcjami bazowymi i regularyzacją

Sztuczne sieci neuronowe są wykorzystywane do rozwiązywania wielu różnych problemów takich, jak klasyfikacja, aproksymacja, rozpoznawanie wzorców, przetwarzania sygnałów, przewidywania szeregów czasowych, pozyskiwania reguł logicznych, pamięci asocjacyjnych, samoorganizacji danych, itp. Większość z tych problemów modele sieci neuronowych rozwiązują poprzez uczenie nieznanego odwzorowania pomiędzy przestrzenią wejściową i wyjściową wektorów uczących z pewnego zbioru $\mathcal{S} = \{\langle \mathbf{x}_1, y_1 \rangle, \ldots, \langle \mathbf{x}_n, y_n \rangle\}$, gdzie $\langle \mathbf{x}_i, y_i \rangle$ jest wzorcem uczącym, który jest parą wartości wektora przestrzeni wejściowej i wartości z przestrzeni wyjściowej ($\mathbf{x}_i \in \mathcal{R}^N$, $y_i \in \mathcal{R}$). Tak przedstawione odwzorowanie $F(\cdot)$ można zapisać w poniż-

szej formie:

$$F(\mathbf{x}_i) = y_i + \eta, \qquad i = 1, \ldots, n \tag{3.1}$$

η jest szumem z zerową wartością oczekiwaną i rozmyciem równym σ^2_{ns}.



Rysunek 3.1: Sieć z radialnymi funkcjami bazowymi.

Pierwotnie sieci o radialnych funkcjach bazowych były dedykowane głównie do problemów aproksymacji w przestrzeniach wielowymiarowych [151, 52, 60, 76, 150]. Tym samym proces uczenia rekonstruował powierzchnie nieznanego odwzorowania $F(\cdot)$ za pomocą wzorców uczących ze zbioru $\mathcal{S}$, poprzez adaptacje wolnych parametrów modelu, którymi na początku były tylko wagi połączeń warstwy ukrytej z warstwą wyjściową. Typową sieć o radialnych funkcjach bazowych (rys. 3.1) można zapisać poniższym wzorem:

$$f(\mathbf{x}; \boldsymbol{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i). \tag{3.2}$$

M określa liczbę neuronów warstwy ukrytej, a $G_i(\mathbf{x}, \mathbf{p}_i)$ to jedna z radialnych funkcji bazowych jako funkcja transferu i-tego neuronu warstwy ukrytej, wektory $\mathbf{p}_i$ to parametry adaptowalne i-tego neuronów warstwy ukrytej takie jak centrum (położenia i-tego neuronu) rozmycie i inne, zależnie od wyboru funkcji $G_i(\cdot)$. Najpopularniejszymi funkcjami radialnymi są: funkcja gaussowska, funkcja sferyczna, funkcje

potęgowe i sklejane (rys. 2.10, 2.8, 2.9, 2.7):

$$h_1(\mathbf{x};\mathbf{t},b) = e^{-||\mathbf{x}-\mathbf{t}||^2/b^2} \tag{3.3}$$
$$h_2(\mathbf{x};\mathbf{t},b) = ||\mathbf{x}-\mathbf{t}||/b^2 \tag{3.4}$$
$$h_3(\mathbf{x};\mathbf{t},b,\alpha) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{-\alpha}, \quad \alpha>0 \tag{3.5}$$
$$h_4(\mathbf{x};\mathbf{t},b,\beta) = (b^2+||\mathbf{x}-\mathbf{t}||^2)^{\beta}, \quad 0<\beta<1 \tag{3.6}$$
$$h_5(\mathbf{x};\mathbf{t},b) = (b||\mathbf{x}-\mathbf{t}||)^2\ln(b||\mathbf{x}-\mathbf{t}||) \tag{3.7}$$

Radialne funkcje bazowe są również aproksymatorem uniwersalnym [81, 145]. Więcej informacji o funkcjach RBF można znaleźć w poprzednim rozdziale, a szczególnie w podrozdziale 2.3.2.

Funkcje radialne, jak widać z powyższych wzorów, zawsze korzystają z normy $||\mathbf{x}-\mathbf{t}||$. W większości przypadków jest to norma euklidesowa:

$$f(\mathbf{x};\boldsymbol{w},\mathbf{t}) = \sum_{i=1}^{M} w_i G_i(||\mathbf{x}-\mathbf{t}_i||). \tag{3.8}$$

Przyjmując, że liczba wzorców uczących n jest równa liczbie neuronów w warstwie ukrytej M, otrzymujemy:

$$\begin{bmatrix} G_{11} & G_{12} & \cdots & G_{1n} \\ G_{21} & G_{22} & \cdots & G_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ G_{n1} & G_{n2} & \cdots & G_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \tag{3.9}$$

G_{ij} jest równe wartości funkcji G_i w punkcie $\mathbf{x}_j$, $G_{ij} = G_i(||\mathbf{x}_j-\mathbf{t}_i||)$.

Przyjmując, że $\mathbf{G}$, to macierz złożona z wartości G_{ij} i

$$\boldsymbol{w} = [w_1,\ldots,w_n]^T \tag{3.10}$$
$$\mathbf{y} = [y_1,\ldots,y_n]^T \tag{3.11}$$

otrzymujemy

$$\mathbf{G}\boldsymbol{w} = \mathbf{y} \tag{3.12}$$

Twierdzenie Light'a [85, 121] mówi, że istnieje klasa radialnych funkcji bazowych takich, że jeśli tylko wektory $\mathbf{x}_i$ są różne, to macierz $\mathbf{G}$ jest dodatnio określona i możemy wtedy wyznaczyć wektor wag $\boldsymbol{w}$, wymnażając lewostronnie równanie (3.12) przez $\mathbf{G}^{-1}$

$$\boldsymbol{w} = \mathbf{G}^{-1}\mathbf{y} \tag{3.13}$$

Twierdzenie Light'a można stosować między innymi do funkcji gaussowskiej (2.61,3.3) i odwrotnej funkcji potęgowej z $\alpha = 1$ $h_3(\mathbf{x};\mathbf{t},b,1)$ (2.58,3.5).

W praktyce jednak teoretyczna możliwość rozwiązania równania (3.12) nie jest wystarczająca, ponieważ macierz $\mathbf{G}$ może być niedostatecznie dodatnio określona (podobne wzorce uczące), a sieć, która ma tyle samo radialnych funkcji bazowych co wzorców uczących, rzadko może okazać się dobrym rozwiązaniem, ponieważ zazwyczaj będzie to prowadziło do niesatysfakcjonującej generalizacji. Sieć będzie uczyła się nie tylko rozpoznawania wzorców lecz również szumu, który znajdzie się w danych (*ang. overfitting*) [19]. Dlatego też należy wprowadzić czynnik regularyzacyjny do macierzy $\mathbf{G}$, $\mathbf{G} + \lambda\mathbf{I}$, który wynika z metody regularyzacji Tikhonova.

Tikhonov zaproponował metodę regularyzacji, która polega na stabilizacji rozwiązania poprzez dodanie funkcjonału, osadzając w modelu adaptacyjnym dodatkową informację *a priori*. Dla przykładu aproksymacji może to być założenie o gładkości funkcji, która będzie aproksymowała nieznane odwzorowanie. W ten sposób zadanie stanie się dobrze określone i można uniknąć przewymiarowania problemu określonego poprzez układ równań (3.12) [150].

Aby skorzystać z teorii regularyzacji trzeba będzie przedefiniować funkcje oceny błędu kwadratowego procesu minimalizacji

$$E_0(f) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 \tag{3.14}$$

dołączając człon regularyzacyjny

$$E_r(f) = \frac{1}{2}\|\mathbf{P}f\|^2, \tag{3.15}$$

co razem daje

$$\begin{aligned} E(f) &= E_0(f) + \lambda E_r(f) \\ &= \frac{1}{2}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2 + \lambda\frac{1}{2}\|\mathbf{P}f\|^2 \end{aligned} \tag{3.16}$$

Aby wyznaczyć minimum $E(f)$ Poggio i Girosi posługują się różniczką Fréchet'a funkcji błędu, przyrównując ją do zera

$$dE(f,h) = dE_0(f,h) + \lambda dE_r(f,h) \tag{3.17}$$

gdzie $h(\mathbf{x})$ jest pewną funkcją wektora $\mathbf{x}$. To z kolei prowadzi do równania

$$\mathbf{P} * \mathbf{P}f(\mathbf{x}) = \frac{1}{\lambda}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))\ \delta(\mathbf{x} - \mathbf{x}_i) \tag{3.18}$$

Powyższą równość daje się uzyskać dzięki zastosowaniu funkcji Greena, która ma następująca własność

$$\mathbf{P} * \mathbf{P}G_r(\mathbf{x};\mathbf{x}_i) = 0 \tag{3.19}$$

Dzięki temu i kilku innym własnością Poggio i Girosi dostają następujący rezultat

$$f(\boldsymbol{x}) = \frac{1}{\lambda}\sum_{i=1}^{n}[y_i - f(\boldsymbol{x}_i)]G_r(\boldsymbol{x};\boldsymbol{x}_i) \tag{3.20}$$

Równanie (3.20) wyznacza jako rozwiązanie $f(\boldsymbol{x})$ problemu regularyzacji liniową superpozycję n funkcji Greena, położonych w punktach $\boldsymbol{x}_i$ z wagami $w_i = (1/\lambda)[y_i - f(\boldsymbol{x}_i)]$. Mamy więc

$$\boldsymbol{w} = \frac{1}{\lambda}(\boldsymbol{y} - \mathbf{f}) \tag{3.21}$$

$$\mathbf{f} = \mathbf{G}\boldsymbol{w} \tag{3.22}$$

gdzie $\mathbf{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)]^T$, a $\mathbf{G}$ jest macierzą Greena:

$$\mathbf{G} = \begin{bmatrix} G(\boldsymbol{x}_1\boldsymbol{x}_1) & G(\boldsymbol{x}_1\boldsymbol{x}_2) & \ldots & G(\boldsymbol{x}_1\boldsymbol{x}_n) \\ G(\boldsymbol{x}_2\boldsymbol{x}_1) & G(\boldsymbol{x}_2\boldsymbol{x}_2) & \ldots & G(\boldsymbol{x}_2\boldsymbol{x}_n) \\ \vdots & \vdots & & \vdots \\ G(\boldsymbol{x}_n\boldsymbol{x}_1) & G(\boldsymbol{x}_n\boldsymbol{x}_2) & \ldots & G(\boldsymbol{x}_n\boldsymbol{x}_n) \end{bmatrix} \tag{3.23}$$

Stąd otrzymujemy rozwiązanie

$$\boldsymbol{w} = (\mathbf{G} + \lambda\mathbf{I})^{-1}\boldsymbol{y} \tag{3.24}$$

gdzie $\mathbf{I}$ jest macierzą jednostkową.

Poggio i Girosi [150] udowodnili, że sieć neuronowa oparta o funkcje Greena, jako bazowe funkcje radialne, jest uniwersalnym aproksymatorem, czyli może aproksymować dowolną funkcję ciągłą wielu zmiennych na wypukłym podzbiorze w $\mathcal{R}^{\mathcal{N}}$, używając wystarczającej liczby neuronów w warstwie ukrytej. Udowodnili też, że dla dowolnej nieliniowej funkcji F istnieją takie współczynniki w_i, że aproksymują lepiej niż każde inne, co oznacza, iż tak zbudowana sieć ma własność najlepszego aproksymatora. Pokazano też, że rozwiązanie sieci z taką regularyzacją jest optymalne, czyli minimalizuje funkcjonał, który mierzy, jak daleko rozwiązanie jest od danych treningowych.

3.2. Uogólniona sieć z radialnymi funkcjami bazowymi (GRBF)

Powyższa sieć RBF zdaje się jednak w pełni na możliwości regularyzacji i zużywa tyle neuronów ile jest wzorców uczących, co niestety nie zawsze wystarcza. Osiągnięcie wysokiego poziomu aproksymacji (przy pewnych ograniczeniach regularyzacji) to nie to samo, co osiągnięcie wysokiego poziomu generalizacji, choć oczywiście własności te nie pozostają bez związku. Tym samym kierunek zmian powinien iść w stronę uzyskania większego związku pomiędzy złożonością danych uczących, a liczbą węzłów

warstwy ukrytej sieci RBF. Najprostszym sposobem jest oczywiście utworzenie sieci, w której liczba neuronów w warstwie ukrytej będzie znacznie mniejsza, niż liczba wzorców uczących. W pierwszym podejściu liczbę neuronów warstwy ukrytej można wyznaczyć na podstawie pewnej wiedzy a priori o danych uczących. Informacje o różnych podejściach do problemu wyboru liczby jak i położeń neuronów warstwy ukrytej będzie można znaleźć w dalszej części rozdziału.

Można więc napisać równanie sieci RBF

$$f(\mathbf{x}; \boldsymbol{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i). \tag{3.25}$$

z założeniem, że liczba neuronów M jest istotnie mniejsza niż liczba wzorców uczących n. W związku z powyższym, należy przedefiniować funkcję błędu (3.14) na

$$\begin{aligned} E(f) &= \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{M} w_j G(\|\mathbf{x}_i - \mathbf{t}_j\|) \right)^2 + \lambda \|\mathbf{P}f\|^2 \\ &= \|\mathbf{d} - \mathbf{G}\boldsymbol{w}\|^2 + \lambda \|\mathbf{P}f\|^2 \end{aligned} \tag{3.26}$$

gdzie

$$\mathbf{d} = [d_1, d_2, \ldots, d_n]^T \tag{3.27}$$

$$\mathbf{G} = \begin{bmatrix} G(\mathbf{x}_1; \mathbf{t}_1) & G(\mathbf{x}_1; \mathbf{t}_2) & \cdots & G(\mathbf{x}_1; \mathbf{t}_M) \\ G(\mathbf{x}_2; \mathbf{t}_1) & G(\mathbf{x}_2; \mathbf{t}_2) & \cdots & G(\mathbf{x}_2; \mathbf{t}_M) \\ \vdots & \vdots & \ddots & \vdots \\ G(\mathbf{x}_n; \mathbf{t}_1) & G(\mathbf{x}_n; \mathbf{t}_2) & \cdots & G(\mathbf{x}_n; \mathbf{t}_M) \end{bmatrix} \tag{3.28}$$

$$\boldsymbol{w} = [w_1, w_2, \ldots, w_M]^T \tag{3.29}$$

Tak zdefiniowana macierz $\mathbf{G}$ funkcji Greena nie jest już kwadratowa. Ma wymiary n na M, choć sam wektor $\mathbf{d}$ nie zmienił rozmiarów. Wektor $\boldsymbol{w}$ ma rozmiar M, lecz M jest różne od n. Jak pokazują Poggio i Girosi [150] prawą stronę wyrażenia (3.26) można rozwinąć do

$$\|\mathbf{P}f\|^2 = \boldsymbol{w}^T \mathbf{G}_0 \boldsymbol{w} \tag{3.30}$$

gdzie macierz $\mathbf{G}_0$ jest macierzą kwadratową M na M, zdefiniowaną jako

$$\mathbf{G}_0 = \begin{bmatrix} G(\mathbf{t}_1; \mathbf{t}_1) & G(\mathbf{t}_1; \mathbf{t}_2) & \cdots & G(\mathbf{t}_1; \mathbf{t}_M) \\ G(\mathbf{t}_2; \mathbf{t}_1) & G(\mathbf{t}_2; \mathbf{t}_2) & \cdots & G(\mathbf{t}_2; \mathbf{t}_M) \\ \vdots & \vdots & \ddots & \vdots \\ G(\mathbf{t}_M; \mathbf{t}_1) & G(\mathbf{t}_M; \mathbf{t}_2) & \cdots & G(\mathbf{t}_M; \mathbf{t}_M) \end{bmatrix} \tag{3.31}$$

Można powrócić do minimalizacji równania (3.26) i wyznaczyć pochodną

$$\begin{aligned} dE(f) &= d\|\mathbf{d} - \mathbf{G}\boldsymbol{w}\|^2 + d\lambda \|\mathbf{P}f\|^2 \\ &= \mathbf{G}^T(\mathbf{d} - \mathbf{G}\boldsymbol{w}) - \lambda \mathbf{G}_0 \boldsymbol{w} \end{aligned} \tag{3.32}$$

i przyrównać ją do zera

$$\mathbf{G}^{\mathrm{T}}\mathbf{d} - (\mathbf{G}^{\mathrm{T}}\mathbf{G} - \lambda\mathbf{G}_0)\boldsymbol{w} = 0 \tag{3.33}$$

skąd osiągamy już szukany wektor wag $\boldsymbol{w}$

$$\boldsymbol{w} = (\mathbf{G}^{\mathrm{T}}\mathbf{G} - \lambda\mathbf{G}_0)^{-1}\mathbf{G}^{\mathrm{T}}\mathbf{d} \tag{3.34}$$

Dla $\lambda = 0$ rozwiązaniem minimalizacji jest iloczyn macierzy pseudoodwrotnej z wektorem wartości oczekiwanych $\mathbf{d}$

$$\boldsymbol{w} = (\mathbf{G}^{\mathrm{T}}\mathbf{G})^{-1}\mathbf{G}^{\mathrm{T}}\mathbf{d} \tag{3.35}$$

3.3. Metody inicjalizacji i uczenia bez nadzoru sieci typu RBF

W poprzedniej części zaprezentowano metody regularyzacji dla sieci z radialnymi funkcjami bazowymi. Pozostają jednak jeszcze pewne pytania. Na przykład, jak dokonać wyboru położeń funkcji bazowych w metodzie opisanej w podrozdziale 3.2, czy też jak dobrać parametry rozmyć dla funkcji bazowych (i być może wartości innych parametrów, zależnie od wyboru funkcji bazowych). W tym podrozdziale zostaną omówione problemy inicjalizacji i beznadzorowego uczenia sieci RBF i różnych innych sieci zbliżonych do sieci RBF.

Jak widać z równania (3.2), jak i rys. 3.1, warstwy sieci RBF spełniają różne role. Pierwsza warstwa lokalizuje węzły warstwy ukrytej w przestrzeni wejściowej, pokrywając ją w okolicach obszarów, w których występują dane. Zadaniem drugiej warstwy jest wyznaczenie takiej liniowej kombinacji aktywacji neuronów warstwy ukrytej, aby estymowała ona nieznane odwzorowanie $\mathrm{F}(\cdot)$ równania (3.1). Stąd też płynie natura uczenia sieci z radialnymi funkcjami bazowymi, która przeplata uczenie z nadzorem i uczenie bez nadzoru. Inicjalizacja pierwszej warstwy odbywa się bez nadzoru (patrz poniższe metody), natomiast inicjalizacja drugiej warstwy, jak i sam proces uczenia przebiegają już z nadzorem (patrz kolejny podrozdział 3.4 i rozdział 4).

Ważną cechą sieci o radialnych funkcjach bazowych jest to, że wagi pierwszej warstwy mogą zostać wyznaczone za pomocą wiedzy a priori o samych danych wejściowych. Dzięki wykorzystaniu informacji o danych wejściowych proces uczenia nie będzie startował z losowego punktu przestrzeni wag i ma znacznie większe szanse na pomyślny przebieg procesu uczenia sieci.

3.3.1. Inicjalizacja położeń zbiorem wektorów uczących

Najprostszy sposób wyznaczenia liczby i położeń węzłów warstwy ukrytej został już opisany w podrozdziale 3.1, w którym warstwa ukryta sieci składa się z tylu

węzłów, ile jest wektorów uczących, a ich położenia wyznaczają wektory uczące. Trzeba jednak pamiętać, iż taka sieć powinna być wyposażona w regularyzacje, w przeciwnym przypadku może dojść do odwzorowywania szumu jaki może istnieć w realnych danych.

3.3.2. Inicjalizacja położeń poprzez podzbiór zbioru uczącego

Jednym z pierwszych pomysłów było wybranie pewnej liczby M wektorów uczących i zainicjowanie ich położeniami położeń węzłów ukrytych zgodnie z rozkładem danych [19]. Takie zainicjowanie położeń, choć wydaje się bardzo proste, daje naprawdę dobre rezultaty, dzięki skupianiu węzłów w obszarach głównej aktywności danych. Widać stąd, iż błędne dane spoza właściwych obszarów, mają statystycznie nieistotny wpływ na późniejszy proces uczenia sieci. Spotyka się też propozycje nieznacznego losowego zaburzenia tak ustalonych położeń węzłów ukrytych, aby zadanie minimalizacji było lepiej określone [94]. W przypadku gdy funkcjami transferu mają być funkcje Gaussa pozostaje jeszcze dobór parametrów rozmyć. David Lowe [122] zaproponował, by uzależnić rozmycie od liczby węzłów warstwy ukrytej i maksymalnej odległości pomiędzy nimi:

$$G(\|\mathbf{x} - \mathbf{t}_i\|^2) = \exp\left(-\frac{M}{d^2}\|\mathbf{x} - \mathbf{t}_i\|^2\right), \quad i = 1, 2, \ldots, M \tag{3.36}$$

M określa liczbę neuronów warstwy ukrytej, a d jest maksymalną odległością pomiędzy wszystkimi wybranymi węzłami (centrum funkcji gaussowskiej jest położone w punkcie $\mathbf{t}_i$). Tym samym szerokość każdego z neuronów to

$$\frac{d}{\sqrt{2M}} \tag{3.37}$$

3.3.3. Inicjalizacja położeń metodą klasteryzacji k-średnich

Oczywiście nie ma powodu, aby prawdą było, iż losowy wybór wektorów wejściowych na położenia centrów, powiedzmy funkcji gaussowskich, neuronów ukrytych dla sieci RBF miało być optymalne. Fakt ten przyczynił się do powstania szeregu różnych, mniej i bardziej popularnych, sposobów inicjalizacji położeń centrów neuronów ukrytych. Jedną z najszerzej stosowanych metod jest metoda klasteryzacji k-średnich [51]. Jako pierwsi metodę k-średnich do sieci RBF użyli Moody i Darken [135]. Metoda ta stara się umiejscowić docelowo położenia centrów neuronów ukrytych w obszarach przestrzeni, w których znajdują się najbardziej znaczące dane.

Metoda k-średnich cel ten stara się osiągnąć poprzez minimalizację następującej funkcji

$$kmean_{err} = \sum_{i=1}^{M} \sum_{j \in \mathcal{S}_i} \|\mathbf{x}_j - \mathbf{t}_i\|^2 \tag{3.38}$$

gdzie $\mathcal{S}_i$ to zbiór indeksów wektorów przynależnych do klastra i, a $\mathbf{t}_i$ stanowi centrum i-tego klastra.

W pierwszym etapie następuje ustalenie liczby klastrów i tym samym liczby neuronów M w warstwie ukrytej. Następnie, w losowy sposób wybieramy wstępne położenia klastrów spośród wektorów wejściowych, po czym w procesie iteracyjnym dla każdego wektora $\mathbf{x}_j$ następuje uaktualnienie położenia najbliższego z centrów klastrów $\mathbf{t}_i$:

$$\Delta \mathbf{t}_i = \eta(\mathbf{x}_j - \mathbf{t}_i) \tag{3.39}$$

η jest parametrem określającym szybkość uczenia.

Znana jest też wersja *off-line* metody k-średnich [10]. Pierwsza z różnic między tą wersją, a opisaną wyżej polega na tym, iż centra początkowo wybierane są w losowych położeniach w przestrzeni wejściowej i następuje przypisanie każdego z wektorów do najbliższego klastra. Z kolei położenia centrów przeliczane są co epokę (po prezentacji całego zbioru uczącego):

$$\mathbf{t}_i = \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} \mathbf{x}_j \tag{3.40}$$

gdzie $\mathcal{S}_i$ to zbiór indeksów wektorów przynależnych do klastra i, a $|\mathcal{S}_i|$ to liczba przynależnych wektorów do klastra i. Następnie dokonuje się powtórnego przypisania wszystkich wektorów do najbliższych klastrów.

Najczęściej stosowaną wersją algorytmu k-średnich jest pierwsza z podanych, choć oczywiście obie wersje algorytmu nie gwarantują optymalnego rozwiązania minimalizacji równania 3.38. Przy złych warunkach startowych, algorytm może nie umieścić żadnego klastra (centrum) w miejscach, które mogą być istotne, a w których danych jest zbyt mało (rzadko opisują daną część przestrzeni wejściowej). Zapobiec (choć raczej częściowo) takim problemom można poprzez uczynienie parametru η zmiennym w czasie procesu minimalizacji:

$$\eta = \frac{\eta_0}{1 + \frac{i}{T}} \tag{3.41}$$

η_0 jest wartością początkową procesu, natomiast w kolejnych iteracjach i wraz z *a priori* wyznaczonym współczynnikiem T (dopasowanym dla konkretnych danych) wartość η będzie zmniejszana, najpierw niemal nieznacznie, a później coraz szybciej.

Metoda k-średnich jest też pewnym szczególnym przypadkiem metody mieszania modeli gaussowskich (*ang. Gaussian mixture models*), w której używa się do optymalizacji algorytmu EM (*ang. expectation maximization*) [10]. Również i algorytm samoorganizujących się map topograficznych Kohonena (SOFM) [112, 113] jest ogólnym algorytmem uczenia, dla którego metoda k-średnich jest szczególnym przypadkiem.

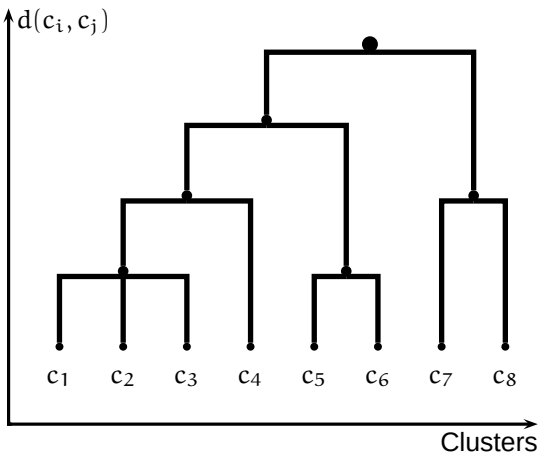
Alternatywną metodą klasteryzacji dla metody k-średnich może też być metoda klasteryzacji maksymalnej entropi [21] przy założeniu, iż przypisania do klastrów mają rozkład Gibbs'a.

3.3.4. Inicjalizacja za pomocą metody k najbliższych sąsiadów

Ta metoda również jest pewną odmianą metod samoorganizacji i stara się wyznaczyć położenia centrów funkcji radialnych tak, by znalazły się one w miejscach gdzie znajdują się najbardziej reprezentatywne wektory danych.

Do tego celu Moody i Darken [135] wykorzystali, w nieco niestandardowy sposób, metodę k najbliższych sąsiadów (kNN) [51]. Jako centra zostają wybrane te wektory wejściowe, które najczęściej są wybierane jako k najbliższe. Ta metoda, podobnie jak metoda k-średnich, pozwala wybrać położenia centrów neuronów w bardziej deterministyczny sposób, niż losowy wybór k centrów.

3.3.5. Konstruowanie klastrów za pomocą dendrogramów



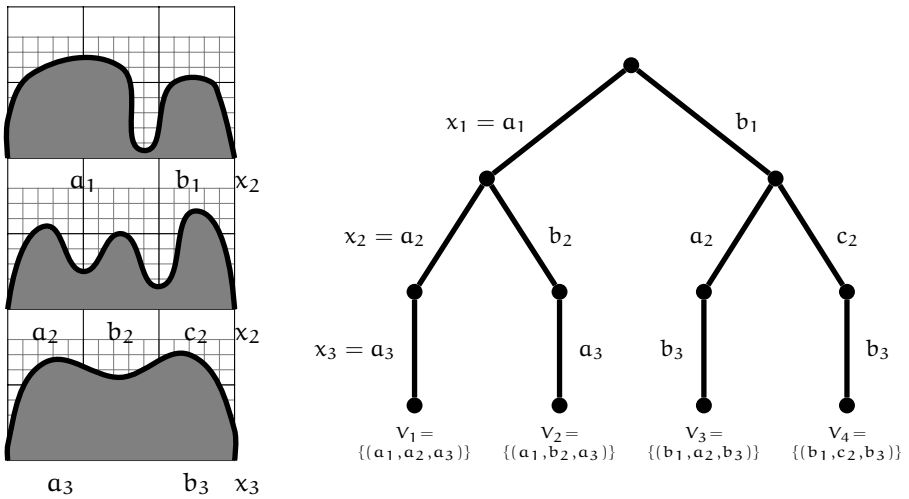
Rysunek 3.2: Dendrogramy.

Inną ciekawą metodą inicjalizacji położeń centrów, jak i ich rozmyć, które w tym przypadku mogą być różne dla różnych wymiarów, jest metoda dendrogramów. Dzięki tej metodzie można uzyskać ciekawsze rezultaty niż innymi metodami inicjalizacji [42]. Początkowo każdy z wektorów treningowych tworzy odrębny klaster. Po czym, w procesie iteracyjnym następuje łączenie najbliższych klastrów na podstawie pewnej miary odległości (patrz rys. 3.2). Taka procedura jest powtarzana do momentu aż liczba powstałych klastrów jest wystarczająco mała lub najmniejsza odległość w pewnej iteracji okaże się za duża, aby dokonywać dalszych połączeń.

Takie postępowanie wymaga jednak wstępnie wyznaczenia tablicy odległości pomiędzy każdą parą wektorów treningowych ($N(N-1)$ odległości), a następnie w każdej iteracji. Po dokonaniu połączenia odległości pomiędzy połączonymi klastrami a innymi klastrami stają się niepotrzebne. Z kolei należy wyznaczyć odległości pomiędzy

nowym klastrem i pozostałymi klastrami i tym samym tablica odległości staje się aktualna. Bardzo ważną cechą tej metody jest to, że możemy użyć dowolnej funkcji odległości. Funkcja odległości może być zdefiniowana nie tylko jako odległość pomiędzy centrami klastrów, ale na przykład, jako minimalna odległość pomiędzy brzegami dwóch rozpatrywanych klastrów. Należy zwrócić uwagę, że podczas łączenia klastrów dokonuje się łączenia obszarów klastrów, co również może być różnie realizowane i jest bardzo zależne od przyjętej miary odległości i wstępnej obróbki danych.

3.3.6. Inicjalizacja za pomocą histogramów i drzew decyzyjnych



Rysunek 3.3: Histogramy.

Metoda dendrogramów jest pod pewnym względem, odwrotnością metod opartych o analizę histogramów lub drzew klasyfikacyjnych (zwanych też drzewami decyzji). Odwrotność ta polega na tym, iż w metodzie dendrogramów opis danych jest z iteracji na iterację coraz ogólniejszy, natomiast drzewa klasyfikacyjne i metody oparte o histogramy lub inne podziały poszczególnych wymiarów, coraz bardziej uściślają opis danych uczących.

Pierwszym etapem tej metody jest wyznaczenie punktów w oparciu o metodę histogramów lub pewien arbitralny podział poszczególnych wymiarów. W przypadku histogramów taka dyskretyzacja uwidacznia gęstości występowania poszczególnych klas lub klastrów danych w wyznaczonych przedziałach, które tworzą klastry w jednowymiarowej przestrzeni. Histogramy informują o punktach w danym wymiarze, w których dane zmieniają przynależność do klasy lub klastra, czyli o potencjalnych granicach klas lub klastrów.

Tak wyznaczone punkty definiują nam granice wstępnych klastrów w jednowymiarowej podprzestrzeni przestrzeni d wymiarowej danych (patrz rys. 3.3). Każdy z

jednowymiarowych klastrów jest rzutem pewnej liczby niezależnych klastrów w d wymiarowej przestrzeni. Każdy z klastrów w d wymiarowej przestrzeni przynależy do dokładnie jednego klastra w jednowymiarowej przestrzeni (jest zdefiniowany poprzez ciąg klastrów w poszczególnych wymiarach). W ten sposób, z histogramów powstaje drzewo klasyfikacyjne po uszeregowaniu wymiarów w ciąg (patrz rys. 3.3). Z każdym klastrem można związać informacje o liczbie wektorów, którą reprezentuje. Najczęściej nazywa się to masą klastra [42].

Jeśli liczba powstałych klastrów jest zbyt duża, szczególnie gdy dokonuje się pewnych arbitralnych podziałów w poszczególnych wymiarach, można wtedy (a nawet należy), po powyżej opisanym etapie, dokonać próby łączenia klastrów. Często zdarza się, iż w jakimś wymiarze możemy mieć do czynienia z nieefektywnym podziałem — zbyt szczegółowym — i tym samym nie wnoszącym żadnych korzyści, a z drugiej strony zwiększającym liczbę klastrów. W takich przypadkach na pewno część klastrów da się połączyć i w ten sposób otrzymujemy prostszy opis danych uczących. Tym samym zmniejszamy złożoność wstępną modelu. Etap ten można zrealizować za pomocą metody dendrogamowej, w której zostaną połączone najbliższe klastry, zgodnie z odpowiednio dobraną miarą odległości (patrz podrozdział 3.3.5) lub poprzez przeszukiwanie powstałego drzewa decyzyjnego [42] i łączenie sąsiadujących klastrów.

Bywają też sytuacje w których same histogramy nie są wystarczające. Na przykład dane o rozkładzie przedstawionym na rys. 3.4 — punkty podziałów, które wynikają z analizy histogramów, nie prowadzą do efektywnego podziału przestrzeni danych i tym samym powstałe klastry nie przynoszą pożądanych efektów.

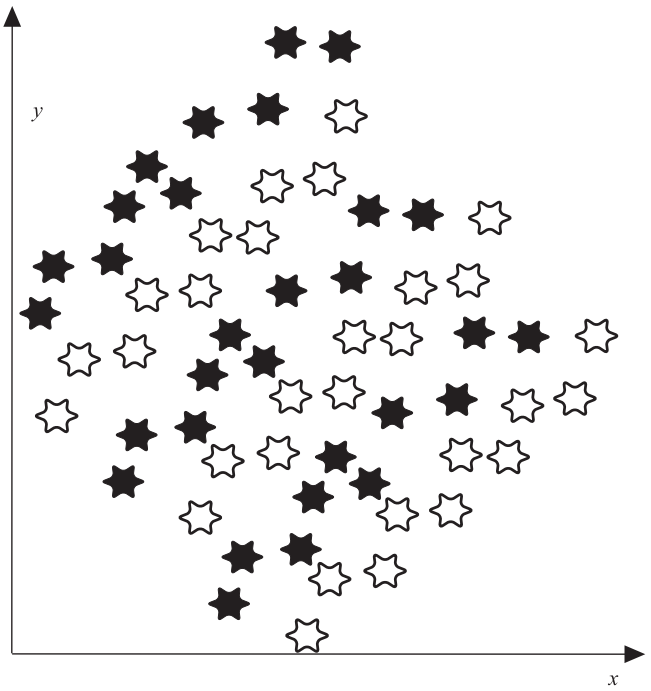
W końcowym etapie, podobnie, jak dla metody k-średnich, następuje wyznaczenie rozmyć (szerokości) poszczególnych centrów funkcji radialnych. Należy tu wspomnieć iż warto w takiej metodzie inicjalizacji skorzystać z wielowymiarowych funkcji gaussowskich lub funkcji bicentralnych i ich rozszerzeń, które umożliwiają używanie różnych rozmyć w różnych wymiarach, co prowadzi do optymalniejszego wykorzystania informacji z klasteryzacji.

Bardzo ciekawym jest algorytm Breimana, który tworzy drzewo klasyfikacji i regresji (CART) (*ang. classification and regression tree*) [17], z powodzeniem stosowany w klasyfikacji, jak i do wyciągania reguł logicznych z danych. Algorytm ten może być również użyty do inicjalizacji sieci typu RBF. CART w każdej iteracji procesu tworzenia drzewa stara się znajdować optymalny punkt w jednym z wymiarów danych, tak, aby za pomocą jak najmniejszej liczby podziałów przestrzeni zbudować drzewo klasyfikacji. Oceny, czy w jakimś punkcie dokonać podziału, czy nie, dokonuje się na podstawie funkcji kosztu i estymacji prawdopodobieństw danego węzła w rejonie $\mathbf{R}$:

$$p(t) = \frac{n(t)}{n} \tag{3.42}$$

$$p(j|t) = \frac{n_j(t)}{n(t)} \tag{3.43}$$

n jest liczbą wektorów trenujących, $n(t)$ jest liczbą wektorów trenujących w rejonie $\mathbf{R}$. Z kolei $n_j(t)$ jest liczbą wektorów klasy j w rejonie $\mathbf{R}$. Funkcję kosztu można



Rysunek 3.4: Gęstość, dla której analiza histogramów nie daje żadnych korzyści.

wtedy zdefiniować na kilka sposobów:

$$Q(t) = 1 - \max_j p(j|t) \tag{3.44}$$

$$Q(t) = \sum_j \sum_{i \neq j} p(i|t)p(j|t) = 1 - \sum_j [p(j|t)]^2 \tag{3.45}$$

$$Q(t) = -\sum_j p(j|t) \ln p(j|t) \tag{3.46}$$

Efektywność pierwszej funkcji jest najniższa. Może ona informować o niemożności zmniejszenia funkcji kosztu, co najczęściej nie jest prawdą. Dwie pozostałe funkcje dają dość porównywalne i dobre wyniki.

Teraz, gdy chcemy w węźle t dokonać podziału na dwa węzły t_L i t_R, w punkcie v trzeba móc ocenić zmniejszenie *zanieczyszczenia* poprzez ten podział:

$$\Delta Q(v, k, t) = Q(t) - Q(t_L)p_L(t) - Q(t_R)p_R(t) \tag{3.47}$$

gdzie $p_L(t)$ i $p_R(t)$ są zdefiniowane przez:

$$p_L(t) = \frac{p(t_L)}{p(t)} \tag{3.48}$$

$$p_R(t) = \frac{p(t_R)}{p(t)} \tag{3.49}$$

Rekurencyjny proces podziałów przebiega aż do spełnienia pewnego warunku stopu, który zazwyczaj jest określony przez osiągnięcie pewnego progu klasyfikacji. Proces tworzenia drzewa klasyfikacji może czasami prowadzić do jego przerośnięcia, a wtedy część węzłów odgrywa mało istotną rolę w procesie klasyfikacji. CART w tym momencie uruchamia procedurę usuwania zbędnych węzłów, a procedura minimalizuje karę za ryzyko:

$$R_{pen} = R_{emp} + \lambda|T| \tag{3.50}$$

R_{emp} jest współczynnikiem błędu klasyfikacji dla danych treningowych, $|T|$ jest liczbą węzłów–liści w drzewie. Optymalna wartość parametru λ jest wyznaczana poprzez minimalizację błędu klasyfikacji dla różnych podzbiorów zbioru uczącego.

Do podobnego schematu inicjalizacji można wykorzystać kryterium dipolowe [14, 12, 13], jak i inną metodę zbliżoną do metody CART [78].

3.4. Uczenie z nadzorem sieci RBF

Ten podrozdział omawia najbardziej typowy sposób uczenia sieci z radialnymi funkcjami bazowymi. Metoda ta bazuje na wcześniej ustalonej architekturze sieci, w której przede wszystkim została ustalona liczba węzłów ukrytych i typ funkcji transferu realizowany przez te węzły. Zakłada się również, iż dokonano wstępnej inicjalizacji sieci RBF, na przykład za pomocą jednej z metod inicjalizacji przedstawionych w podrozdziale 3.3. Prezentowana tu metoda uczenia z nadzorem polega na iteracyjnym wprowadzeniu korekcji, wyznaczanych poprzez wyliczanie kierunku gradientu funkcji błędu. Poggio i Girosi [149] proponują, by uczeniu podlegały wagi wyjściowe i położenia centrów. Takie sieci nazywa się uogólnionymi sieciami RBF (GRBF) (*generalized radial bases networks*). Z kolei Lowe [122] proponuje, by wszystkie parametry były adaptowane, czyli wagi, położenia centrów i rozmycia funkcji. Same poniższe równania, opisujące algorytm uczenia, są bardzo podobne do równań dla wyznaczonych dla algorytmu LMS (*ang. least mean square*).

Punktem wyjścia jest zdefiniowanie funkcji błędu, którą następnie algorytm będzie starał się minimalizować w procesie uczenia:

$$E = \frac{1}{2}\sum_{i=1}^{n} e_i^2 \tag{3.51}$$

gdzie n jest liczbą wektorów uczących, a e_i jest błędem, jaki popełnia sieć dla i-tego wektora uczącego:

$$e_i = y_i - f(\mathbf{x}_i) = y_i - \sum_{j=1}^{M} w_j G_j(\|\mathbf{x}_i - \mathbf{t}_j\|) \tag{3.52}$$

y_j jest oczekiwaną wartością wyjściową, a $f(\mathbf{x}_i)$ wartością zwracaną przez sieć.

Warto zwrócić uwagę, iż funkcja kosztu określona równaniem (3.51), jest wypukła ze względu na parametry wag w_i, ale nie jest wypukła ze względu na parametry

położeń centrów funkcji transferu i ich rozmyć, co może być przyczyną utykania procesu minimalizacji w lokalnych minimach przestrzeni parametrów.

Proces minimalizacji opisują poniższe równania, które wyznaczają gradienty dla poszczególnych parametrów adaptacyjnych:

$$w_i(n+1) = w_i(n) - \eta_w \frac{\partial E(n)}{\partial w_i(n)} \tag{3.53}$$

$$\mathbf{t}_i(n+1) = \mathbf{t}_i(n) - \eta_t \frac{\partial E(n)}{\partial \mathbf{t}_i(n)} \tag{3.54}$$

$$b_i(n+1) = b_i(n) - \eta_b \frac{\partial E(n)}{\partial b_i(n)} \tag{3.55}$$

n oznacza numer bieżącej iteracji algorytmu minimalizacji, η_w, η_t, η_b są współczynnikami szybkości adaptacji parametrów wag, położeń centrów i ich rozmyć.

Jeśli funkcją transferu jest funkcja gaussowska

$$g(\mathbf{x}; \mathbf{t}, b) = e^{-||\mathbf{x}-\mathbf{t}||^2/b^2} \tag{3.56}$$

to poszczególne gradienty są równe:

$$\frac{\partial E(n)}{\partial w_i(n)} = \sum_{j=1}^{n} e_j(n)\ g(\mathbf{x}_j; \mathbf{t}_i(n), b_i(n)) \tag{3.57}$$

$$\frac{\partial E(n)}{\partial \mathbf{t}_i(n)} = \frac{2w_i(n)}{b_i^2(n)} \sum_{j=1}^{n} e_j(n)\ g(\mathbf{x}_j; \mathbf{t}_i(n), b_i(n))\ [\mathbf{x}_j - \mathbf{t}_i(n)] \tag{3.58}$$

$$\frac{\partial E(n)}{\partial b_i(n)} = \frac{2w_i(n)}{b_i^3(n)} \sum_{j=1}^{n} e_j(n)\ g(\mathbf{x}_j; \mathbf{t}_i(n), b_i(n))\ ||\mathbf{x}_j - \mathbf{t}_i(n)||^2 \tag{3.59}$$

Przedstawiona powyżej procedura minimalizacji nazywana jest często metodą *off-line*. Bierze się to stąd, iż poprawki związane z funkcją błędu dla wektorów testowych są nanoszone po prezentacji całego zbioru uczącego, co jest oczywiście konsekwencją ujęcia całego zbioru treningowego w definicji funkcji błędu (3.51). Sieci typu RBF i podobne można też uczyć w trybie *on-line*, co oznacza, iż celem jest bezpośrednie nanoszenie poprawek po prezentacji każdego wektora uczącego. Wtedy funkcja błędu wygląda tak:

$$E = \frac{1}{2}|e_i|^2 \qquad i = 1, 2, \ldots, n \tag{3.60}$$

Adaptacja parametrów, podobnie jak i przedtem, następuje zgodnie z równaniami (3.53, 3.54, 3.55), natomiast zmianie ulegają gradienty liczone dla poszczególnych

parametrów adaptacyjnych:

$$\frac{\partial E(n)}{\partial w_i(n)} = e(n)\ g(\mathbf{x}(n); \mathbf{t}_i(n), b_i(n)) \tag{3.61}$$

$$\frac{\partial E(n)}{\partial \mathbf{t}_i(n)} = 2w_i(n)e(n)\ g(\mathbf{x}(n); \mathbf{t}_i(n), b_i(n))\ \frac{\mathbf{x}(n) - \mathbf{t}_i(n)}{b_i^2(n)} \tag{3.62}$$

$$\frac{\partial E(n)}{\partial b_i(n)} = 2w_i(n)e(n)\ g(\mathbf{x}(n); \mathbf{t}_i(n), b_i(n))\ \frac{\|\mathbf{x}(n) - \mathbf{t}_i(n)\|^2}{b_i^3(n)} \tag{3.63}$$

$\mathbf{x}(n)$ określa wektor treningowy, prezentowany w n-tej iteracji procesu minimalizacji, a $e(n)$ błąd, jaki popełniła dla tego wektora sieć jest równy różnicy $y_i(n) - f(\mathbf{x}(n))$.

W praktyce uczenie *off-line* jest wolniejsze i stabilniejsze, a uczenie *on-line* szybsze, ale mniej stabilne. Metoda *on-line* jest też najbardziej podobna do algorytmu LMS.

Wettschereck i Dieterich [174] porównywali działanie sieci RBF z i bez adaptacji centrów z sieciami MLP, uczonymi algorytmem wstecznej propagacji (BP). Z pracy wynika, że sieci RBF, w których nie dokonuje się jedynie adaptacji wag wyjściowych, generalizują gorzej niż sieci MLP, ale sieci RBF, w których dokonuje się adaptacji położeń centrów funkcji transferu, generalizują lepiej niż sieci BP.

3.5. Rozszerzenia sieci RBF

Sieci z radialnymi funkcjami bazowymi, podobnie jak i inne modele sieci neuronowych, doczekały się licznych rozszerzeń. Nie sposób wymienić wszystkie z nich, ale mam nadzieję wspomnieć te najciekawsze i częściej spotykane. Bazując już na tym, co do tej pory przedstawiono w tym rozdziale, można skonstruować bardzo wiele różnych sieci. Mogły by się one różnić metodami inicjalizacji i/lub dalszym postępowaniem. Dołączając do tego możliwość skorzystania z różnych funkcji transferu, przedstawionych w rozdziale 2, klasa możliwych sieci neuronowych coraz bardziej się rozszerza i zmienia swoje własności.

3.5.1. Rozszerzenia głównego równania sieci RBF

Dość standardowymi już rozszerzeniami są różne rozszerzenia głównego równania sieci RBF (3.2), czyli liniowej kombinacji aktywacji funkcji transferu. Pierwszym przykładem niech będzie prosty, choć bardzo użyteczny dodatek w postaci współczynnika adaptacyjnego w_0

$$f(\mathbf{x}; \boldsymbol{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i) + w_0 \tag{3.64}$$

którego celem jest uwolnienie samej sieci od średniej wartości wyjścia (por. [10] rozdział 3.4.3). Częściej spotyka się bardziej radykalne rozszerzenia takie jak

$$f(\mathbf{x};\boldsymbol{w},\mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x},\mathbf{p}_i) + \sum_{i=1}^{m} d_i p(\mathbf{x}) \qquad m \leqslant N \tag{3.65}$$

gdzie $p(\mathbf{x})$ jest wielomianem pewnego stopnia k z przestrzeni $\mathcal{R}^N$ [149].

Do typowych rozszerzeń zalicza się również użycie macierzy wag $\mathbf{C}$ w normie $\|\cdot\|_{\mathbf{C}}$ [150]

$$f(\mathbf{x};\boldsymbol{w},\mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\|\mathbf{x}-\mathbf{t}_i\|_{\mathbf{C}_i}) \tag{3.66}$$

gdzie $\|\cdot\|_{\mathbf{C}}$ jest zdefiniowana jako

$$\|\mathbf{x}-\mathbf{t}_i\|_{\mathbf{C}_i} = (\mathbf{x}-\mathbf{t}_i)^T \mathbf{C}_i^T \mathbf{C}_i (\mathbf{x}-\mathbf{t}_i) \tag{3.67}$$

Jednakże taka macierz jest trudna do adaptacji ze względu na liczbę parametrów adaptacyjnych, która rośnie kwadratowo z rozmiarem przestrzeni wejściowej.

Inne, ciekawe i dość proste rozszerzenie, to użycie niejednorodnych miar odległości [175], które zostały już opisane w rozdziale 2.2.1.

3.5.2. Regularyzacja

Niezwykle ważną częścią rozważań nad sieciami RBF (jak i innymi sieciami), jest regularyzacja tych modeli, która w dużym stopniu wpływa na stabilizację procesu uczenia sieci i jest głównym narzędziem wspomagającym uzyskanie możliwie maksymalnej generalizacji i tym samym pozwala unikać przeuczenia się podczas adaptacji. Regularyzacja sieci RBF była już wspomniana na początku rozdziału, jednakże opis ten nie wyczerpał całego tematu, szczególnie różnych innych, ciekawych podejść do regularyzacji.

Najczęściej regularyzacja sprowadza się do dodania pewnego czynnika do funkcji błędu modelu $E_0(f)$ (3.14). Należy wspomnieć, że można jako podstawowego członu funkcji błędu używać nie tylko funkcji $E_0(f)$, ale również jej ogólniejszej formy w postaci funkcji błędu Minkowskiego, przechodząc do normy L_R:

$$E_M(f;R) = \sum_{i=1}^{n} |y_i - f(\mathbf{x}_i)|^R \tag{3.68}$$

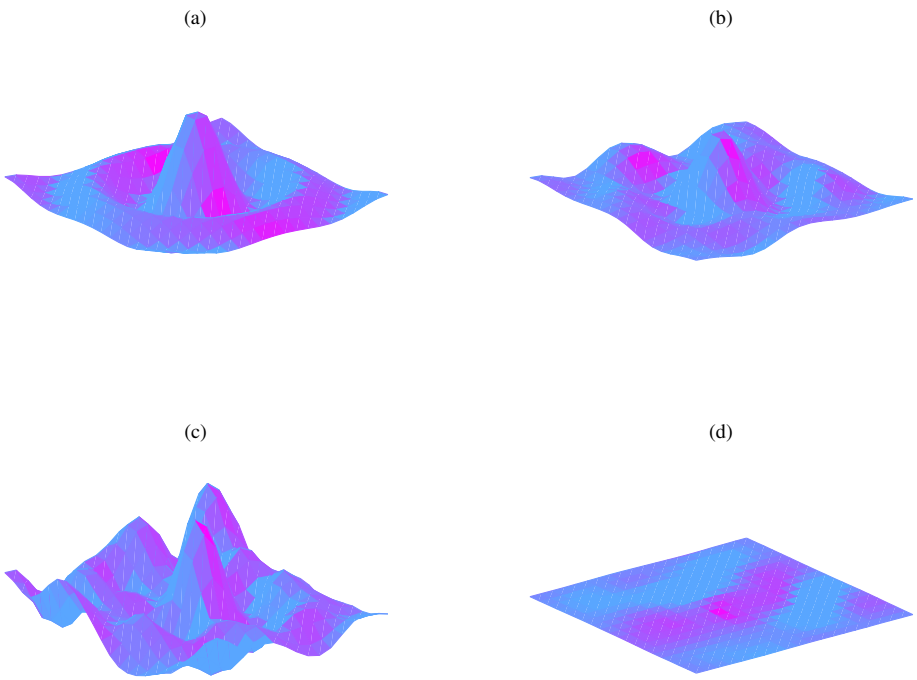
w szczególnym przypadku dla $R = 2$ mamy tożsamość z funkcją $E_0(f)$, czyli normę L_2, a dla $R = 1$ mamy metrykę Manhattan.

Jednym z najbardziej znanych czynników regularyzacyjnych jest rozpad wag (*ang. weight decay*). Wtedy do miary błędu modelu $E_0(f)$ (3.14) zostaje dodany czynnik

regularyzacyjny:

$$E_{wd}(f, \boldsymbol{w}) = E_0(f) + \lambda \sum_{i=1}^{M} w_i^2 \tag{3.69}$$

W aproksymacji i statystyce ten typ regularyzacji nazywany jest regresją grzbietową (*ang. ridge regression*). Uwzględnienie takiego czynnika w funkcji błędu znacznie poprawia uzyskiwane wyniki [87]. Breiman [16] twierdzi, iż taka regularyzacja sprawia, że proces uczenia jest stabilny, natomiast nie jest tak gdy do wyznaczania niektórych parametrów uczenia stosuje się techniki uczenia na podzbiorach (patrz też niżej). Przykład zastosowania regularyzacji można zobaczyć na rysunku 3.5.



Rysunek 3.5: Zastosowanie regularyzacji do aproksymacji funkcji $10\frac{\sin(|xy|)}{|xy|}$. Rysunek a) pokazuje oryginalną funkcję. Kolejne rysunki pokazują aproksymację z regularyzacją dla b) $\lambda = 1$, c) $\lambda = 10^{-4}$, d) $\lambda = 100$. Patrz równanie (3.69).

Lokalna regresja grzbietowa (*ang. local ridge regression*) jest uogólnieniem poprzedniej wersji regularyzacji:

$$E_{lrr}(f, \boldsymbol{w}) = E_0(f) + \sum_{i=1}^{M} \lambda_i w_i^2 \tag{3.70}$$

W przypadku takiej regresji dla lokalnych funkcji, takich, jak większość funkcji RBF, gładkość regresji nie jest kontrolowana jednym współczynnikiem, lecz każda z funkcji

jest kontrolowana niezależnie. To prowadzi do lokalnej adaptacji gładkości w zależności od stanu w lokalnej części przestrzeni. Regresja nielokalna dla problemów, w których gładkość funkcji w różnych częściach przestrzeni powinna być różna, często nie daje pożądanych rezultatów [143]. Do wyznaczania parametrów regularyzacyjnych stosuje się często uczenie poprzez kroswalidację (*ang. cross-validation*, co można przetłumaczyć jako *krzyżową wiarygodność*, ale nie ma powszechnie przyjętego terminu w języku polski) [143, 77] i różne ich odmiany. Metoda polega na podziale zbioru uczącego na k części, a następnie usuwaniu po jednej z nich, uczeniu sieci na pozostałych i testowaniu na usuniętej części. Zebrane informacje o jakości klasyfikacji lub aproksymacji na kolejno usuwanych częściach zbioru uczącego dają obraz działania algorytmu dla wcześniej ustalonych parametrów regresji (oczywiście metodę tę można wykorzystywać do wyznaczania i innych parametrów, jak i innych modeli adaptacyjnych).

Bishop w [8] zaproponował jeszcze inny człon regularyzacyjny:

$$E_{r2} = E_0(f) + \frac{1}{2}\eta^2 \sum_n \sum_i \frac{1}{f(\mathbf{x}_n)(1 - f(\mathbf{x}_n))} \left(\frac{\partial f(\mathbf{x}_n)}{\partial x_{n,i}} \right)^2 \tag{3.71}$$

Powyższy człon regularyzacyjny nie wymaga aby funkcjami bazowymi równania sieci RBF (3.2) była pewna funkcja Green'a. Nie wymaga się również, aby liczba funkcji bazowych była równa liczbie wektorów zbioru treningowego (porównaj podrozdział 3.1).

Bardzo ciekawym wynikiem było udowodnienie przez Bishopa, iż uczenie z regularyzacją Tikhonova jest równoważne uczeniu z szumem [9]. Poprzez uczenie z szumem rozumie się dodanie losowego szumu do wejściowego wektora uczącego przed użyciem go do procesu adaptacji.

Inne metody regularyzacji zostały przedstawione w rozdziale 4.

3.5.3. Inne metody uczenia sieci RBF

Warto tu również wspomnieć o metodzie ortogonalizacji najmniejszych kwadratów (*ang. ortogonal least squares*) Chen'a (i. in.) [27, 28] do uczenia i konstrukcji sieci z radialnymi funkcjami bazowymi. Metoda najczęściej wykorzystuje algorytm ortogonalizacji Grama-Schmidta.

Rozszerzenie tej metody o regularyzację zaproponował Orr [143].

Z kolei Bishop proponuje używanie algorytmu EM (*ang. expectation maximization*) [11] do uczenia sieci RBF.

Lowe w [125] opisał specjalną wersję sieci RBF, przeznaczoną do klasyfikacji danych poprzez estymacje prawdopodobieństw rozkładów. Metoda polega na estymacji prawdopodobieństw a posteriori $p(c|\mathbf{x})$, czyli prawdopodobieństw, że dany wektor $\mathbf{x}$

przynależy do klasy c. Takie prawdopodobieństwo a posteriori można zrekonstruować z prawdopodobieństw cząstkowych, korzystając z twierdzenia Bayesa

$$p(c_i|\mathbf{x}) = \frac{p(c_i)p(\mathbf{x}|c_i)}{p(\mathbf{x})} \tag{3.72}$$

Ponieważ raczej nie zdarza się, aby za pomocą pojedynczego rozkładu gaussowskiego można było estymować rozkład danych w klastrach danej klasy, używa się mieszanki rozkładów warunkowych $q(\mathbf{x}|s)$ z różnymi współczynnikami mieszania

$$p(x) = \sum_s p(s)q(\mathbf{x}|s) \tag{3.73}$$

$$p(\mathbf{x}|c_i) = \sum_s p(s;i)q(\mathbf{x}|s) \tag{3.74}$$

wtedy wykorzystując (3.73) i (3.74), równanie (3.72) przyjmuje postać

$$p(c_i|\mathbf{x}) = \sum_s \frac{p(c_i)p(s;i)}{p(s)} \cdot \frac{p(s)q(\mathbf{x}|s)}{\sum_{s'} p(s')q(\mathbf{x}|s')} \equiv \sum_j w_{ij} G(\mathbf{x}|j) \tag{3.75}$$

współczynniki $w_{ij} = p(c_i)p(s;i)/p(s)$ są wagami warstwy wyjściowej, opisującymi istotność j-tego węzła-podrozkładu dla i-tej klasy, a funkcjami bazowymi są znormalizowane funkcje $G(\mathbf{x}|j)$ (porównaj z równaniem (2.75)).

Wilson i Martinez [175] pokazują używając sieci RBF, iż kiedy atrybuty danych są różnych typów (ciągłe, dyskretne, nominalne), należy wtedy dobrać odpowiednią miarę odległości, aby uzyskać możliwie najlepsze rezultaty. Różne miary odległości zostały już przedstawione w podrozdziale 2.2.1.

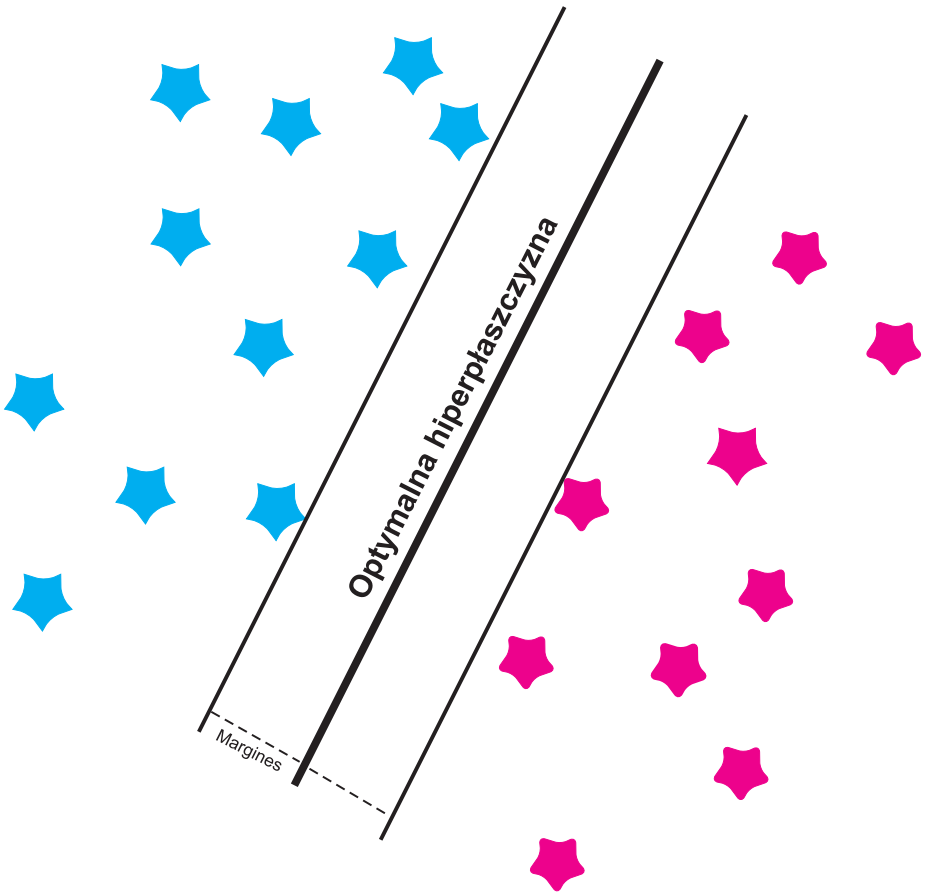
3.5.4. Support Vector Machines (SVM)

Innym, bardzo dobrze określonym matematycznie modelem, jest SVM zaproponowany przez Vapnika [168, 169]. SVM może być używany do klasyfikacji i regresji. Centralnym punktem metody SVM jest konstrukcja *optymalnej hiperpłaszczyzny*, której zadaniem jest rozseparowanie danych, należących do przeciwnych klas, z możliwie największym marginesem zaufania. Poprzez margines zaufania rozumie się tu odległość pomiędzy optymalną hiperpłaszczyzną i najbliższym jej wektorem. Przykład jest zilustrowany na rysunku 3.6). Tak zdefiniowana optymalna hiperpłaszczyzna, określona przez współczynniki $\boldsymbol{w}$ i w_0, spełnia poniższą nierówność

$$\frac{y_k D(\mathbf{x}_k)}{||\boldsymbol{w}||} \geqslant \tau \qquad k = 1, 2, \ldots, n \tag{3.76}$$

oczywiście przy założeniu, że istnieje margines ufności τ, a $D(\mathbf{x})$ jest zdefiniowane przez:

$$D(\mathbf{x}) = (\boldsymbol{w} \cdot \mathbf{x}) + w_0 \tag{3.77}$$



Rysunek 3.6: Optymalna hiperpłaszczyzna.

Jednakże ideą SVM nie jest konstrukcja optymalnej hiperpłaszczyzny w przestrzeni wejściowej, lecz w bardzo wielowymiarowej przestrzeni cech $\mathcal{Z}$, która jest nieliniowym produktem funkcji bazowych $g_i(\mathbf{x})$ (wybranym *a priori*) z przestrzeni wejściowej. Równanie optymalnej hiperpłaszczyzny przyjmuje wtedy postać

$$D(\mathbf{x}) = \sum_{i=1}^{n} a_i y_i H(\mathbf{x}_i, \mathbf{x}) \tag{3.78}$$

gdzie $H_i((\mathbf{x}_i, \mathbf{x})$ jest jądrem iloczynu skalarnego (*ang. inner product kernel*) funkcji bazowych (przestrzeni cech $\mathcal{Z}$) $g_j(\mathbf{x}), j = 1, 2, \ldots, m$. Iloczyn skalarny wpływu może być zdefiniowany przez poniższe równanie

$$H(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{m} g_i(\mathbf{x}) g_i(\mathbf{x}') \tag{3.79}$$

W szczególności m może być nieskończone. Zgodnie z teorią przestrzeni Hilberta,

funkcja jądrowa H musi spełniać warunki

$$\int\int H(\boldsymbol{x},\boldsymbol{x}')\phi(\boldsymbol{x})\phi(\boldsymbol{x}')\ d\boldsymbol{x}\ d\boldsymbol{x}' > 0 \qquad \text{for all } \phi \neq 0, \int \phi^2(\boldsymbol{x})d\boldsymbol{x} < \infty \tag{3.80}$$

Iloczyn skalarny może być definiowany różnie dla różnych problemów, np. dla wielomianów stopnia q mamy:

$$H(\boldsymbol{x},\boldsymbol{x}') = [(\boldsymbol{x}\cdot\boldsymbol{x}') + 1]^q \tag{3.81}$$

z kolei dla sieci RBF zdefiniowanej przez

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{n} a_i \exp\left(-\frac{|\boldsymbol{x}-\boldsymbol{x}'|^2}{\sigma^2}\right)\right) \tag{3.82}$$

H przyjmuje postać

$$H(\boldsymbol{x},\boldsymbol{x}') = \exp\left(-\frac{|\boldsymbol{x}-\boldsymbol{x}'|^2}{\sigma^2}\right) \tag{3.83}$$

Dla sieci MLP zdefiniowanych poprzez

$$f(\boldsymbol{x}) = \text{sign}\left(\sum_{i=1}^{n} a_i \tanh(v(\boldsymbol{x}\cdot\boldsymbol{x}') + a) + b\right) \tag{3.84}$$

H jest równe

$$H(\boldsymbol{x},\boldsymbol{x}') = \tanh(v(\boldsymbol{x}\cdot\boldsymbol{x}') + a) \tag{3.85}$$

Dla rozwiązania problemu klasyfikacji musimy wyznaczyć parametry $a_1, a_2, \ldots, a_n$ równania (3.78). W tym celu należy znaleźć maksimum funkcjonału $Q(\boldsymbol{a})$

$$Q(\boldsymbol{a}) = \sum_{i=1}^{n} a_i - \sum_{i,j=1}^{n} a_i a_j y_i y_j H(\boldsymbol{x}_i,\boldsymbol{x}_j) \tag{3.86}$$

przy założeniach

$$\sum_{i=1}^{n} y_i a_i = 0, \qquad 0 \leqslant a_i \leqslant \frac{C}{n}, \quad i = 1,2,\ldots,n \tag{3.87}$$

dla danych treningowych $(\boldsymbol{x}_i, y_i), i = 1,2,\ldots,n$ z wybranym iloczynem skalarnym H i stałą C (nie ma teoretycznych wytycznych co do samego wyboru stałej C, dla przypadku separowalnego $C = \infty$). Bardzo ważną cechą SVM jest niezależność procesu od wymiarowości przestrzeni cech $\mathcal{Z}$.

Girosi [74] pokazał równoważność pomiędzy metodą *Sparse Approximation* sieci typu RBF i SVM. Jedyną, choć bardzo istotną, zmianą w sieci RBF jest definicja funkcji błędu:

$$E[\boldsymbol{w},\xi] = \left\| f(\boldsymbol{x}) - \sum_{i=1}^{n} \xi_i w_i G_i(\boldsymbol{x}) \right\|_{L_2}^2 + \lambda \left(\sum_{i=1}^{n} \xi_i\right)^p \tag{3.88}$$

gdzie wartości ξ_i należą do zbioru $0, 1$, p jest stałą dodatnią, zazwyczaj równą 1.

Jak widać z powyższych własności modelu SVM, może on być skutecznie używany do konstrukcji sieci RBF. Wydaje się również, że ciekawe rezultaty można by uzyskać, używając różnych funkcji bicentralnych jako iloczynu skalarnego H zamiast funkcji gaussowskiej.

3.6. Porównanie sieci RBF z sieciami MLP

Jak widać z równania (3.2) główną różnicą pomiędzy siecią RBF i MLP są funkcje transferu neuronów warstw ukrytych. Dla sieci RBF mamy pewną funkcję radialną

$$h_i^{RBF} = \phi_1(||\mathbf{x} - \mathbf{t}_i||) \tag{3.89}$$

podczas gdy dla sieci MLP jest to produkt skalarny

$$h_i^{MLP} = \phi_2(\mathbf{x}^T \mathbf{t}_i) \tag{3.90}$$

Stąd też i sposoby działania tych sieci różnią się zasadniczo. Sieć MLP użyta do klasyfikacji będzie dzieliła wielowymiarową przestrzeń hiperpłaszczyznami. Przez to część powstałych podobszarów jest nieskończona. Natomiast sieć RBF będzie tworzyła lokalne obszary wokół klastrów danych (patrz rys. 3.7).

Również patrząc niejako *z góry* na całościowe równania sieci MLP i sieci RBF można dostrzec sporą różnicę:

$$\textbf{RBF}: \quad f_1(\mathbf{x}; \boldsymbol{w}) = \boldsymbol{w} \cdot \mathbf{G}(\mathbf{x}) \tag{3.91}$$

$$\textbf{MLP}: \quad f_2(\mathbf{x}; \boldsymbol{w}) = \sigma\left(\sum_{i_1} w_{i_1}^1 \sigma\left(\sum_{i_2} w_{i_2}^2 \sigma\left(\ldots \sigma\left(\sum_{i_L} w_{i_L}^L x_{i_L}\right)\ldots\right)\right)\right) \tag{3.92}$$

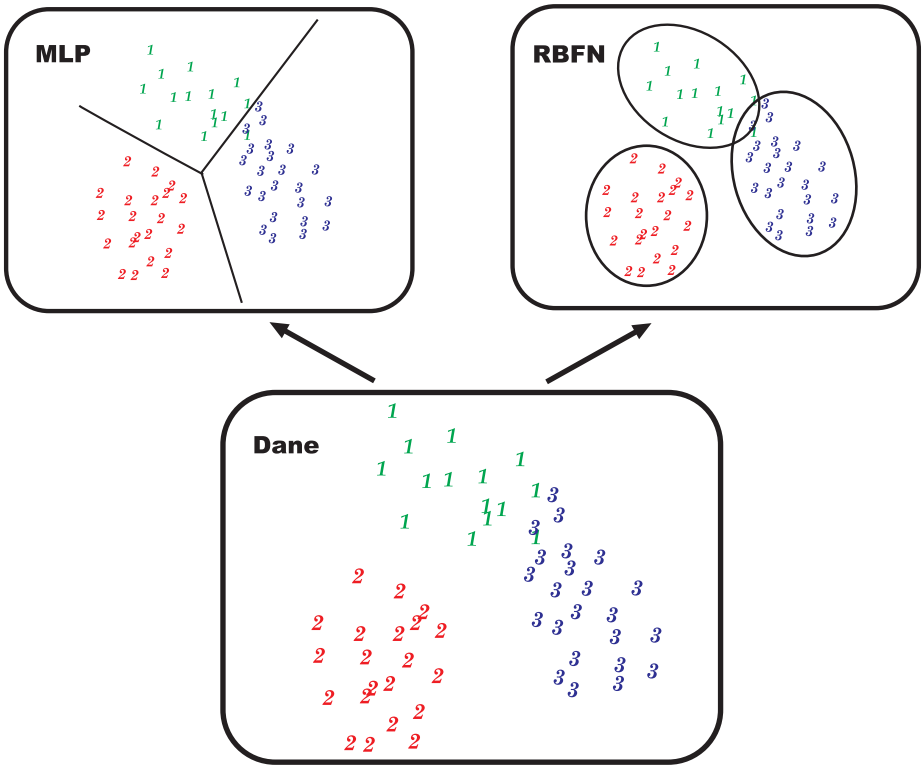
Wartym uwagi jest też porównanie powierzchni stałych wartości aktywacji neuronów wielowarstwowej sieci MLP i sieci RBF. W przypadku sieci typu MLP, wartość aktywacji neuronów jest stała dla wektorów leżących na hiperpłaszczyźnie

$$\boldsymbol{w}^T \mathbf{x} + w_0 = \mathrm{const}, \tag{3.93}$$

natomiast w przypadku radialnych funkcji bazowych aktywacja jest stała na powierzchni hipersfery określonej miarą odległości

$$||\mathbf{x} - \mathbf{t}||^2 = \mathrm{const} \tag{3.94}$$

Z kolei, gdy wektory wejściowe mają normę równą 1 ($||\mathbf{x}|| = 1$), to można to wykorzystać do nielosowej inicjacji parametrów uczenia sieci MLP i tym samym ułatwić



Rysunek 3.7: Podziały przestrzeni danych przy użyciu sieci RBF i MLP.

proces uczenia się sieci. Taka inicjalizacja jest częściowym odpowiednikiem nielosowego wyboru wag startowych w sieciach RBF.

Normę wejściowych wektorów równą 1 można uzyskać poprzez transformacje wektora wejściowego, dodanie do niego dodatkowego wymiaru i wyrzutowania na sferę jednostkową. Wtedy, łącząc to z wybraną metodą klasteryzacji (patrz rozdział 3.3) możemy dokonać wyboru liczby neuronów i wstępnej inicjalizacji wag, zgodnie z procedurą opisaną w [43]. Niech $\mathbf{x} = \{x_1, , x_2 \dots , x_d\}$, wtedy niech $\mathbf{x}' = \{x'_1, x'_2, \dots , x'_d, x'_{d+1}\}$ będzie zdefiniowany jako [35]

$$\begin{aligned} x'_i &= \frac{x_i}{r} \qquad i = 1, 2, \dots , d \\ x'_{d+1} &= \sqrt{1 - \frac{1}{r^2} \sum_{i=1}^{d} x_i^2} \end{aligned} \tag{3.95}$$

gdzie $r \geqslant \max_{<\mathbf{x},y>\in\mathcal{S}} \|\mathbf{x}\|$, $\mathcal{S}$ jest zbiorem wektorów uczących. Po takiej transformacji wszystkie punkty zostają wyrzutowane na półhipersferę. Najważniejszy jest fakt, iż podobne wektory będą wtedy tworzyły dość spójne klastry. Takie klastry, reprezentowane przez odpowiednio wybrane prototypy, można odciąć hiperpłaszczyzną od owej półhipersfery, co z kolei można zrealizować, wstawiając neuron z funkcją tanh lub funkcją sigmoidalną. Parametry początkowe tych funkcji można wyznaczyć już

całkiem łatwo.

Niezależnie bardzo podobny sposób inicjalizacji zaproponowali Denoeux i Lengellé [35], a ich przykłady dowodzą, iż taki sposób inicjalizacji znacznie poprawia efektywność późniejszego procesu uczenia sieci ze wsteczną propagacją błędu.

Istnieje możliwość uzyskania efektywniejszej transformacji, poniższa transformacja ma na celu wykorzystanie jak największej powierzchni hipersfery, a nie jej połowy lub części połowy. Z drugiej strony nie wykorzystuje całej hipersfery, aby bardzo odległe dane nie mogły ulec błędnemu połączeniu — rozpinanie płaszczyzny na kule powoduje, iż krawędzie spotykają się. Przekształcenie następuje niezależnie dla każdej i-tej ($i = 1, 2, \ldots, d$) współrzędnej p-tego wektora ($\mathbf{x}_p = \{x_{p,i}, x_{p,i}, \ldots, x_{p,i}\}$):

$$x'_{p,i} = x_{p,i} - (\max_s x_{s,i} + \min_s x_{s,i})/2 \tag{3.96}$$

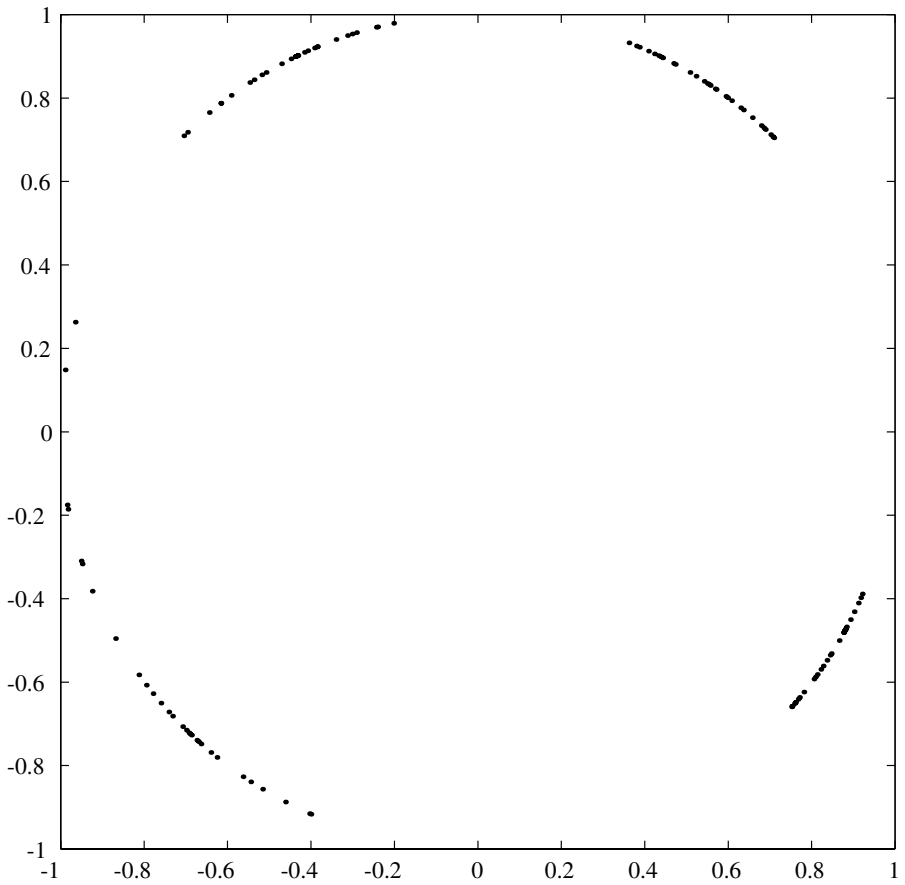
$$x''_{p,i} = \eta \cdot 2 \cdot x'_{p,i} / \max_s \|\mathbf{x}'_s\| \tag{3.97}$$

$$x'''_{p,i} = \begin{cases} -|x''_{p,i} - 1| + 1 & x''_{p,i} \geqslant 0 \\ |x''_{p,i} + 1| - 1 & x''_{p,i} < 0 \end{cases} \tag{3.98}$$

$$x'''_{p,d+1} = \operatorname{sign}\left(1 - \|\mathbf{x}''_p\|^2\right) \sqrt{\left|1 - \|\mathbf{x}'''_p\|^2\right|} \tag{3.99}$$

η określa jaką część obwodu hipersfery będzie rozpinać transformacja. Finalny wektor $\mathbf{x}'''_p$ jest określony poprzez $\{x'''_{p,1}, x'''_{p,2}, \ldots, x'''_{p,d+1}\}$ gdzie p określa indeks wektora uczącego.

Należy się jednak liczyć z tym, że transformacja jest niezależna dla każdego wymiaru i tym samym ulegają zmianie zależności pomiędzy poszczególnymi wymiarami. Na rysunku 3.8 można zobaczyć przykład użycia powyższej transformacji dla przestrzeni dwu wymiarowej. Na przykładzie widać jak 4 klastry zostały rozmieszczone na hipersferze. Po wykonaniu takiego przekształcenia danych można przejść do następnych etapów opisanych w [43], czyli dokonać klasteryzacji, która umożliwi wyznaczenie prototypów, na których podstawie zostaną obliczone wartości wag w nowej przestrzeni $d + 1$ wymiarowej.



Rysunek 3.8: Przykładowa transformacja danych wejściowych z czterema klastrami na hipersferę.

Rozdział 4

Ontogeniczne modele sieci neuronowych

Do ontogenicznych modeli sieci neuronowych zalicza się takie modele, które mogą zmieniać swoją architekturę w procesie uczenia się. Architektura to po prostu układ warstw neuronów i połączeń pomiędzy neuronami, co wraz z funkcjami transferu, odpowiadającymi neuronom, determinuje możliwości całego modelu adaptacyjnego[1].

Architektura +	Funkcje Transferu	=	Złożoność Modelu

Modele, które nie mogą modyfikować swojej architektury muszą ją mieć dobrze ustaloną na podstawie wiedzy *a priori* przed rozpoczęciem procesu uczenia. To niestety jest często bardzo trudne, ponieważ zazwyczaj nie jest znana złożoność problemu, który ma być rozwiązany. Zakładając iż nasza sieć umie dobrze podejmować decyzje (możliwie najlepiej w każdej chwili uczenia) o powiększaniu i pomniejszaniu swojej architektury, może ona kontrolować dobrze czy aktualna architektura modelu jest wystarczalna do reprezentowania rozwiązywanego problemu lub jest niewystarczalna albo za duża.

Złożoność Modelu	$\approx$	Złożoność Problemu

Jak zostało wspomniane w poprzednim rozdziale, gdy liczba powiązań lub neuronów będzie za mała, model adaptacyjny nie będzie w stanie nauczyć się reprezentacji danego problemu, natomiast gdy połączeń między neuronami lub samych neuronów będzie

[1] Oczywiście abstrahując od algorytmu adaptacji sieci.

za dużo, to model adaptacyjny nie będzie w stanie osiągnąć zadowalającego poziomu generalizacji (za duża pojemność modelu powoduje, iż dochodzi do przeuczenia się sieci). Dlatego złożoność modelu adaptacyjnego powinna odpowiadać złożoności rozpatrywanego problemu. Wtedy model ma największą szansę na uzyskanie możliwie najlepszej jakości generalizacji.

CEL:	**Jak najlepsza GENERALIZACJA**

Należy zwrócić uwagę, iż nie minimalna liczba neuronów, czy połączeń między neuronami powinna być głównym celem, lecz znalezienie takiej architektury, która umożliwi uzyskanie jak najlepszej jakości generalizacji. Dlatego, aby sieć mogła ewoluować w kierunku uzyskania jak najlepszej generalizacji, musi posiadać pewien margines swobody, który leży w parametrach adaptacyjnych i pozwala zmieniać płynnie stany modelu. Dobrze dobrane marginesy swobody wraz z kryteriami kontroli złożoności modelu, pozwalają także walczyć z problemem lokalnych minimów. Model zmieniając swoją architekturę przechodzi do innych przestrzeni funkcji błędów, w której następuje kontynuacja procesu uczenia, po czym znów mogą nastąpić zmiany architektury itd. Tym sposobem, taki model uczący może eksplorować bardzo różne przestrzenie w poszukiwaniu pewnego optimum. Dlatego należy zadbać, aby sam proces uczenia, jak i mechanizmy kontroli złożoności architektury mogły jak najefektywniej czerpać informacje ze zbioru treningowego i wszelkiej istniejącej wiedzy *a priori*. Na przykład, dodając do modelu informacje *a priori* o dokładności danych (tj. dokładności dokonanych pomiarów), na których opiera się proces adaptacji modelu, możemy doprowadzić do znacznie innych, zazwyczaj lepszych, wyników niż te, które można by uzyskać, nie dodając takiej informacji.

W przeciągu ostatnich lat powstały różnorodne modele, które modyfikują swoją architekturę. Wszystkie metody kontroli złożoności architektur sieci można podzielić na trzy grupy:

- powiększające — do tych modeli należą algorytmy, umożliwiające dokładanie nowych neuronów, nowych połączeń pomiędzy neuronami lub umożliwiające dodawanie neuronów i połączeń, a także sieci, których liczba warstw również może rosnąć;
- zmniejszające — do których z kolei należą metody, które pozwalają na usuwanie zbędnych neuronów czy połączeń miedzy neuronami lub algorytmy, które potrafią konkatenować grupy neuronów lub połączenia pomiędzy neuronami;
- systemy kooperujące — te systemy, to grupy modeli, z których każdy indywidualnie ma za zadanie rozwiązywać ten sam problem lub jakąś jego część, a następnie system zarządzający decyduje jaka będzie ostateczna decyzja.

Najczęściej jednak spotyka się systemy, które należą do jednej z powyższych grup, czyli gdy mają możliwość powiększania swoje struktury, często startując z pustej

struktury (lub bardzo prostej), to zazwyczaj nie usuwają neuronów ani połączenia pomiędzy neuronami. Natomiast rzadziej spotyka się systemy, które należą do grupy systemów kooperujących, których podsystemy zmieniałyby swoją architekturę.

PYTANIA:	**Czy/Kiedy**	**usuwać dodawać**	**?**
	Co	**usuwać dodawać**	**?**

Jednakże najważniejszą część modeli ontogenicznych stanowią kryteria, które decydują o tym, czy i/lub kiedy należy dokonać zmian w strukturze modelu. Z kolei inne kryteria umożliwiają ustalenie, której części struktury ma dotyczyć dana zmiana oraz jak jej dokonać.

Od skuteczności tych kryteriów w prostej linii uzależnione jest osiągnięcie żądanego poziomu generalizacji lub totalnej klęski podczas uczenia. Należy zwrócić uwagę, iż zmiany dokonywane na już istniejącej strukturze nie powinny prowadzić do pogorszenia już istniejącego poziomu *reprezentacji* problemu, lecz umożliwić jej polepszenie w dalszej procedurze adaptacji. Z kolei wybór części struktury, która ma podlegać zmianie, neuron(-y) lub połączenie(-nia), też nie jest trywialny. To, iż dany neuron lub pewne połączenie w danej chwili nie odgrywa istotnej roli w sieci neuronowej wcale nie musi oznaczać jego zbędności. Czasami proces adaptacji może po prostu nie zdążyć wykorzystać owego neuronu lub połączenia. Innymi słowy uczenie może być jeszcze po prostu w dość intensywnej fazie.

Niektóre systemy ontogeniczne mogą sprostać jeszcze ciekawszym i trudniejszym problemom. Mogą mianowicie estymować rozkład danych (opisujących pewien problem), który może podlegać zmianom w czasie, w tym i złożoność samego problemu może podlegać zmianie, co wymaga bieżących zmian architektury sieci, tak, aby model mógł znów odzwierciedlać zmienioną złożoność problemu. Poza proponowaną przeze mnie siecią IncNet, opisywaną w tej pracy, estymacją niestacjonarnych rozkładów zajmował się Fritzke [69]. Aby dany model uczący mógł estymować rozkłady, które zmieniają się w czasie, musi sam kontrolować wystarczalność lub niewystarczalność aktualnej w danej chwili struktury modelu, tak aby mógł ją zmieniać w zależności od obserwowanej złożoności problemu. Większość spotykanych algorytmów jednak albo dopuszcza np. usuwania neuronów podczas uczenia, albo w modelach rozrastających się neurony dokładane nierzadko są zamrażane i nie podlegają dalszej adaptacji. W przyszłości właśnie takie modele, które będą mogły, a nawet szybko mogły, estymować zmienne rozkłady, będą wykorzystywane do symulacji zaawansowanych systemów kognitywnych.

4.1. Modele zmniejszające strukturę

Modele, które potrafią zmniejszać swoją strukturę poprzez usuwanie połączeń lub usuwanie neuronów, próbują wykorzystywać przeróżną informację jaka drzemie w samych danych i/lub w aktualnym w danej chwili stanie sieci (wartości wag, progów, rozmyć, etc.) i jej architektury.

Na samym początku należy wspomnieć o najprostszym chyba sposobie na usuwanie nieprzydatnych neuronów, który, jak i część z poniżej opisanych metod, wyznacza współczynniki *istotności* lub *przydatności* każdego neuronu w bieżącej strukturze sieci. W tym algorytmie są one opisane wzorem

$$s_i = E(\text{bez neuronu } i) - E(\text{z neuronem } i) \tag{4.1}$$

który wyznacza różnice pomiędzy błędem sieci uzyskanym *bez* i *z* udziałem neuronu i. Wyznaczone w ten sposób wartości s_i dla poszczególnych neuronów opisują ich przydatność w strukturze sieci. Jednak sposób ten wymaga dość sporych nakładów obliczeniowych — do wyznaczenia każdego współczynnika s_i równania (4.1) należy wyznaczyć błąd dla każdego wektora ze zbioru treningowego. Opierając się na powyżej zdefiniowanych współczynnikach istotności, można wybrać neurony, dla których współczynniki te są znacznie mniejsze od wartości funkcji błędu wyznaczonych dla tych neuronów, a następnie usunąć wybrane neurony.

Zbliżony (również pasywny[2]) sposób wyznaczania współczynników istotności został użyty w rozwijanym w naszym zespole systemie FSM (*ang. Feature Space Mapping*) [1, 44, 49, 45]. Jego krótki opis został zamieszczony w podrozdziale 4.2.

Współczynniki istotności wyznacza się dla każdego neuronu warstwy ukrytej po przerwaniu procesu uczenia w następujący sposób:

$$Q_i = \frac{C_i(\mathbf{X})}{|\mathbf{X}|} \tag{4.2}$$

gdzie $|\mathbf{X}|$ jest liczbą wzorców uczących, $C_i(\mathbf{X})$ określa liczbę poprawnie sklasyfikowanych wzorców ze zbioru $\mathbf{X}$ przez i-ty neuron. W sieci FSM każdy neuron warstwy ukrytej odpowiada pewnej podprzestrzeni (tj. klastrowi), z którą związana jest informacja o jego przynależności do pewnej klasy. Mając wyznaczone współczynniki Q_i, można już dokonać usunięcia tych neuronów, dla których wartości te są najbliższe zeru.

4.1.1. Modele zmniejszające strukturę a regularyzacja

Metody zmniejszające strukturę sieci neuronowej często można rozpatrywać jako metody regularyzacji. Najbardziej wzorcowym przykładem może być rozpad wag (*ang. weight decay*) [87] przedstawiony już w podrozdziale 3.5.2, gdzie do miary

[2] Przez pasywność rozumie się tu brak związku z samym algorytmem uczenia, jak i samą funkcją błędu.

błędu modelu $E_0(f)$ (3.14) zostaje dodany czynnik regularyzacyjny

$$E_{wd}(f, \boldsymbol{w}) = E_0(f) + \lambda \sum_{i=1}^{M} w_i^2 \tag{4.3}$$

Ustalając pewien próg θ, możemy dokonać usunięcia połączeń czy neuronów, których wartości wag są mniejsze od progu θ po zakończeniu lub przerwaniu procesu uczenia. Jednakże powyższa funkcja błędu przejawia tendencje do tego, aby istniało wiele małych wag i wręcz doprowadza do sytuacji, w której nie pozostaje nawet mała część wag z dużymi wartościami. Taka sytuacja jest spowodowana karą za **każdą** znaczącą wartość wagi, w tym i istotną wagę zgodnie z (4.3). Aby uciec od powyższego problemu Weigend (m. in.) [170, 171] zaproponował eliminację wag (*ang. weight elimination*), stosując istotnie inny człon regularyzacyjny

$$E_{we}(f, \boldsymbol{w}) = E_0(f) + \lambda \sum_{i=1}^{M} \frac{w_i^2/w_0^2}{1 + w_i^2/w_0^2} \tag{4.4}$$

gdzie w_0 jest pewnym ustalonym współczynnikiem. Eksperymenty wykazały, iż współczynnik w_0 rzędu jedności odpowiada wartością aktywacji podobnego rzędu. Tak sformułowany człon regularyzacyjny przestaje zmuszać, by wartości wszystkich wag były małe, a pozwala, aby istniała pewna liczba wag (większych od w_0) przyjmujących dowolnie duże wartości (optymalne). Wynika to z własności powyższego członu: waga w_i dla której $|w_i| >> w_0$ ma wkład bliski wartości λ, natomiast dla wag w_i dla których $|w_i| << w_0$, wkład jest bliski zeru.

Parametr λ może podlegać adaptacji podczas uczenia. Adaptacje można przeprowadzać raz na epokę, zgodnie z poniższym schematem:

- $\lambda = \lambda + \Delta\lambda \quad$ gdy $E_n < D \vee E_n < E_{n-1}$
- $\lambda = \lambda - \Delta\lambda \quad$ gdy $E_n \geqslant E_{n-1} \wedge E_n < A_n \wedge E_n \geqslant D$
- $\lambda = 0.9\lambda \quad$ gdy $E_n \geqslant E_{n-1} \wedge E_n \geqslant A_n \wedge E_n \geqslant D$

E_n jest błędem ostatniej (n-tej) epoki uczenia dla całego zbioru treningowego, $A_n = \gamma A_{n-1} + (1-\gamma)E_n$ (γ jest bliskie 1), natomiast D określa błąd docelowy dla całego procesu uczenia.

Z kolei poniższy człon dodany do standardowej funkcji błędu $E_0(f)$ (3.14)

$$E_{mlp2ln}(f, \boldsymbol{w}) = E_0(f) + \lambda \sum_{i=1}^{M} w_i^2 (w_i - 1)^2 (w_i + 1)^2 \tag{4.5}$$

został użyty do budowania i uczenia sieci MLP, służącej do ekstrakcji reguł logicznych [40].

Innymi przykładami metod regresji, umożliwiających kontrolę struktury sieci, może być lokalna regresja grzbietowa (*ang. local ridge regression*), opisana już w podrozdziale 3.5.2, czy też metody współdzielenia wag poprzez różne neurony.

Z kolei pośród metod współdzielenia wag ciekawą jest metoda miękkiego współdzieleniem wag (*ang. soft weight sharing*), która nie wymaga, aby wagi danej grupy były sobie równe, lecz dopuszcza pewien rozrzut [142] σ_j. Funkcja błędu z takim członem regularyzacyjnym przyjmuje postać

$$E_{sws}(f, \boldsymbol{w}) = E_0(f) - \lambda \sum_{i}^{M} \ln \left(\sum_{j=1}^{k} \alpha_j \phi_j(w_i) \right) \tag{4.6}$$

gdzie $\phi(w_i)$ jest zdefiniowane poprzez

$$\phi_j(w) = \frac{1}{(2\pi\sigma_j^2)^{1/2}} \exp\left(-\frac{(w-\mu_j)^2}{2\sigma_j^2} \right) \tag{4.7}$$

Taki człon również może pełnić role nie tylko czysto regularyzacyjną, ale i umożliwiać usuwanie połączeń. Parametry σ_j podlegają adaptacji, co może doprowadzić do stanu w którym pewne grupy wag będą odpowiadały bardzo podobnym wagom, a w takim przypadku można dokonać ich ostatecznego połączenia.

4.1.2. Usuwanie wag metodami Optimal Brain Damage (OBD) i Optimal Brain Surgeon (OBS)

Niestety, nie zawsze użycie metody rozpadu wag, jak i eliminacji wag daje wystarczająco dobre rezultaty. Czasem aby uzyskać naprawdę mały błąd, niezbędne okazują się i małe wagi. Dla przykładu popatrzmy na rysunek 4.1 funkcji o kształcie meksykańskiego kapelusza. Aby dokonać aproksymacji takiej funkcji, można by użyć jednej funkcji gaussowskiej, jednakże wtedy błąd aproksymacji nie będzie mały. Natomiast, gdy użyć trzech funkcji gaussowskich, w tym dwie miałyby znacznie mniejsze wagi, błąd aproksymacji byłby bardzo mały.

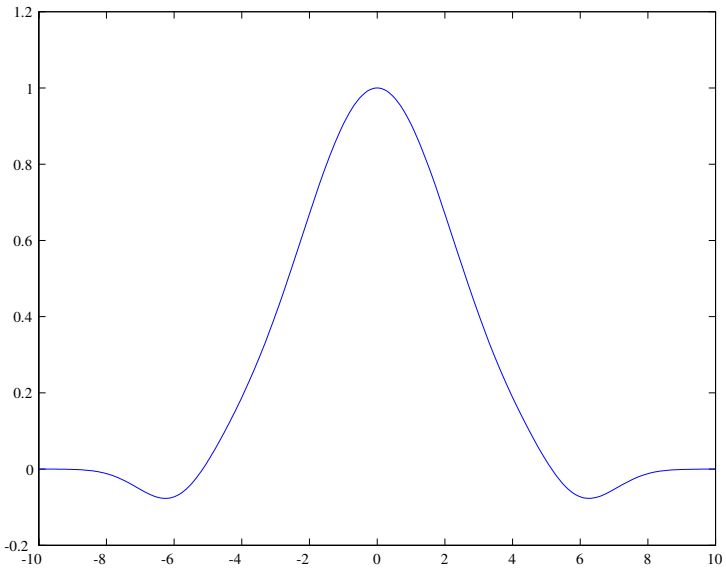
Z powyższych powodów celem pracy Le Cun'a było wyznaczenie parametrów *istotności* dla poszczególnych wag sieci neuronowej. Na podstawie tych parametrów można by dokonywać prób usuwania wag, których współczynniki istotności były najmniejsze, nie powodując istotnych zmian w funkcji błędu. Dlatego też Le Cun (m. in.) nazwał tą metodę *Optimal Brain Damage* (OBD) [32].

Droga rozwiązania problemu wiedzie przez analizę różnicy funkcji błędu δE, jaka powstaje pod wpływem zaburzenia wektora wag sieci o $\delta\boldsymbol{w}$. Analizuje się funkcję błędu, rozwiniętą w szereg Taylora

$$\delta E = \left(\frac{\partial E}{\partial \boldsymbol{w}} \right)^T \cdot \delta\boldsymbol{w} + \frac{1}{2} \delta\boldsymbol{w}^T \cdot \frac{\partial^2 E}{\partial \boldsymbol{w}^2} \cdot \delta\boldsymbol{w} + O(\|\delta\boldsymbol{w}\|^3) \tag{4.8}$$

gdzie $\frac{\partial^2 E}{\partial \boldsymbol{w}^2}$ tworzy Hessian H.

Gdy proces adaptacji zbiegł się, czyli parametry wag znajdują się w minimum funkcji błędu, to pierwszy człon sumy prawej strony równania (4.8) można pominąć. Również



Rysunek 4.1: Meksykański kapelusz.

i trzeci składnik dla uproszczenia można pominąć. Le Cun zakłada również, iż tylko przekątna macierzy Hessianu jest rzeczywiście istotna, co prowadzi do uproszczenia równania (4.8) do postaci

$$\delta E = \frac{1}{2} \sum_{i}^{M} H_{ii} \delta w_i^2 \tag{4.9}$$

gdzie H_{ii} to i-ty element diagonalny macierzy Hessianu H. Z równania (4.9) widać, jak zmiana poszczególnych wag w_i może wpływać na zmiany funkcji błędu. W szczególnym przypadku można zobaczyć, jak wpływa usunięcie i-tej wagi. Wtedy $w_i = \delta w_i$, co prowadzi do wyznaczenia współczynników istotności s_i dla każdej wagi w_i

$$s_i = H_{ii} w_i^2 \tag{4.10}$$

Algorytm składa się wtedy z następujących etapów:

1. Wstępny wybór architektury,
2. Uczenie sieci do etapu, w którym zmiany funkcji błędu nie będą już istotne,
3. Wyznaczenie współczynników istotności zgodnie z (4.10),
4. Usunięcie wag, których współczynnik istotności jest niski,
5. Jeśli któraś z wag została usunięta, to następuje skok do punktu 2.

Metoda ta została pomyślnie użyta do rozpoznawania pisma ręcznego, uzyskując błąd rzędu 3.4% na zbiorze testowym (2700 wektorów). Sieć była uczona na 9840 wektorach. Sama sieć składała się z pięciu wyspecjalizowanych warstw [31].

Następnym ciekawym krokiem była metoda zaprezentowana przez Hassibiego i Storka nazwana Optimal Brain Surgeon [82, 83].

Ta metoda usuwania zbędnych połączeń, podobnie jak metoda OBD, używa rozwinięcia funkcji błędu w szereg Taylora (4.8). Jednakże Hassibi (i. in.) twierdzi, iż zazwyczaj macierz Hessianu wag jest zupełnie niediagonalna i prowadzi do usuwania dobrych, czyli istotnych wag.

Metoda Optimal Brain Surgeon również zakłada, że pierwszą i trzecią część prawej strony równania (4.8) można zaniedbać. Tym samym, jako cel metody proponuje się minimalizację

$$\min_q \min_{\delta \boldsymbol{w}} \{\frac{1}{2}\delta \boldsymbol{w}^T H \delta \boldsymbol{w}\} \quad \text{pod warunkiem } \delta w_q + w_q = 0 \tag{4.11}$$

która następnie jest rozwiązywana za pomocą mnożników Lagrange'a

$$L = \frac{1}{2}\delta \boldsymbol{w}^T H \delta \boldsymbol{w} + \lambda(\delta w_q + w_q) \tag{4.12}$$

Co dalej prowadzi do rozwiązania i wyznaczenia istotności dla wag s_q:

$$s_q = \frac{1}{2}\frac{w_q^2}{H^{-1}_{qq}} \tag{4.13}$$

$$\delta \boldsymbol{w} = -\frac{w_q}{H^{-1}_{qq}} H^{-1} \cdot \mathbf{i}_q \tag{4.14}$$

$\mathbf{i}_q$ jest wektorem składającym się z zer i jedynki na q-tej pozycji. Z kolei $\delta \boldsymbol{w}$ to poprawki dla wag jakie należy nanieść po usunięciu wagi w_q.

Proces uczenia sieci przebiega podobnie jak dla sieci z OBD (patrz powyżej). Hassibi prezentuje też zależność pomiędzy usuwaniem połączeń bazującym na wielkości wag, a metodami OBD i OBS

$$E(wagi) \geqslant E(OBD) \geqslant E(OBS) \tag{4.15}$$

4.1.3. Statystyczne i inne metody zmniejszania struktury sieci neuronowych

Oba algorytmy OBD i OBS mogą być stosowane do sieci MLP, jak i do sieci RBF. Statystyczne podejście do usuwania zbędnych połączeń zostało przedstawione przez Finnoffa i Zimmermann [59, 58] i Cottrella (i. in.) [30], a później użyte też w pracy Weigend'a (i. in.) w [172]. Idea tej metody opiera się na kumulowaniu różnic poszczególnych wag podczas uczenia w ramach jednej epoki. Następnie definiuje

się współczynniki s_i, oceniające przydatność dla każdej wagi i. Współczynnik s_i jest równy ilorazowi wartości wagi powiększonej o średnie wahanie wartości wagi podzielonemu przez standardowe odchylenie średniej różnicy tej wagi:

$$s_i = \frac{|w_i + mean(\Delta w_i^j)|}{std(\Delta w_i^j)} \tag{4.16}$$

w_i jest wartością wagi przed rozpoczęciem wyliczania zmian w kolejnej epoce, Δw_i^j jest równe zmianie wartości wagi w_i pod wpływem prezentacji j-tego wzorca uczącego. Poza tym mamy

$$mean(\Delta w_i^j) = \frac{1}{N}\sum_{j=1}^{N} \Delta w_i^j \tag{4.17}$$

a $std(\Delta w_i^j)$ jest zdefiniowane przez:

$$std(\Delta w_i^j) = \sqrt{\frac{1}{N}\sum_{j=1}^{N} (\Delta w_i^j - mean(\Delta w_i^j))^2} \tag{4.18}$$

Współczynnik s_i, testujący wagę i określony równaniem (4.16), jest duży, gdy waga jest duża, a przeciętne różnice zmian tej wagi są małe. W odwrotnym przypadku mamy do czynienia z wagą, która jest raczej mało istotna lub zupełnie nieprzydatna.

W pracy [58] została opisana też procedura *epsi-prune*, której zadaniem nie jest pełne usunięcie zbędnej wagi, lecz jej dezaktywacja, która może być tylko czasowa. Zanim dojdzie do usuwania wag sieć zostaje poddana procedurze uczenia, aż do przeuczenia się (np. aż do zaobserwowania pogorszenia błędu klasyfikacji bądź aproksymacji na podzbiorze zbioru treningowego wyodrębnionym do testowania). Następnie, po kilku epokach, wyznacza się współczynniki s_i zgodnie z równaniem (4.16). Wtedy następuje dezaktywacja tych wag, dla których współczynniki s_i są mniejsze od ϵ ($\epsilon > 0$). Natomiast wagi, które już zostały dezaktywowane poprzednio, a teraz ich współczynniki s_i są większe od ϵ, podlegają aktywacji z pewną małą losową wartością początkową. Po za tym w każdej epoce wartość ϵ jest powiększana o pewną wartość $\Delta\epsilon$ ($\Delta\epsilon > 0$), aż do osiągnięcia pewnej wartości ϵ_{max}.

Inna, statystyczna metoda kontroli usuwania neuronów dla sieci typu RBF zostanie przedstawiona w podrozdziale 4.3.

Wartym wspomnienia jest również podejście Orr'a [143] do sieci RBF, bazujące na macierzy projekcji P, która wspomaga dokonywanie różnych operacji. Macierz projekcji jest zdefiniowana jako

$$P = I - HA^{-1}H^T \tag{4.19}$$

gdzie H, to macierz wartości funkcji bazowych dla poszczególnych wektorów treningowych:

$$H = \begin{bmatrix} h_1(\mathbf{x}_1) & \dots & h_M(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ h_1(\mathbf{x}_n) & \dots & h_M(\mathbf{x}_n) \end{bmatrix} \tag{4.20}$$

z kolei A jest równe

$$A = H^T H + \Lambda \tag{4.21}$$

gdzie Λ jest macierzą przekątniową opisującą parametry regularyzacyjne poszczególnych wymiarów.

Macierz projekcji P pojawia się przy rozpisaniu błędu jaki popełni sieć RBF dla wektorów treningowych

$$\begin{aligned} \hat{y} - F &= \hat{y} - Hw && (4.22)\\ &= (I - HA^{-1}H^T)\hat{y} && (4.23)\\ &= P\hat{y} && (4.24) \end{aligned}$$

Macierz projekcji pozwala na szybkie wyznaczenie sumy błędu kwadratowego:

$$\begin{aligned} E &= (\hat{y} - F)^T(\hat{y} - F) && (4.25)\\ &= \hat{y}^T P^2 \hat{y} && (4.26) \end{aligned}$$

Także i obliczanie funkcji kosztu może być szybkie:

$$\begin{aligned} C &= (Hw - \hat{y})^T(Hw - \hat{y}) + w^T \Lambda w && (4.27)\\ &= \hat{y}^T P \hat{y} && (4.28) \end{aligned}$$

Usunięcie zbędnej funkcji bazowej h_j pociąga za sobą następującą operację na macierzy projekcji (zakłada się iż funkcja h_j została przesunięta do ostatniej kolumny macierzy P_m):

$$P_{m-1} = P_m + \frac{P_m h_j h_j^T P_m}{\lambda_j - h_j^T P_m h_j} \tag{4.29}$$

Takie usunięcie kosztuje n operacji arytmetycznych (n to liczba wektorów uczących), natomiast pełne przetrenowanie sieci wymagałoby $M^3 + nM^2 + p^2 m$ operacji (M liczba funkcji bazowych).

W podobny sposób można dodać nową funkcję bazową. Zostało to opisane w podrozdziale 4.2.

4.2. Modele o strukturach rozrastających się

Jednym z pierwszych modeli rozrastających się była sieć do klasyfikacji wzorców binarnych Mezarda i Nadala [130]. Sieć powstaje poprzez dokładanie kolejnych, nowych warstw o coraz mniejszej liczbie neuronów w kolejnych warstwach (połączenia między neuronami są tylko pomiędzy sąsiednimi warstwami neuronów). Każda z warstw musi spełniać następujący warunek, który został nazwany warunkiem *wierności* zbiorowi treningowemu. Warunek ten wymaga, aby dwa wzorce uczące należące do różnych klas nie były odwzorowane na taki sam wzorzec aktywacji danej warstwy. W przeciwnym razie dochodzi do sytuacji, w których dla wzorców różnych klas powstają aktywacje odpowiadającym tym samym grupom aktywacji danej warstwy, co uniemożliwiło by separacje takich wzorców w na poziomie następnej warstwy neuronów. Dopóki algorytm nie może zbudować *wiernej* warstwy, następuje dodawanie i uczenie kolejnych neuronów, aż dana warstwa spełni warunek. Uczenie odbywa się przy pomocy algorytmu kieszonkowego. Do uczenia wykorzystuje się wektory wejściowe, dla których uzyskano takie same aktywacje w budowanej warstwie, podczas gdy one należały do różnych klas (czyli wektory, które sprawiają, że nie jest spełniony warunek wierności). Główna idea algorytmu kieszonkowego polega na przetrenowaniu podzbioru wag, które nie ulegały zmianom podczas prezentacji wielu wektorów treningowych [72]. Tak zdefiniowany algorytm gwarantuje zbieżność procesu uczenia.

Następny algorytm (*ang. upstart*) również służy do klasyfikacji wzorców binarnych i został zaproponowany przez Freana [61]. Ten algorytm dokłada neurony pomiędzy już istniejącymi neuronami a wejściami sieci. Algorytm upstart startuje ucząc jeden neuron algorytmem kieszonkowym. Następnie, gdy neuron nie jest wystarczający, dokładane są dwa potomne neurony pomiędzy neuron który *popełnia błędy* a jemu odpowiadające wejścia. Wagi neuronów ustala się na takie wartości, aby pierwszy neuron odpowiadał wzorcom niesklasyfikowanym przez neuron-ojca niepoprawnie, a drugi sklasyfikowanym ale błędnie. Po zamrożeniu wag dochodzących do neuronu-ojca następuje przetrenowanie neuronów potomnych. Następnie i te wagi są zamrażane, i jeśli klasyfikacja nie jest satysfakcjonująca, dodawany jest kolejny neuron i proces jest powtarzany.

Algorytm korelacji kaskadowej (*ang. cascade-correlation network*) (CC) Fahlmana i Lebiere'a [54, 55] jest jednym z najsprawniejszych algorytmów uczenia sieci MLP, który umożliwia rozrastanie się struktury aż do uzyskania sieci o wystarczającej pojemności. Fahlman i Lebiere używali do adaptacji wag algorytmu szybkiej wstecznej propagacji, choć można rozważać użycie i innych algorytmów adaptacji.

Sieć kaskadowej korelacji rozpoczyna z jedną warstwą wag pomiędzy wejściem i wyjściem sieci. Następnie sieć jest uczona przez pewnie czas, po czym następuje dołożenie nowego neuronu do warstwy ukrytej. Proces ten jest powtarzany aż do uzyskania zakładanego poziomu błędu. Na wejścia kolejno dodawanych neuronów składają się wszystkie wejścia sieci i wyjścia poprzednio dodanych neuronów, skąd bierze się słowo kaskada w nazwie sieci.

Drugi człon nazwy sieci to maksymalizacja korelacji poniższego wyrażenia (4.30),

którego celem jest ustalenie możliwie najlepszych wag dla nowego neuronu

$$S = \sum_{i=1}^{p} \left| \sum_{j=1}^{n} (o_j - \bar{o})(e_j^i - \bar{e}^i) \right| \tag{4.30}$$

gdzie n oznacza liczbę wzorców uczących, p jest liczbą wyjść sieci, o_j jest wartością aktywacji neuronu, który ma być dodany dla j-tego wzorca uczącego, $\bar{o}$ jest średnią aktywacją dodawanego neuronu, e_j^i jest błędem i-tego wyjścia dla j-tego wzorca uczącego, a $\bar{e}^i$ jest średnim błędem i-tego wyjścia.

Łatwo można wyznaczyć pochodną równania (4.30)

$$\frac{\partial S}{\partial w_k} = z_i \sum_{i=1}^{p} \sum_{j=1}^{n} (e_j^i - \bar{e}^i) g_j' I_j^k \tag{4.31}$$

z_i jest równe 1 lub -1 zależnie od tego, czy korelacja pomiędzy neuronem i i-tym wyjściem sieci jest dodatnia lub ujemna; g_j' jest wartością pochodnej funkcji transferu dodawanego neuronu dla j-tego wektora uczącego, z kolei I_j^k jest wartością k-tego wejścia dodawanego neuronu dla j-tego wektora uczącego.

Start uczenia neuronu, który ma zostać dodany, rozpoczyna się od wartości losowych, dlatego też często wybiera się kilka neuronów-kandydatów, z których następnie wybiera się lepszego po wstępnym procesie adaptacji, w którym maksymalizuje się korelacje.

Falhman opracował również rekurencyjną wersję wyżej opisanego algorytmu [53].

Innym, ciekawym modelem, umożliwiającym rozbudowywanie struktury podczas uczenia się przez dodawanie nowych neuronów, jest wspomniany już system FSM (*ang. Feature Space Mapping*), rozwijany w naszym zespole [1, 44, 49]. Model ten używany głównie do klasyfikacji i wyciągania reguł logicznych z danych.

Architektura FSM jest praktycznie taka sama, jak sieci RBF. Składa się z warstwy wejściowej, wyjściowej i jednej warstwy ukrytej. Jako, iż typowym zastosowaniem systemu jest klasyfikacja, na wyjściu pojawia się informacja o tym, do której klasy został przypisany wektor pokazany warstwie wejściowej. Jednakże najważniejszą część stanowi warstwa ukryta, która jest odpowiedzialna za konstrukcje odpowiednich zbiorów odwzorowań. Mamy więc nie tylko podobieństwo architektur sieci RBF, ale i roli ich warstw ukrytych. W zastosowaniach klasyfikacyjnych, jak i w ekstrakcji reguł logicznych, neurony warstwy ukrytej odpowiadają pewnej (zależnej od funkcji transferu) klasteryzacji przestrzeni wejściowej. Typowymi funkcjami transferu neuronów warstwy ukrytej są funkcje gaussowskie wielu zmiennych 2.79, funkcje bicentralne 2.93, jak i funkcje definiujące hiperprostopadłościany.

Inicjalizacja architektury jest dokonywana za pomocą algorytmów klasteryzacji: poprzez histogramy, drzewa decyzyjne lub dendrogramy. Algorytmy te zostały opisane w podrozdziałach 3.3.6 i 3.3.5, i [42].

Po procesie inicjalizacji następuje etap estymacji położeń, jak i rozmyć (i innych) parametrów algorytmem rozmytym opisanym w [1, 44]. Główna część procesu estymacji oparta jest o analizę poszczególnych wektorów uczących neuronu, dla którego uzyskano największą aktywację w wyniku prezentacji danego wektora uczącego i neuronu najbliższego neuronowi, dla którego uzyskano największą aktywację.

Algorytm adaptacji umożliwia dodanie nowego neuronu, gdy są spełnione trzy warunki:

1. dla danego wektora uczącego $\mathbf{x}$, neuron, dla którego została uzyskana największa aktywacja N_M i neuron jemu najbliższy N_N, należą do różnych klas,
2. odległość, pomiędzy wektorem uczącym, a neuronem maksymalnie pobudzonym N_M, spełnia poniższą nierówność

$$\|\mathbf{x} - t_{N_M}\| > \sigma_{N_M}\sqrt{\ln 10} \tag{4.32}$$

3. maksymalna aktywacja, uzyskana dla neuronu N_M, jest mniejsza niż ustalona wartość Act_{min}

$$G_{N_M}(\mathbf{x}) < Act_{min} \tag{4.33}$$

Wspomniana już sieć MLP2LR [40], służąca do wyciągania reguł logicznych, również umożliwia dodawanie nowych neuronów, a samo kryterium niewystarczalności struktury sieci jest zdefiniowane w bardzo prosty sposób: sieć przestała się uczyć (wartość funkcji błędu przestała maleć) i poprawność klasyfikacji jest wciąż zbyt mała.

W podrozdziale 4.1.3 opisano m. in. metodę usuwania funkcji bazowych, opisaną przez Orr'a [143] za pomocą operacji na macierzy projekcji P. Wykonując inne, choć podobne, operacje na macierzy projekcji, można dodać nowy neuron. Odpowiednie zmiany w macierzy projekcji P_m pokazane są poniżej

$$P_{m+1} = P_m - \frac{P_m h_{m+1} h_{m+1}^T P_m}{\lambda_j + h_{m+1}^T P_m h_{m+1}} \tag{4.34}$$

W pracach [66, 67] Fritzke prezentuje sieć RBF, w której co λ kroków uczenia (prezentacji pojedynczego wektora uczącego) następuje dodanie nowego neuronu. Jednakże zanim nastąpi dodanie nowego neuronu, podczas procesu adaptacyjnego, wyznaczane są skumulowane błędy, jakie sieć popełnia w poszczególnych podobszarach, które reprezentują neurony warstwy ukrytej. Przy każdej prezentacji kolejnego wektora uczącego następuje kumulowanie błędu, doliczając sumaryczny błąd kwadratowy neuronowi s, który był najbliżej prezentowanego wektora uczącego:

$$\Delta err_s = \sum_{i=1}^{d} (F(\mathbf{x}^i) - y^i)^2 \tag{4.35}$$

$F(x)^i$ oznacza wartość i-tego wyjścia dla wektora x, a y^i wartość oczekiwaną na i-tym wyjściu. Po upływie każdych λ kroków uczenia następuje dodanie nowego neuronu w pobliżu neuronu, dla którego aktualny skumulowany błąd jest największy. Dokładniej pozycja nowego neuronu jest wyznaczona jako średnia pozycji neuronu, dla którego skumulowany błąd był największy i jednego z jego najbliższych sąsiadów.

Informacje o jeszcze innych modelach sieci samoorganizujących się można znaleźć w [69, 68]. Tu na szczególną uwagę zasługuje pierwsza praca, w której rozkład topologii sieci może podążać za zmieniającym się w czasie rozkładem danych.

Fiesler w [57] dokonał porównania ponad 30 modeli ontogenicznych. Zestawienie zawiera informacje o liczbie warstw, czy sieć umożliwia rozrastanie się, czy sieć może się kurczyć, czy metoda jest lokalna, jakie są dodatkowe warunki nakładane na sieć (np. czy startuje z pustej struktury), czy istnieją jakieś matematyczne dowody na zbieżność metody. Jednakże zestawienie nie wskazuje na jakiekolwiek wady, czy zalety poszczególnych modeli.

Poniżej zostanie pokazany inny sposób rozrastania się sieci typu RBF, wraz z kryterium wystarczalności sieci neuronowej RAN. Natomiast w podrozdziale 4.3 pokazana zostanie sieć IncNet, korzystająca z jeszcze sprawniejszego kryterium wystarczalności modelu, wykorzystującego informację czysto statystyczną, wyznaczaną na podstawie stanu filtra Kalmana.

4.2.1. Sieć RAN z przydziałem zasobów

Sieć z przydziałem zasobów, której oryginalna angielska nazwa to *Resource Allocation Network* (RAN), została zaproponowana przez Platt'a w [146]. Nieco później Kadirkamanathan i Niranjan [107, 103] pokazali, iż, przy pewnych założeniach, użyty w sieci RAN sekwencyjny sposób uczenia jest poprawny matematycznie. Podobnie zostało pokazane, że kryteria rozrostu sieci (nazwane geometrycznym kryterium rozrostu) są również poprawne przy założeniach, które nie stoją w opozycji do opisanego modelu przez Platta.

Uczenie sekwencyjne.

Sieć RAN można rozpatrywać, jako sekwencyjną metodę estymacji pewnego nieznanego gładkiego odwzorowania F^*, co oznacza, że celem jest wyznaczenie nowego *stanu modelu* $F^{(n)}$ na podstawie poprzedniego stanu modelu $F^{(n-1)}$ i nowej obserwacji $I^{(n)}$, która jest parą uczącą $< x_n, y_n >$ ($x \in \mathcal{R}^N, y \in \mathcal{R}$).

Tak określone zadanie można zapisać w postaci funkcji celu $\mathcal{F}$-projekcji:

$$F^{(n)} = \arg\min_{f \in \mathcal{H}} ||f - F^{(n-1)}|| \qquad F^{(n)} \in \mathcal{H}_n \tag{4.36}$$

gdzie $\mathcal{H}$ jest przestrzenią Hilberta funkcji:

$$\mathcal{H} = \{f : \|f\| < \infty\} \tag{4.37}$$

$\|\cdot\|$ jest normą L^2 zdefiniowaną przez

$$\|f\| = \int_{x \in \mathcal{D}} |f(\mathbf{x})|^2 d\mathbf{x} \qquad \mathcal{D} \subseteq \mathcal{R}^N \tag{4.38}$$

a $\mathcal{H}_n$ jest zbiorem funkcji określonych poprzez:

$$\mathcal{H}_n = \{f : f \in \mathcal{H} \ \wedge \ f(\mathbf{x}_n) = y_n\} \tag{4.39}$$

Warunek $F(\mathbf{x}_n) = y_n$ można zapisać jako iloczyn skalarny funkcji należących do przestrzeni $\mathcal{H}$:

$$< F, g_n >= \int_{x \in \mathcal{D}} F(\mathbf{x}) g(\mathbf{x} - \mathbf{x}_n) = y_n \tag{4.40}$$

gdzie $g(\cdot)$ jest funkcją impulsową. To pozwala na zapisanie funkcji celu jako $\mathcal{F}$-projekcji w postaci modelu *a posteriori*:

$$F^{(n)} = F^{(n-1)} + e_n \frac{g_n}{\|g_n\|^2} = F^{(n-1)} + e_n h_n \tag{4.41}$$

e_n jest błędem popełnionym przez a priori model $F^{(n-1)}$ w punkcie $\mathbf{x}_n$:

$$e_n = y_n - F^{(n-1)}(\mathbf{x}_n) \tag{4.42}$$

Takie rozwiązanie wprowadza bardzo ostrą funkcję impulsową w punkcie $\mathbf{x}_n$ do już istniejącego modelu $F^{(n-1)}$ tak, aby model w kolejnym stanie osiągał wartość y_n w punkcie $\mathbf{x}_n$. Kłóci się to z założeniem, iż szukane (estymowane) odwzorowanie jest funkcją gładką. To prowadzi do ograniczenia na gładkość funkcji $h_n(\cdot)$, która może być użyta w naszym rozwiązaniu (4.41)

$$h_n(\mathbf{x}_n) = 1 \qquad \wedge \qquad h_n(\mathbf{x}_n + a) = 0 \quad \text{dla } \|a\| > 0 \tag{4.43}$$

Dobrym przykładem takiej funkcji $h_n(\cdot)$ może być funkcja gaussowska z odpowiednio dobranym współczynnikiem rozmycia b

$$G(\mathbf{x}; \mathbf{t}, b) = e^{-\|\mathbf{x}-\mathbf{t}\|^2/b^2} \tag{4.44}$$

mamy wtedy:

$$G(\mathbf{x}; \mathbf{x}_n, b) = 1 \qquad \wedge \qquad G(\mathbf{x} + a; \mathbf{x}_n, b) \to 0 \quad \text{dla } \|a\| \to \infty \tag{4.45}$$

Podsumowując, korzystając z $\mathcal{F}$-projekcji i powyższego założenia o gładkości, możemy nasz model w stanie n przedstawić jako:

$$F^{(n)} = F^{(n-1)} + e_n G(\mathbf{x}; \mathbf{x}_n, b_0) \tag{4.46}$$

b_0 jest pewnym rozmyciem początkowym.

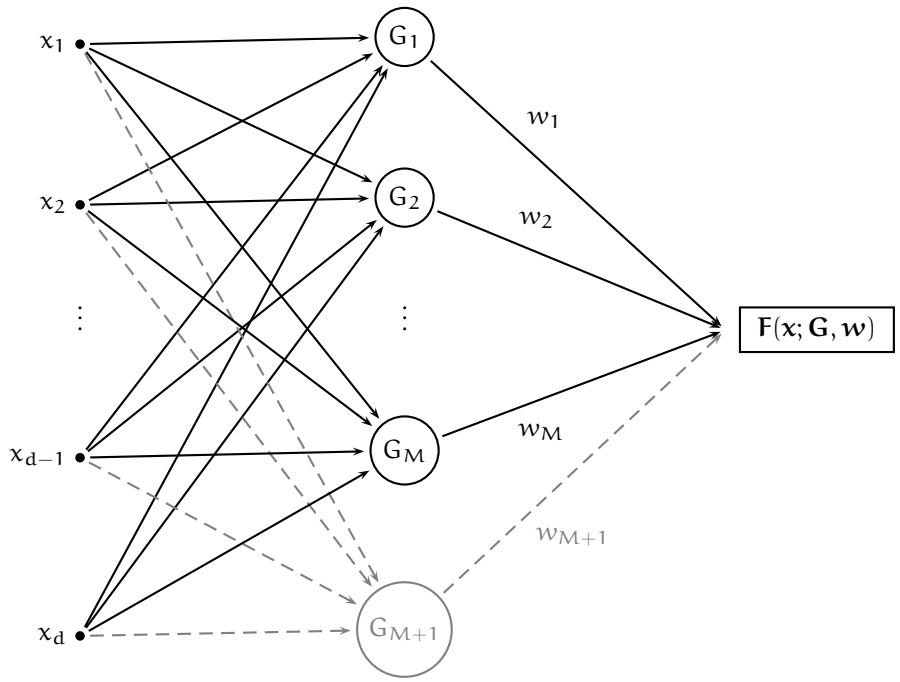
Przyjmując, że poprzedni model $F^{(n-1)}$ składał się z M funkcji bazowych (neuronów) mamy:

$$F^{(n)} = \sum_{i=1}^{M} w_i G(\mathbf{x}; \mathbf{t}_i, b_i) + e_n G(\mathbf{x}; \mathbf{x}_n, b_0) \quad (4.47)$$

$$= \sum_{i=1}^{M+1} w_i G(\mathbf{x}; \mathbf{t}_i, b_i) \quad (4.48)$$

z $\mathbf{t}_{m+1} = \mathbf{x}_n$ i $b_{M+1} = b_0$. Oznacza to rozbudowującą się sieć RBF (patrz rys. 4.2).

Z powyższych rozważań widać, iż sekwencyjny ciąg kolejnych modeli, wybierając odpowiednio małą wartość początkową b_0, może w rezultacie prowadzić do dowolnie małego ustalonego błędu interpolacji ϵ.

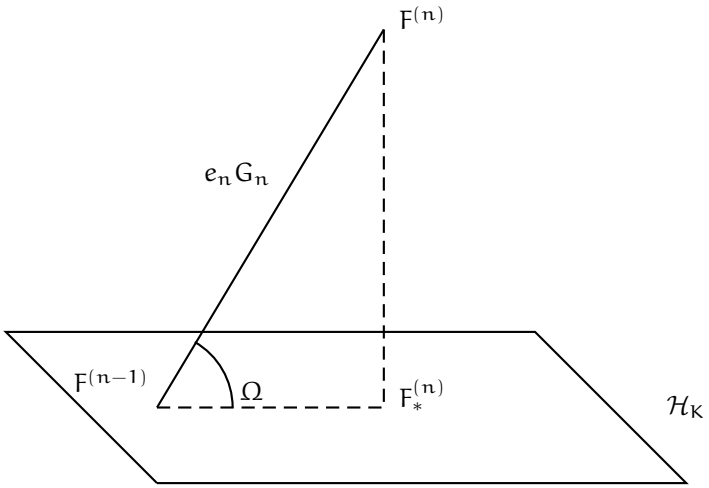


Rysunek 4.2: Sieć RAN z nową funkcją bazową G_{M+1}.

Oczywiście każdy kolejny stan k modelu $F^{(n+k)}$ nie powinien kosztować dodania nowej funkcji bazowej. Efekty takiej strategii były już opisywane w rozdziale 3, a szczególnie w podrozdziale 3.1 i 3.2. Dlatego też poniżej zostanie opisane geometryczne kryterium rozrostu sieci RAN, umożliwiające regulowanie złożoność sieci, w zależności od aktualnego jej stanu i napływających nowych obserwacji I_n.

Geometryczne Kryterium Rozrostu

Geometryczne Kryterium Rozrostu umożliwia kontrolę złożoności modelu podczas sekwencyjnego uczenia. Kryterium musi odpowiedzieć na pytanie: czy model $F^{(n)}$ wystarczy, aby dostatecznie dobrze estymować nową obserwację I_n, czy też jest niewystarczający i wymaga dodania nowej funkcji bazowej.



Rysunek 4.3: Zależności pomiędzy modelami a posteriori $F_*^{(n)}$ i $F^{(n)}$ (odpowiednio z przestrzeni $\mathcal{H}_M$ i $\mathcal{H}_{M+1}$) względem a priori modelu $F^{(n-1)}$.

Aby odpowiedzieć na powyższe pytanie trzeba prześledzić sytuację, w której model nie zostaje powiększony o nowy neuron i sytuację, w której model zostaje rozbudowany o nową funkcję bazową (patrz rysunek 4.3). Modele *a priori* $F^{(n-1)}$ i *a posteriori* $F_*^{(n)}$ (nie powiększony o nową funkcję bazową) należą do przestrzeni $\mathcal{H}_M$. Natomiast model *a posteriori* $F^{(n)}$ powiększony o nową funkcję bazową należy już do przestrzeni $\mathcal{H}_{M+1}$. Model $F_*^{(n)}$ jest projekcją modelu $F^{(n)}$ do przestrzeni $\mathcal{H}_M$.

Tym samym odpowiedź na powyższe pytanie zawarta jest w wielkości odległości $||F^{(n)} - F_*^{(n)}||$. Jeśli jest ona większa od pewnego ϵ oznacza to, iż przestrzeń funkcji $\mathcal{H}_M$ jest niewystarczająca do uzyskania akceptowalnego poziomu estymacji:

$$||F^{(n)} - F_*^{(n)}|| > \epsilon \tag{4.49}$$

Poza tym mamy też następującą własność (porównaj z rys. 4.3):

$$||F^{(n)} - F_*^{(n)}|| = |e_n| \cdot ||G_n|| \sin(\Omega) \tag{4.50}$$

Ponieważ $||G_n||$ zależy tylko od stałej początkowego rozmycia b_0, a z kolei kąt Ω może przyjmować wartości od 0 do $\pi/2$, można uprościć nierówność (4.49) do po-

niższych kryteriów geometrycznych:

$$e_n > e_{min} \tag{4.51}$$
$$\Omega > \Omega_{min} \tag{4.52}$$

Pierwsza z powyższych nierówność to *kryterium predykcji błędu*. Ocenia ono błąd interpolacyjny. Druga z powyższych nierówności to *kryterium kąta*, które ocenia na ile funkcja bazowa G_n jest ortogonalna (duży kąt) do funkcji bazowych z przestrzeni funkcji $\mathcal{H}_M$.

W ogólnym przypadku wyznaczenie poszczególnych kątów jest trudne, ale można dokonać pewnych przybliżeń, na przykład przyjmując równe rozmycia funkcji G_n i funkcji G_i. Wtedy kąt pomiędzy funkcjami G_n i G_i jest równy:

$$\Omega_i = \cos^{-1}\left(\exp\left(-\frac{1}{2b_0^2}||\mathbf{x}_n - \mathbf{t}_i||^2\right)\right) \tag{4.53}$$

Korzystając z powyższej własności można przepisać kryterium kąta (4.52) do postaci:

$$\sup_i G_i(\mathbf{x}_n) \leqslant \cos^2(\Omega_{min}) \tag{4.54}$$

co ze względu na monotoniczne nachodzenie się funkcji gaussowskiej przy o oddalaniu się punktu $\mathbf{x}_n$ od centrum funkcji G_i, można zastąpić przez:

$$\inf_i ||\mathbf{x}_n - \mathbf{t}_i|| \geqslant \epsilon \tag{4.55}$$

Powyższe kryterium przeradza się w *kryterium odległości* i tak naprawdę pokazuje, że porównaniu podlega odległość pomiędzy punktem $\mathbf{x}_n$ i centrum najbliższej funkcji bazowej, a wartości ϵ

$$\epsilon = b_0\sqrt{2\log(1/\cos^2\Omega_{min})} \tag{4.56}$$

Można też określić optymalne rozmycie dla nowej funkcji bazowej, które będzie maksymalnie duże, ale będzie też spełniać kryterium kąta (4.52):

$$b_n = \frac{||\mathbf{x}_n - \mathbf{t}_k||}{\sqrt{2\log(1/\cos^2\Omega_{min})}} \qquad k = \arg\min_k ||\mathbf{x}_n - \mathbf{t}_k|| \tag{4.57}$$

Podsumowując, stwierdzić można, że nowy neuron jest dodawany, gdy spełnione jest kryterium predykcji błędu (4.51) i kryterium odległości (4.55).

Adaptacja sieci RAN

Gdy nie jest spełnione kryterium predykcji błędu (4.51) lub kryterium odległości (4.55) następuje adaptacja wolnych parametrów sieci

$$F(\mathbf{x}) = \sum_{i=1}^{M} w_i G(\mathbf{x}; \mathbf{t}_i, b_i) + w_0 \tag{4.58}$$

Wagi i położenia centrów funkcji bazowych

$$\mathbf{p}^{(n)} = [w_0, w_1, \ldots, w_M, \mathbf{t}_1^T, \mathbf{t}_2^T, \ldots, \mathbf{t}_M^T]^T \tag{4.59}$$

są uczone algorytmem LMS:

$$\mathbf{p}^{(n)} = \mathbf{p}^{(n-1)} + \eta e_n \mathbf{d}_n \tag{4.60}$$

gdzie $\mathbf{d}_n$ jest gradientem funkcji $F(\mathbf{x}_n)$ po parametrach $\mathbf{p}$:

$$\mathbf{d}_n = \frac{\partial f(\mathbf{x}_n)}{\partial \mathbf{p}_{n-1}} = \left[1, G_1(\mathbf{x}_n), \ldots, G_M(\mathbf{x}_n), G_1(\mathbf{x}_n)\frac{2w_1}{b_1^2}(\mathbf{x}_n - \mathbf{t}_1)^T, \ldots, G_M(\mathbf{x}_n)\frac{2w_M}{b_M^2}(\mathbf{x}_n - \mathbf{t}_M)^T\right] \tag{4.61}$$

Zastępując η przez $\frac{\eta}{||\mathbf{x}_n||^2}$, uzyskuje się znormalizowaną wersję algorytmu LMS.

W praktyce również odległość minimalna ϵ podlega zmianom podczas procesu uczenia. Początkowa wartość ϵ jest równa ϵ_{max}, a następnie podlega iteracyjnym zmianom

$$\epsilon = \epsilon_{max}\, \gamma^n \tag{4.62}$$

($0 < \gamma < 1$), aż do osiągnięcia wartości ϵ_{min}.

4.3. Sieć IncNet ze statystyczną kontrolą złożoności sieci

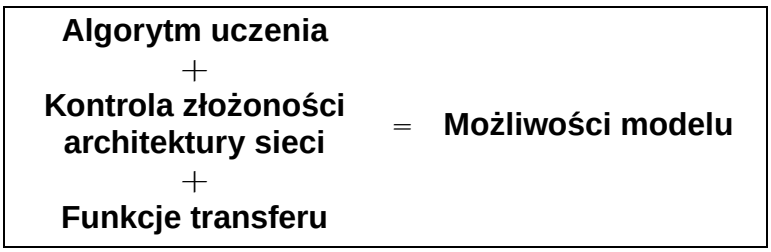
Praktycznie wszystkie przedstawione powyżej modele ontogeniczne zawsze nakładają dość drastyczne uproszczenia dla całości procesu uczenia. Wynikają one z koncentracji uwagi nad częścią rozwiązywanego problemu, a nie na całości. Dla przykładu algorytmy OBD [32, 31], OBS[82, 83] czy FSM [1, 44, 49, 45], zezwalają na usuwanie neuronów praktycznie po zakończeniu właściwego procesu uczenia. Z kolei inne modele dodające człon regularyzacyjny, wymuszający niskie wartości wag, co często prowadzi do zbyt małych wartości niemal wszystkich wag, a z kolei inne metody regularyzacji wymagają dobrego dobrania parametru (lub parametrów) na podstawie wiedzy a priori, co nie zawsze jest proste w praktyce. Rozważania w poniższych podrozdziałach pokażą, iż można zdefiniować inne kryteria, które umożliwią usuwanie neuronów praktycznie z iteracji na iterację, czyli gdy tylko okaże się to korzystne dla procesu adaptacji, a nie co epokę lub po zakończeniu procesu uczenia tak jak jest to realizowane w innych algorytmach.

Algorytmy, które mogą modyfikować swoją architekturę podczas uczenia, nie czynią tego w zbyt optymalny sposób. Podobnie zresztą jak i modele, które są zdolne do eliminacji neuronów lub wag, nierzadko czeka się, aż sieć neuronowa kompletnie zakończy uczenie i dopiero wtedy próbuje się dokładać nowe neurony lub wagi. Czy też, tak, jak jest to w sieci RAN (por. podrozdział 4.2.1), niekoniecznie spełnienie

kryteriów (kryterium predykcji błędu i kryterium odległości) opisanych równaniami (4.51 i 4.55), odpowiada sytuacjom, w których sieć nie jest zdolna do uwzględnienia nowej obserwacji. Może to odpowiadać zbyt szybkiej prezentacji danej w dość odległym rejonie, do którego i tak jakaś z funkcji bazowych zostałaby *przyciągnięta* w miarę postępowania procesu uczenia. Spotyka się również naiwną taktykę dokładania nowych funkcji bazowych do sieci RBF co stałą liczbę kroków uczenia. Dlatego też w jednym z poniższych podrozdziałach zaprezentowane zostanie alternatywne i znacznie efektywniejsze kryterium.

Jeszcze inną wadą już nie tylko powyżej opisanych systemów ontogenicznych lecz znacznej części sieci neuronowych jest lekceważenie istotności wyboru funkcji transferu, które silnie determinują możliwości generalizacji modeli adaptacyjnych.

Głównym celem całościowego systemu IncNet, zaprezentowanego w kolejnych Podrozdziałach, stało się stworzenie takiego modelu, który będzie korzystał z efektywnego algorytmu uczenia i jednocześnie będzie zdolny do auto-kontroli złożoności architektury sieci podczas procesu uczenia, tak aby złożoność architektury sieci odzwierciedlała złożoność zaprezentowanej dotychczas części zbioru wektorów uczących. Oczywiście tak postawione zadanie jest bardzo trudnym problemem i nie jest możliwe jego rozwiązanie w 100%.



Proponowanym rozwiązaniem problemu może być system, który będzie łączył w sobie poniższe cechy:

Uczenie: wykorzystanie efektywnego filtra Kalmana do adaptacji swobodnych parametrów sieci,

Kontrola złożoności sieci: wykorzystanie kryteriów kontrolujących rozbudowywanie się sieci, jak i kurczenie się sieci, poprzez dodawanie, usuwanie i łączenie się neuronów,

Elastyczne funkcje transferu: użycie bicentralnych funkcji transferu umożliwiają znacznie efektywniejszą estymacje, a w efekcie umożliwia adaptacje bardziej złożonych problemów.

System, składający się z takich funkcji, jest zdolny do efektywnego uczenia sieci neuronowej przy jednoczesnej kontroli użyteczności każdego z podelementów (funkcji

bazowych formujących przestrzeń modelowania) i wystarczalności całości systemu do estymacji nowych obserwacji.

Poniżej zostaną omówione poszczególne elementy, składające się na sieć neuronową Incremental Network (IncNet).

4.3.1. Struktura sieci i funkcje transferu

Struktura sieci IncNet jest praktycznie taka sama jak sieci RBF (czy też RAN). Różni je jednakże fakt, iż struktura sieci RBF jest statyczna i nie podlega zmianom podczas uczenia, natomiast sieć IncNet jest dynamiczna i może się kurczyć i rozrastać. Widać to przy porównaniu praktycznie identycznych równań dla sieci RBF (3.2) i sieci IncNet (4.63), i częściowo różnych schematów tych sieci przedstawionych na rysunkach 3.1 i 4.4.

$$f(\mathbf{x}; \boldsymbol{w}, \mathbf{p}) = \sum_{i=1}^{M} w_i G_i(\mathbf{x}, \mathbf{p}_i). \tag{4.63}$$

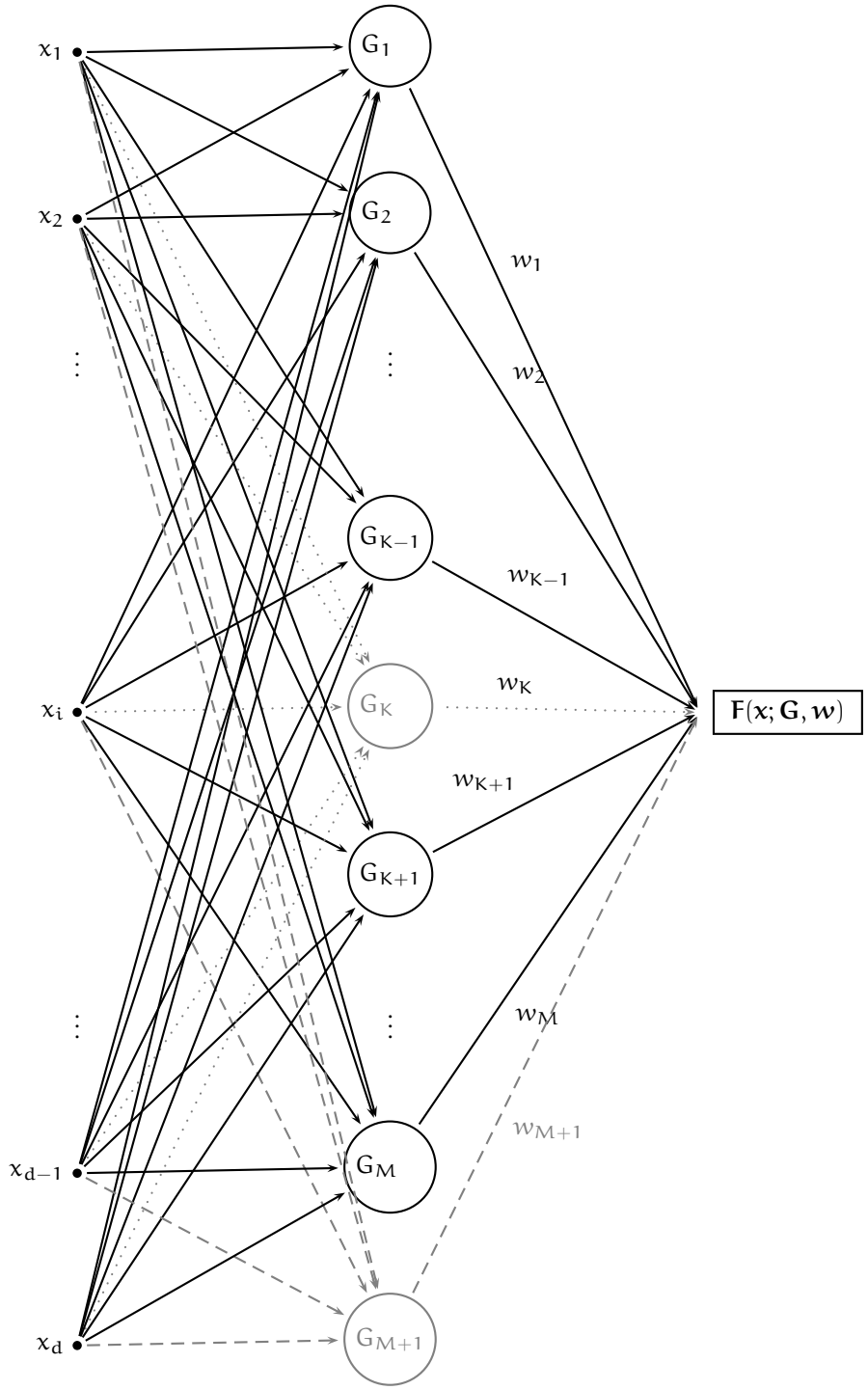
W standardowej sieci RBF, w sieci RAN i w pierwszej wersji sieci IncNet [104], jako funkcje bazowe używane były funkcje gaussowskie. Znaczny postęp uzyskano stosując w miejsce funkcji gaussowskich funkcje bicentralne, które powstają z produktu par funkcji sigmoidalnych. Pozwalają one na estymacje znacznie bardziej zróżnicowanych powierzchni przy użyciu mniejszej liczby funkcji bazowych. Standardowa funkcja bicentralna umożliwia niezależną kontrolę rozmycia jak i skosu w każdym wymiarze niezależnie. Zaproponowane różne rozszerzenia funkcji bicentralnych umożliwiają wzbogacenie możliwości o rotację gęstości w wielowymiarowej przestrzeni, czy uzyskiwanie semi-lokalnych gęstości, jak i desymetryzacji w podwymiarach. Szczegółowo funkcje bicentralne zostały już opisane i zilustrowane w podrozdziałach 2.4.6 i 2.4.7.

4.3.2. Rozszerzony filtr Kalmana

Jak przedstawiono w podrozdziale 4.2.1, na potrzeby uczenia sekwencyjnego funkcję błędu należy zdefiniować nieco inaczej niż robi się to dla typowych sieci neuronowych, czyli w postaci równania (3.14). Definicja błędu dla uczenia sekwencyjnego sieci RAN przez $\mathcal{F}$-projekcję, zdefiniowanej równaniem (4.36), może zostać uogólniona do minimalizacji funkcji:

$$E_{sq} = \int_{\mathcal{D}} |F^{(n)}(\mathbf{x}) - F^{(n-1)}(\mathbf{x})|^2 p^{(n-1)} d\mathbf{x} \; + \; \frac{1}{n-1} |y_n - F^{(n)}(\mathbf{x_n})|^2 \tag{4.64}$$

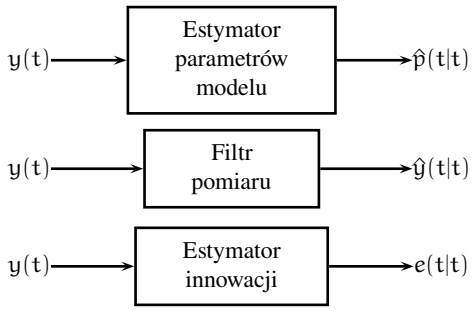
gdzie $n-1$ i n odpowiadają określeniu kolejnych stanów modelu adaptacyjnego, $p^{(n-1)}$ jest rozkładem prawdopodobieństwa ostatnich $n-1$ obserwacji (wektorów wejściowych). Powyższa funkcja błędu została użyta do algorytmu RNLS (*ang. recursive nonlinear least squares*) w pracach Kadirkamanathana i Niranjana [103, 105].



Rysunek 4.4: Struktura sieci IncNet. Sieć umożliwia dodawanie nowej funkcji bazowej G_{M+1} lub usuwanie pewnej funkcji bazowej G_K, jak również konkatenację neuronów.

Minimalizacja powyższej funkcji błędu może być aproksymowana przez użycie rozszerzonej wersji algorytmu filtru Kalmana (EKF) [25, 86, 155, 114]. Prowadzi to do zastąpienia algorytmu LMS algorytmem EKF do estymacji parametrów adaptacyjnych sieci. Pierwszy raz filtr EKF do estymacji parametrów sieci neuronowych został użyty przez Singhala i Wu w [161] dla sieci MLP; natomiast dla sieci typu RBF przez Kadirkamanathana w [103]. Z kolei z najnowszych i ciekawych zastosowań filtru EKF należy wspomnieć prace Freitas'a i Niranjana [62] do sekwencyjnej wersji modelu Support Vector Machines (por. podrozdział 3.5.4) i prace [63] prezentującą próbę modelowania giełdowego rynku finansowego.

Filtr Kalmana jest estymatorem trzech różnych wyjść dla danej obserwacji (wektora danych), co do której zakłada się, że jest obarczona pewnym szumem, o zerowej wartości oczekiwanej, z pewnym niezerowym odchyleniem standardowym. Pierwsze z wyjść stanowi estymator parametrów modelu. To właśnie główny element uczenia sieci. Kolejne dwa wyjścia to filtr pomiaru i estymator innowacji (lub błędu). Wyjścia te będą wykorzystywane do estymacji i do kontroli złożoności całości modelu.



Filtr EKF prowadzi estymacje parametrów *a posteriori* modelu $\mathbf{p}_n$, w oparciu o ich poprzedni stan *a priori* $\mathbf{p}_{n-1}$ oraz błąd modelu e_n:

$$e_n = y_n - f(\mathbf{x}_n; \mathbf{p}_{n-1}) \tag{4.65}$$

i wartości wektora wzmocnienia Kalmana (*ang. Kalman gain*), które są pochodną macierzy kowariancji błędu a priori $\mathbf{P}_{n-1}$:

$$\mathbf{p}_n = \mathbf{p}_{n-1} + e_n \mathbf{k}_n \tag{4.66}$$

Wektor Kalmana $\mathbf{k}_n$ jest wyznaczany przez:

$$\mathbf{k}_n = \mathbf{P}_{n-1}\mathbf{d}_n / R_y \tag{4.67}$$

$\mathbf{d}_n$ jest wektorem gradientu funkcji modelu względem parametrów modelu:

$$\mathbf{d}_n = \frac{\partial f(\mathbf{x}_n; \mathbf{p}_{n-1})}{\partial \mathbf{p}_{n-1}} \tag{4.68}$$

natomiast R_y jest całkowitą wariancją modelu wyznaczaną poprzez:

$$R_y = R_n + \mathbf{d}_n^T \mathbf{P}_{n-1} \mathbf{d}_n \tag{4.69}$$

z kolei R_n określa wariancję szumu pomiarów, kontroluje proces regularyzacji. Estymacja a priori macierzy kowariancji błędu przebiega zgodnie z równaniem:

$$\mathbf{P}_n = [\mathbf{I} - \mathbf{k}_n \mathbf{d}_n^T]\mathbf{P}_{n-1} + Q_0(n)\mathbf{I} \tag{4.70}$$

$\mathbf{I}$ jest macierzą jednostkową. Człon $Q_0(n)\mathbf{I}$ spełnia rolę (drobnego) losowego skoku w kierunku gradientu, wprowadzając małą perturbację w procesie adaptacji parametrów modelu i zapobiegając zbyt szybkiej zbieżności, a czasami umożliwia *ucieczkę* z lokalnych minimów. $Q_0(n)$ może być bardzo małym dodatnim skalarem bądź funkcją monotonicznie malejącą, przyjmującą bardzo małe dodatnie wartości (stopniowe zastyganie). Dla przykładu:

$$Q_0(n) = \begin{cases} Q_0 & n = 1 \\ Q_0(n-1) \cdot Q_{des} & n > 1 \wedge Q_0(n-1) \cdot Q_{des} > Q_{min} \\ Q_{min} & n > 1 \wedge Q_0(n-1) \cdot Q_{des} \leqslant Q_{min} \end{cases} \tag{4.71}$$

gdzie Q_0 jest wartością początkową dla $Q_0(n)$, Q_{des} definiuje szybkość zmniejszania pobudzania ($Q_{des} > 1$, zazwyczaj ok. 0.9988), a Q_{min} określa minimalną wartość $Q_0(n)$.

Wielką różnicę pomiędzy algorytmem EKF i LMS można zauważyć porównując ich równania (4.66) i (4.60), adaptacji parametrów p. W miejscu członu $\eta \mathbf{d}_n$ z algorytmu LMS, w filtrze EKF znajduje się wektor wzmocnienia Kalmana $\mathbf{k}_n$, który, jak widać z kolei w równaniu (4.67), wyznacza szybkość adaptacji każdego z parametrów nie tylko w zależności od wektora gradientu $\mathbf{d}_n$ (jak to jest w algorytmie LMS czy stochastycznym spadku gradientu), lecz również w oparciu o macierz kowariancji $\mathbf{P}_{n-1}$. Właśnie to prowadzi do znacznie efektywniejszego procesu uczenia, radykalnie zmniejszając liczbę iteracji potrzebną do uzyskania zbieżności.

Słuszności wyboru filtra EKF jako algorytmu adaptacji parametrów modelu dowiodły już pierwsze jego zastosowania do adaptacji sieci RAN (RAN-EKF). W pracach [107, 108, 106] przedstawione zostały rezultaty użycia sieci RAN-EKF do aproksymacji funkcji Hermita (por. rozdział 5), przewidywania wartości szeregów czasowych, czy do klasyfikacji samogłosek. Rezultaty pokazują, iż użycie algorytmu EKF jest znacząco efektywniejsze i pozwala uzyskać większą generalizację często korzystając z mniejszej liczby funkcji bazowych.

4.3.3. Szybka wersja rozszerzonego filtru Kalmana

Macierz kowariancji $\mathbf{P}_n$, w miarę przybywania nowych neuronów, może urosnąć do sporych rozmiarów, gdy sieć uczy się klasyfikacji, bądź aproksymacji złożonych danych. Należy pamiętać, iż liczba elementów macierzy $\mathbf{P}_n$ to kwadrat liczby parametrów adaptacyjnych. To właśnie może okazać się zbyt obliczeniochłonnym procesem.

Rozsądną decyzją okazało się zredukowanie sporej części macierzy kowariancji przyjmując, że korelacje pomiędzy parametrami różnych neuronów nie są tak istotne, jak korelacje pomiędzy parametrami tego samego neuronu. Takie uproszczenie prowadzi do sporych zmian macierzy $\mathbf{P}_n$ upraszczając ją do macierzy $\widetilde{\mathbf{P}}_n$, która tak naprawdę składa się z diagonalnego łańcucha podmacierzy $\widetilde{\mathbf{P}}_n^k$ (elementy poza diagonalnym łańcuchem są równe zeru):

$$\widetilde{\mathbf{P}}_n = \begin{bmatrix} \widetilde{\mathbf{P}}_n^1 & 0 & \dots & 0 & 0 \\ 0 & \widetilde{\mathbf{P}}_n^2 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \widetilde{\mathbf{P}}_n^{M-1} & 0 \\ 0 & 0 & \dots & 0 & \widetilde{\mathbf{P}}_n^M \end{bmatrix} \tag{4.72}$$

Podmacierze $\widetilde{\mathbf{P}}_n^k$ ($k = 1, 2, \dots, M$) są macierzami kowariancji związanymi z parametrami adaptacyjnymi k-tego neuronu.

Dzięki takiemu uproszczeniu, co prawda rezygnujemy z części informacji macierzy $\mathbf{P}_n$, ale za to znacznie zmniejszamy jej złożoność. Liczba parametrów macierzy $\mathbf{P}_n$ wynosi $n \cdot M \times n \cdot M$ (n to rozmiar przestrzeni wejściowej, a M liczba neuronów warstwy ukrytej), natomiast macierz $\widetilde{\mathbf{P}}_n$ ma $m^2 M$ istotnych parametrów. Tym samym dla danego problemu $\mathcal{P}$ złożoność macierzy $\mathbf{P}_n$ z $O(M^2)$ redukuje się do $O(M)$ dla macierzy $\widetilde{\mathbf{P}}_n$ (m jest stałe ze względu na $\mathcal{P}$).

Powyższa redukcja prowadzi również do przedefiniowania równań (4.65–4.70) filtra EKF do postaci:

$$e_n = y_n - f(\mathbf{x}_n; \mathbf{p}_{n-1}) \qquad i = 1, \dots, M \tag{4.73}$$

$$\mathbf{d}_n^i = \frac{\partial f(\mathbf{x}_n; \mathbf{p}_{n-1})}{\partial \mathbf{p}_{n-1}^i} \tag{4.74}$$

$$R_y = R_n + \mathbf{d}_n^{1\,T} \widetilde{\mathbf{P}}_{n-1}^1 \mathbf{d}_n^1 + \dots + \mathbf{d}_n^{M\,T} \widetilde{\mathbf{P}}_{n-1}^M \mathbf{d}_n^M \tag{4.75}$$

$$\mathbf{k}_n^i = \widetilde{\mathbf{P}}_{n-1}^i \mathbf{d}_n^i / R_y \tag{4.76}$$

$$\mathbf{p}_n^i = \mathbf{p}_{n-1}^i + e_n \mathbf{k}_n^i \tag{4.77}$$

$$\widetilde{\mathbf{P}}_n^i = [\mathbf{I} - \mathbf{k}_n^i \mathbf{d}_n^{i\,T}] \widetilde{\mathbf{P}}_{n-1}^i + Q_0(n)\mathbf{I} \tag{4.78}$$

4.3.4. Kryterium wystarczalności modelu

> Kiedy aktualna architektura sieci nie jest już wystarczalna, aby akumulować nowe informacje?

Odpowiedź na to pytanie nie jest trywialna, czego dowodem mogą chyba być przedstawione poprzednio przykłady rozwiązania tego problemu. Powiększanie sieci o kolejne wagi czy neurony co pewien czas jest raczej naiwnym podejściem. Natomiast kryteria, których wyznaczenie wymaga przejrzenia (przeanalizowania) zachowania modelu dla wszystkich wektorów treningowych, można realizować jedynie od czasu do czasu w

trakcie procesu adaptacyjnego (raczej nie częściej niż co epokę). Algorytmy bazujące (niemal) jedynie na wielkości popełnionego błędu dla danego wektora treningowego również nie są pozbawione wad, ponieważ podczas procesu uczenia nie możemy w pełni ufać modelowi, który przecież podlega uczeniu — sieć nie jest w pełni wiarygodna. Ta konkluzja zbliża do zdefiniowania ogólnego kryterium, które powinno na jednej szali położyć wielkość błędu, a na drugiej stopień naszego zaufania do aktualnego stanu sieci:

$$\frac{\text{błąd}}{\text{niepewność modelu}} < \alpha(M) \tag{4.79}$$

$\alpha(M)$ jest progiem zależnym od wielkości struktury modelu (liczby stopni swobody).

Statystyczną miarą niepewności dla *całości* rozpatrywanego modelu jest jego wariancja, która składa się z wariancji modelu (sieci neuronowej) i szumu danych (np. niedoskonałości pomiarów). Przyjmijmy iż wariancja szumu danych jest równa σ^2_{ns}, a wariancja samego modelu $Var(f(\mathbf{x};\mathbf{p})) = \sigma^2_f(\mathbf{x})$.

Zakładając, iż błąd interpolacji pewnego nieznanego odwzorowania ma rozkłada normalny, można oszacować, czy błąd leży w pewnym przedziale ufności, wyznaczonym przez niepewność modelu i szumu danych z pewnym, ustalonym stopniem zaufania. Wtedy hipotezę $\mathcal{H}_0$, iż aktualny model jest wystarczający, można zapisać jako:

$$\mathcal{H}_0: \quad \frac{e^2}{Var[f(\mathbf{x};\mathbf{p}) + \eta]} = \frac{e^2}{\sigma^2_y(\mathbf{x}) + \sigma^2_{ns}} < \chi^2_{M,\theta} \tag{4.80}$$

gdzie $\chi^2_{M,\theta}$ jest rozkładem chi-kwadrat z progiem ufności $\theta\%$ i M stopniami swobody (M – liczba funkcji bazowych). Z kolei e jest błędem: $y - f(\mathbf{x};\mathbf{p})$ (por. (4.65)).

Gdy hipoteza $\mathcal{H}_0$ jest spełniona oznacza to, że bieżąca struktura sieci jest wystarczająca przy ustalonym stopniu zaufania. W przeciwnym przypadku bieżący model jest niewystarczający i należy rozszerzyć jego pojemność. W naszym przypadku oznacza to dodanie nowej funkcji bazowej do warstwy ukrytej.

Używając filtru EKF do estymacji parametrów modelu mamy automatycznie wyznaczoną całkowitą wariancję modelu:

$$R_y = Var[f(\mathbf{x};\mathbf{p}) + \sigma^2_{ns}], \tag{4.81}$$

R_y wyznaczane jest równaniem (4.69). Prowadzi to do ostatecznej definicji wystarczalności modelu:

$$\mathcal{H}_0: \quad \frac{e^2_n}{R_y} < \chi^2_{n,\theta} \tag{4.82}$$

Jeśli hipoteza $\mathcal{H}_0$ jest spełniona, sieć IncNet kontynuuje proces adaptacji parametrów, używając do tego filtru EKF (lub jego szybkiej wersji). W przeciwnym przypadku

należy dodać nową funkcję bazową $G_{M+1}(\cdot)$ (neuron) do warstwy ukrytej z pewnymi wartościami początkowymi. Dla gaussowskich funkcji bazowych mamy:

$$\begin{aligned} w_{M+1} &= e_n = y_n - f(\mathbf{x}_n; \mathbf{p}_n) && (4.83)\\ \mathbf{t}_{M+1} &= \mathbf{x}_n && (4.84)\\ b_{M+1} &= b_0 && (4.85)\\ \mathbf{P}_n &= \begin{bmatrix} \mathbf{P}_n & 0 \\ 0 & P_0\mathbf{I} \end{bmatrix} && (4.86) \end{aligned}$$

stała b_0 jest początkową wartością definiującą rozmycie, a P_0 określa nowe parametry diagonalne macierzy kowariancji (P_0 zazwyczaj jest równe 1).

Dla funkcji bicentralnych dodanie nowego neuronu wygląda podobnie:

$$\begin{aligned} w_{M+1} &= e_n = y_n - f(\mathbf{x}_n; \mathbf{p}_n) && (4.87)\\ \mathbf{t}_{M+1} &= \mathbf{x}_n && (4.88)\\ \mathbf{b}_{M+1} &= \mathbf{b}_0 && (4.89)\\ \mathbf{s}_{M+1} &= \mathbf{s}_0 && (4.90)\\ \mathbf{P}_n &= \begin{bmatrix} \mathbf{P}_n & 0 \\ 0 & \mathbf{P}_0\mathbf{I} \end{bmatrix} && (4.91) \end{aligned}$$

wektory stałych $\mathbf{b}_0$ i $\mathbf{s}_0$ definiują początkowe wartości rozmyć i skosów w wielowymiarowej przestrzeni.

Tak zdefiniowany statystyczny test wystarczalności modelu jest bardzo efektywny, a dla jego zweryfikowania nie potrzeba wyznaczać żadnych dodatkowych elementów. Co najważniejsze, taki test można stosować w każdej iteracji algorytmu uczenia, tj. dla każdej prezentacji wektora treningowego. Jak się okazało w praktyce tak zdefiniowany test wystarczalności prowadzi do lepszego wykorzystania posiadanych już funkcji bazowych przez model, jak i do uzyskania lepszego poziomu generalizacji. Widać to porównując sieć IncNet z siecią RAN [146], czy RAN-EKF [107], co można zobaczyć w rozdziale 5 (a szczególnie w podrozdziale 5.4) jak i pracach [99, 101, 104].

4.3.5. Usuwanie neuronów

Podczas procesu uczenia często dochodzi do sytuacji, w której pewne wagi czy neurony przestają odgrywać istotną rolę. Takie okoliczności sprawiają, iż model i tak musi prowadzić adaptację takich neuronów pomimo, iż nie są one przydatne. Co więcej mogą one być przyczyną przeuczenia się naszego modelu.

W powyższej części rozdziału o ontogenicznych sieciach neuronowych można było zauważyć, że prawie żaden model nie był zdolny do powiększania i zmniejszania swojej struktury, a jeśli już mógł zmniejszać i zwiększać swoją strukturę, to nie mógł robić obu czynności podczas uczenia i zazwyczaj usuwanie neuronów dokonywane było po procesie uczenia. Umożliwienie modelowi zwiększania i zmniejszania swojej struktury **podczas** procesu uczenia sprawia, że model może (gdy potrzebuje) powiększyć swoją strukturę jak również może dokonać jej zmniejszenia poprzez usunięcie

części struktury lub zastąpienie pewnej części struktury inną o mniejszej złożoności. Taki mechanizm regulacji złożoności modelu jest znacznie bardziej racjonalny, niż umożliwienie jedynie powiększania albo zmniejszania struktury modelu.

Jak pokaże poniższa cześć podrozdziału można tak zdefiniować algorytm sprawdzania przydatności poszczególnych neuronów, aby można było zwiększać, jak i zmniejszać architekturę sieci, dostosowując złożoność modelu uczącego do złożoności napływających do modelu danych.

Poniżej przedstawiony algorytm opiera się o analizę wartości współczynników istotności (przydatności) dla neuronów sieci. Tym samym, w pierwszym etapie należy ustalić sposób wyznaczania współczynników istotności dla poszczególnych neuronów warstwy ukrytej.

Zakładając, iż funkcje gęstości, opisywane przez poszczególne funkcje transferu neuronów warstwy ukrytej, są podobne (np. zlokalizowane i wypukłe), współczynniki istotności poszczególnych neuronów warstwy ukrytej można zdefiniować przez stosunek wielkości wagi do wariancji tej wagi. Taki stosunek faworyzuje silne wagi (tj. wagi o istotnym wpływie), których tendencje zmian wartości są małe (tj. wagi *nauczone*). W ten sposób zdefiniowane współczynniki można zapisać matematycznie:

$$s_i = \frac{w_i^2}{\sigma_{w_i}} \tag{4.92}$$

gdzie σ_{w_i} oznacza wariancję i-tej wagi (związaną z i-tym neuronem warstwy ukrytej).

Oczywiście jeśli miałoby dojść do usunięcia jakiegokolwiek neuronu, to należy wybrać ten, dla którego współczynnik istotności będzie najmniejszy:

$$L_1 = \min_i s_i = \min_i \frac{w_i^2}{\sigma_{w_i}} \tag{4.93}$$

Używając rozszerzonego filtra Kalmana jako estymatora parametrów modelu do wyznaczania wariancji σ_{w_i}, można użyć macierzy kowariancji $\mathbf{P}_n$.

W tym celu najpierw przyjmijmy, iż parametry sieci w wektorze parametrów p_n są uszeregowane w poniższy sposób:

$$\mathbf{p}_n = [w_1, \ldots, w_M, \ldots]^T \tag{4.94}$$

czyli pierwsze są wagi pomiędzy warstwą drugą i wyjściową, a pozostałe parametry znajdują się w dalszej części wektora p_n. Wtedy macierz kowariancji $\mathbf{P}_n$ wygląda tak:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_w & \mathbf{P}_{wv} \\ \mathbf{P}_{wv}^T & \mathbf{P}_v \end{bmatrix} \tag{4.95}$$

gdzie $\mathbf{P}_w$ jest macierzą kowariancji pomiędzy wagami, macierz $\mathbf{P}_{wv}$ jest macierzą kowariancji pomiędzy wagami i innymi parametrami, a macierz $\mathbf{P}_v$ jest macierzą kowariancji tylko pomiędzy innymi parametrami (tj. bez wag).

Korzystając z powyższego ułożenia parametrów w macierzy $\mathbf{P}$ wzór (4.93) można sprowadzić do postaci:

$$L_1 = \min_i s_i = \min_i \frac{w_i^2}{[\mathbf{P}_w]_{ii}} \tag{4.96}$$

Oczywiście wartość L_1 można wyznaczać dla rozszerzonego filtra Kalmana, jak i dla jego szybkiej wersji, opisanej równaniami (4.73–4.78).

Pozostaje problem podjęcia decyzji, czy w ogóle należy usunąć jakiś neuron w danej k-tej iteracji algorytmu? W tym celu posłużmy się ogólną konkluzją, iż najczęściej nieistotne neurony występują w modelach niestabilnych, a z kolei wartościowe neurony mamy w dobrze nauczonych, stabilnych sieciach. Ta konkluzja wiedzie do ogólniejszego kryterium, które można zapisać jako:

$$\frac{L_1}{R_y} < \chi^2_{1,\vartheta} \tag{4.97}$$

gdzie $\chi^2_{n,\vartheta}$ jest rozkładem chi-kwadrat z poziomem ufności $\vartheta\%$ z jednym stopniem swobody, a R_y jest całkowitą wariancją modelu, która w przypadku użycia filtra Kalmana jest wyznaczana wzorem (4.69).

Jeśli powyższe kryterium jest spełnione oznacza to, że należy dokonać usunięcia neuronu, dla którego uzyskano najmniejszą wartość spośród współczynników istotności.

Tak zdefiniowane statystyczne kryterium usuwania neuronów warstwy ukrytej dla sieci typu RBF może być stosowane w praktyce z iteracji na iterację, szczególnie, gdy prowadzimy estymacje parametrów sieci za pomocą rozszerzonego filtra EKF. Z kolei, używając jednocześnie podczas uczenia sieci, kontroli wystarczalności i kontroli przydatności poszczególnych neuronów, model na bieżąco stara się estymować złożoność struktury sieci do złożoności napływających sekwencyjnie danych uczących. Takie postępowanie bardzo dobrze wpływa na końcowy poziom generalizacji.

Współpracujące statystyczne kryteria wystarczalności sieci i przydatności neuronów są używanie w sieci IncNet, a badania, wykonane dla różnych danych, wykazują dużą skuteczność ich działania [101, 100, 96, 99, 98, 97]. Rezultaty te zostały też umieszczone w rozdziale 5. Szczególnie widoczną kooperację kryteriów kontroli złożoności modelu widać w analizie danych psychometrycznych, podczas procesu uczenia, co można zobaczyć w podrozdziale 5.3.1.

4.3.6. Łączenie neuronów

Podczas rozwoju modeli sztucznych sieci neuronowych powstały różne metody usuwania neuronów. Metody usuwania neuronów opierają się o sposoby wyznaczania neuronów, które przestały być przydatne lub też ich wpływ na działanie sieci jest znikomy lub wręcz zły. Jednakże nierzadko mamy do czynienia z sytuacją, w której regiony decyzyjne, powstałe przez kooperację wielu neuronów, mogłyby zostać

zastąpione przez mniejszą sieć, nie zmniejszając właściwości klasyfikacji, czy aproksymacji, a nawet dając możliwość polepszenia jakości generalizacji, dzięki zbliżeniu złożoności modelu adaptacyjnego do złożoności danych uczących (problemu). Dlatego też możliwość łączenia dwóch (lub większej liczby, choć to prowadzi do bardziej złożonego problemu) neuronów może wpłynąć pozytywnie, nie zmieniając przy tym istotnie powierzchni aproksymowanej funkcji, czy powierzchni regionów decyzji dla klasyfikacji.

Powyżej opisanego połączenia dla pewnych dwóch neuronów i, j oraz zastąpienia ich przez nowy neuron new, można dokonać przy założeniu, iż funkcje transferu neuronów są zlokalizowane, ciągłe i jest spełnione poniższe kryterium:

$$\frac{\int_{\mathbf{d}\subseteq\mathcal{D}^n} \left|\bar{\phi}_i(\mathbf{x}) + \bar{\phi}_j(\mathbf{x}) - \bar{\phi}_{new}(\mathbf{x})\right| \, d\mathbf{x}}{\int_{\mathbf{d}\subseteq\mathcal{D}^n} \left|\bar{\phi}_i(\mathbf{x}) + \bar{\phi}_j(\mathbf{x})\right| \, d\mathbf{x}} < \alpha \tag{4.98}$$

d jest podprzestrzenią aktywności funkcji transferu $\phi_i(\mathbf{x})$ i $\phi_j(\mathbf{x})$, a $\bar{\phi}_i(\mathbf{x}) = w_i\phi_i(\mathbf{x})$ i $\bar{\phi}_j(\mathbf{x}) = w_j\phi_j(\mathbf{x})$ są przeskalowanymi funkcjami przez wagi wyjściowe. Przeskalowana funkcja transferu nowego neuronu $\bar{\phi}_{new}(\mathbf{x})$ ($\bar{\phi}_{new}(\mathbf{x}) = w_{new}\phi_{new}(\mathbf{x})$) dla $\mathbf{x}$ spoza podprzestrzeni d musi być w przybliżeniu równa zeru. Współczynnik α określa błąd, jaki uznaje się za dopuszczalny. W przypadku użycia filtru Kalmana (i nie tylko) można współczynnik α uzależnić liniowo od szumu pomiarów R_n lub od całkowitej wariancji modelu R_y (patrz równanie 4.69).

Kryterium opisane równaniem 4.98 jest trudno zastosować w ogólnym przypadku, lecz gdy funkcje transferu warstwy ukrytej są separowalne wymiarowo (na przykład dla funkcji gaussowskich lub bicentralnych), można dokonać uproszczenia tego kryterium do postaci:

$$\frac{\int_{d_1\subseteq\mathcal{D}_1}\cdots\int_{d_N\subseteq\mathcal{D}_N} \left[\bar{\phi}_i(\mathbf{x}) + \bar{\phi}_j(\mathbf{x}) - \bar{\phi}_{new}(\mathbf{x})\right]^2 \, dx_1 \ldots dx_N}{\int_{d_1\subseteq\mathcal{D}_1}\cdots\int_{d_N\subseteq\mathcal{D}_N} \left[\bar{\phi}_i(\mathbf{x}) + \bar{\phi}_j(\mathbf{x})\right]^2 \, dx_1 \ldots dx_N} < \alpha \tag{4.99}$$

Powyższe kryterium może zostać użyte w formie analitycznej lub numerycznej.

W innych przypadkach kryterium łączenia neuronów może zostać uproszczone do wyznaczania ważonego błędu średniokwadratowego w oparciu o chmurę punktów o gaussowskim, rozkładzie rozmieszczonych na obszarze aktywności neuronów i, j:

$$\frac{\sum_{d\in\mathbf{d}} \left[\bar{\phi}_i(\mathbf{x}) + \bar{\phi}_j(\mathbf{x}) - \bar{\phi}_{new}(\mathbf{x})\right]^2}{\sum_{d\in\mathbf{d}} \left[\bar{\phi}_i(\mathbf{x}) + \bar{\phi}_j(\mathbf{x})\right]^2} < \alpha \tag{4.100}$$

W przypadku użycia funkcji bicentralnych jako funkcji transferu, poszczególne para-

metry funkcji bicentralnej mogą być wyznaczone jak poniżej:

$$\boldsymbol{w}_{new} = \frac{\bar{\phi}(\mathbf{t}_i, \mathbf{t}_i) \cdot \bar{\mathbf{P}}_i + \bar{\phi}(\mathbf{t}_j, \mathbf{t}_j) \cdot \bar{\mathbf{P}}_j}{\bar{\phi}(\mathbf{t}_{new}, \mathbf{t}_{new})} \tag{4.101}$$

$$\mathbf{t}_{new,k} = \frac{1}{\mathbf{M}} \int_{d \in \mathbf{D}} x_k \, [\bar{\phi}(\boldsymbol{x}, \mathbf{t}_i) + \bar{\phi}(\boldsymbol{x}, \mathbf{t}_j)] \, d\boldsymbol{x} \tag{4.102}$$

lub (4.103)

$$\mathbf{t}_{new} = \mathbf{t}_i \cdot \bar{\mathbf{P}}_i + \mathbf{t}_j \cdot \bar{\mathbf{P}}_j \tag{4.104}$$

$$\mathbf{s}_{new} = \mathbf{s}_i \cdot \bar{\mathbf{P}}_i + \mathbf{s}_j \cdot \bar{\mathbf{P}}_j \tag{4.105}$$

$$\mathbf{b}_{new} = \begin{cases} \mathbf{b}_i & \text{neuron } j \text{ wewnątrz neuronu } i \\ \mathbf{b}_j & \text{neuron } i \text{ wewnątrz } j \\ \frac{\mathbf{b}_i + \mathbf{b}_j + |\mathbf{t}_i - \mathbf{t}_j|}{2} & \text{w p. p.} \end{cases} \tag{4.106}$$

gdzie M jest zdefiniowane przez

$$\mathbf{M} = \int_{d \in \mathbf{D}} [\bar{\phi}(\boldsymbol{x}, \mathbf{t}_i) + \bar{\phi}(\boldsymbol{x}, \mathbf{t}_j)] \, d\boldsymbol{x} = \mathbf{P}_i + \mathbf{P}_j \tag{4.107}$$

natomiast $\bar{\mathbf{P}}_i$ i $\bar{\mathbf{P}}_j$ poprzez:

$$\bar{\mathbf{P}}_i = \frac{\mathbf{P}_i}{(\mathbf{P}_i + \mathbf{P}_j)} \tag{4.108}$$

$$\bar{\mathbf{P}}_j = \frac{\mathbf{P}_j}{(\mathbf{P}_i + \mathbf{P}_j)} \tag{4.109}$$

gdzie $\mathbf{P}_i$ i $\mathbf{P}_j$ są zdefiniowane jako

$$\mathbf{P}_i = \int_{d \in \mathbf{D}} \bar{\phi}(\boldsymbol{x}, \mathbf{t}_i) \, d\boldsymbol{x} \tag{4.110}$$

$$\mathbf{P}_j = \int_{d \in \mathbf{D}} \bar{\phi}(\boldsymbol{x}, \mathbf{t}_j) \, d\boldsymbol{x} \tag{4.111}$$

Pozostaje pytanie, kiedy próbować, czy kryterium będzie spełnione, i dla jakich par neuronów sprawdzać, czy kryterium jest spełnione. Jednym ze sposobów jest sprawdzanie kryterium co epokę dla każdego neuronu i ($i = 1, \ldots, M$) i neuronu j, wybranego w następujący sposób:

$$j = \arg\max_k \; \phi(\mathbf{t}_k, \mathbf{t}_i) \tag{4.112}$$

Innym i kosztowniejszym sposobem jest próba łączenia jednej pary neuronów podczas każdej (p-tej) iteracji algorytmu adaptacji. W tym przypadku wybiera się pierwszy neuron i:

$$i = \arg\max_k \; \phi(\boldsymbol{x}_p, \mathbf{t}_k) \tag{4.113}$$

gdzie $\boldsymbol{x}_p$ jest wektorem wejściowym prezentowanym w p-tej iteracji algorytmu. Następnie wyznacza się drugi neuron j:

$$j = \arg\max_{k \neq i} \; \phi(\boldsymbol{x}_p, \mathbf{t}_k) \tag{4.114}$$

po czym bada się, czy jest spełnione kryterium łączenia neuronów dla neuronu i, j.

Gdy, dla pewnej pary neuronów kryterium łączenia będzie spełnione (niezależnie od tego czy sprawdzanie następuje co iterację, czy co epokę), następuje zastąpienie owej pary neuronów nowym neuronem o parametrach opisanych wzorami (4.101–4.106).

Powyżej zaproponowany sposób kontroli złożoności umożliwia zmniejszenie struktury sieci neuronowej poprzez unifikację jej fragmentów, w parciu o analizę neuronów, które wcale nie muszą być nieprzydatne. Wręcz przeciwnie, nierzadko da się zastąpić dwa neurony, które odgrywają istotną rolę. Tak zdefiniowaną metodę łączenia neuronów można stosować praktycznie do każdej sieci typu RBF, jak do innych modeli opartych o ciągłe i zlokalizowane funkcje transferu.

4.3.7. Wykorzystanie sieci IncNet w klasyfikacji

Sieć IncNet, w której adaptacji podlegają nie tylko wagi wyjściowe, może mieć tylko jedno wyjście. Gdyby pozostać jedynie przy adaptacji wag pomiędzy warstwą ukrytą i wyjściową, wtedy łatwo można sformułować sposób uczenia sieci z wieloma wyjściami (zrobił to Kadirkamanathan w pracy [106]). Jednakże rezygnacja z adaptacji położeń funkcji bazowych, rozmyć czy skosów i ewentualnie innych parametrów dla funkcji bicentralnych, znacząco zubaża całość modelu i zmniejsza jego potencjalne możliwości. Jeszcze gorszym rozwiązaniem (wręcz niedopuszczalnym) jest, aby sieć wykorzystywała jedno wyjście, a wartość tego wyjścia dla pewnej danej, po zaokrągleniu, byłaby interpretowana jako numer klasy, do której owa dana zostałaby przypisana. Liniowy układ wyjścia (klasa $1, 2, \ldots, K$) nie odzwierciedla w żaden sposób rzeczywistych zależności pomiędzy klasami, a raczej wprowadza wręcz arbitralne, nie istniejące i niestety niedopuszczalne zależności.

Z powyższych powodów najciekawszym rozwiązaniem wydaje się pomysł zbudowania klastra niezależnych sieci IncNet, a zadaniem każdej z podsieci byłoby wyspecjalizowanie się w rozpoznawaniu jednej (k-tej) klasy.

W tym celu ze zbioru par uczących:

$$\mathcal{S} = \{\langle \mathbf{x}_1, y_1 \rangle, \langle \mathbf{x}_2, y_2 \rangle, \ldots \langle \mathbf{x}_N, y_N \rangle\} \tag{4.115}$$

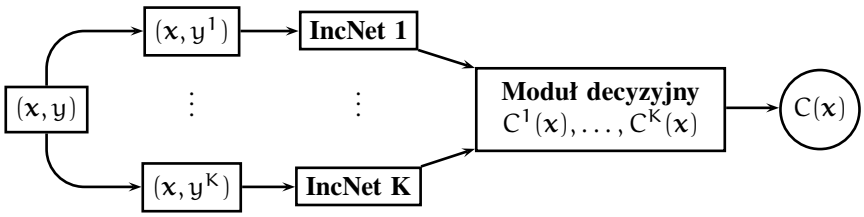
produkujemy K zbiorów, po jednym dla każdej klasy:

$$\mathcal{S}^k = \{\langle \mathbf{x}_1, y_1^k \rangle, \langle \mathbf{x}_2, y_2^k \rangle, \ldots \langle \mathbf{x}_N, y_N^k \rangle\} \quad k = 1, 2, \ldots, K \tag{4.116}$$

gdzie y_i^k przyjmuje wartości 1 lub 0:

$$y_i^k = \begin{cases} 1 & y_i = k \\ 0 & y_i \neq k \end{cases} \quad i = 1, 2, \ldots, N \tag{4.117}$$

Tak zdefiniowane zbiory $\mathcal{S}^k$ są zbiorami uczącymi dla poszczególnych sieci IncNet, które razem tworzą klaster. Schemat tak zdefiniowanego klastra sieci IncNet przedstawiono na rysunku 4.5.



Rysunek 4.5: Klaster sieci IncNet z zastosowaniu do problemów klasyfikacyjnych.

Dla danej obserwacji $\mathbf{x}$ każda z sieci produkuje wartość wyjściową $C^i(\mathbf{x})$ (dla $i = 1, 2, \ldots, K$). Wartość $C^i(\mathbf{x})$ zbliżona do zera oznacza, iż obserwacja $\mathbf{x}$ nie odpowiada klasie i. Natomiast wartość $C^i(\mathbf{x})$ zbliżona do jedynki oznacza, iż obserwacja $\mathbf{x}$ odpowiada klasie i.

Moduł decyzyjny, widoczny na rysunku 4.5, podejmuje ostateczną decyzję i przypisuje pewien wektor $\mathbf{x}$ do jednej klasy $C(\mathbf{x})$. Podejmowanie takiej decyzji może przebiegać zgodnie z poniższą definicją:

$$C(\mathbf{x}) = \arg\max_i \; C^i(\mathbf{x}) \tag{4.118}$$

Tym samym moduł decyzyjny wybiera sieć, której aktywacja była największa, tj. najbardziej odpowiadająca danej obserwacji.

Jednakże przydatność takiego klastra podsieci IncNet wcale nie musi sprowadzać się do obserwacji jedynie wartości $C(\mathbf{x})$. Bardzo przydatne jest wprowadzenie następującej renormalizacji:

$$p(C^i|\mathbf{x}) = \frac{\sigma(C^i(\mathbf{x}) - \frac{1}{2})}{\sum_{j=1}^{K} \sigma(C^j(\mathbf{x}) - \frac{1}{2})} \tag{4.119}$$

gdzie $\sigma(x) = 1/(1 + \exp(-\gamma x))$, a γ jest stałą (np. około 10). Prowadzi to do aproksymacji prawdopodobieństwa przynależności wektora $\mathbf{x}$ do klasy i.

Można zdefiniować p_{max} jako:

$$p_{max}(\mathbf{x}) = \max_i \; p(C^i|\mathbf{x}) \tag{4.120}$$

jako prawdopodobieństwo najbardziej prawdopodobnej klasy.

Natomiast $C(\mathbf{x})$ zdefiniowane poprzez

$$C(\mathbf{x}) = \arg\max_i \; p(C^i|\mathbf{x}) \tag{4.121}$$

wyznacza po prostu najbardziej prawdopodobną klasę dla danej obserwacji $\mathbf{x}$.

Warto jednak pamiętać o fakcie, iż mamy do dyspozycji wszystkie wartości $p(C^i|\mathbf{x})$ dla $i = 1, 2, \ldots, K$, a nie tylko wartość prawdopodobieństwa najbardziej prawdopodobnej klasy. Wyodrębnienie klas, których prawdopodobieństwa są największe (np.

powyżej pewnej minimalnej wartości lub grupa największych, które razem pokrywają niemal całość, np. 95%), można zobaczyć jakie klasy mogą być klasami alternatywnymi dla najbardziej prawdopodobnej klasy $C(\mathbf{x})$. Jeśli wartości tak wyselekcjonowanych prawdopodobieństw są zbliżone (lub ich część) oznacza to, że klasyfikacja nie jest jednoznaczna. Obserwując wszystkie wartości prawdopodobieństw $p(C^i|\mathbf{x})$, łatwo stwierdzić, które spośród klas można uznać za nieprawdopodobne i wykluczyć. Porównaj analizę rezultatów przedstawioną w podrozdziale 5.3.1.

4.3.8. Charakterystyka parametrów kontroli procesu adaptacji sieci IncNet

Powyżej opisane główne elementy modelu IncNet, na które składają się algorytm uczenia, funkcje transferu i metody kontroli złożoności sieci, są już wyposażone w pewne parametry umożliwiające wpływanie na proces adaptacji. Są to takie elementy jak współczynnik określający poziom szumu, funkcja $Q(n)$, która reguluje szybkość zbieżności filtra Kalmana, początkowe parametry funkcji transferu, współczynniki określające stopnie ufności, przy kryteriach wystarczalności struktury, bądź przy zmniejszaniu struktury.

Większość lub niemal wszystkie z tych parametrów są stałe lub zdeterminowane danymi uczącymi lub charakterystykami procesu uczenia. Na przykład poziom wariancji szumu pomiarów R_n powinien być znany ze względu na sposób dokonanych pomiarów lub, jeśli nie jest znany można (czy też trzeba) dokonać jego oszacowania (filtr Kalmana jest wrażliwy na poziom wariancji, ale nie bardzo; dzięki temu (ewentualne) oszacowanie wariancji szumu wcale nie ma krytycznego wpływu na proces estymacji).

Początkowe parametry funkcji transferu określają niejako *rozdzielczość* modelu, ale i początkową gładkość estymacji (poprzez dobór skosu funkcji transferu — jest to szczególnie widoczne dla funkcji bicentralnych). Rozdzielczość jest pochodną wielkości obszaru aktywnego dla zlokalizowanej funkcji transferu. Gdy obszar ten jest bardzo mały, mamy do czynienia z dużą rozdzielczością, lecz wtedy potrzeba wielu neuronów w warstwie ukrytej, by docelowy obszar przestrzeni wielowymiarowej został odpowiednio pokryty, lub też potrzeba by więcej czasu, aby funkcje mogły zostać odpowiednio *rozciągnięte*. I odwrotnie gdy obszar pokrywany przez funkcję jest bardzo duży, zbyt szybko może dojść do znacznego nakrywania się różnych funkcji transferu. Wtedy neurony zamiast kooperować zaczynają wręcz *walczyć* ze sobą. Dlatego też dobór parametrów, określających początkowe wartości parametrów funkcji transferu dobrze powiązać z analizą danych i wszelką dostępną wiedzą *a priori*, jak i próbą użycia różnych wartości dla sprawdzenia jakości działania procesu adaptacji.

Parametry kryteriów kontroli złożoności struktury mogą być niemal niezmienne, ze względu na różne aplikacje, albo wyznaczanie wartości tychże parametrów może być powiązane (liniowo) z poziomem wariancji danych.

Powyżej opisane fakty pokazują, iż przy zrozumieniu znaczenia odpowiednich współczynników, ich dobór nie jest trudny, a z pewnością korzystnie może wpłynąć na

proces adaptacji — im lepsza wiedza *a priori* wszczepiona w początkowy model, tym większe prawdopodobieństwo uzyskania lepszego poziomu generalizacji.

Sieć IncNet rozbudowano o jeszcze kilka mechanizmów kontroli. Do ważniejszych z pewnością należy stopniowe *chłodzenie* dynamiki zmiany architektury. Polega to na wyostrzaniu kryteriów, przy jakich sieć może zmieniać swoją architekturę wraz z upływem procesu uczenia, zmuszając sieć do większej stabilizacji. Taki mechanizm może być przydatny, gdy dane, na których system jest poddawany uczeniu, są mocno zdyskretyzowane (np. realne wartości są ciągłe, natomiast w danych znajdują się wartości poddane dyskretyzacji, w wyniku której uzyskuje się jedynie kilka różnych wartości) lub mocno zaszumione.

Innym mechanizmem, przydatnym na przykład w wyżej opisanych sytuacjach, jest zapamiętywanie najefektywniejszych wersji sieci (architektura i wartości wszelkich parametrów adaptacyjnych) podczas prowadzenia procesu adaptacji. W tym celu, co pewną liczbę iteracji, dokonuje się testu na podstawie którego ocenia się bieżący model i porównuje z aktualnie najlepszym modelem, po czym lepszy zostaje zapamiętany. Zapamiętywaniu podlegają wszelkie parametry modelu wraz ze strukturą sieci. Oczywiście przy ocenie wykorzystywane są jedynie dane treningowe.

Przy korzystaniu z dość dużych zbiorów danych uczących okazuje się, iż warto, w miarę prowadzenia procesu uczenia, zmniejszać początkowy obszar (dyspersje) zlokalizowanych funkcji transferu. Dzięki takiemu zabiegowi w pierwszej fazie adaptacji system dysponuje funkcjami o mniejszej rozdzielczości, natomiast w dalszej części ma do dyspozycji funkcje o większej rozdzielczości, które w już istniejącej strukturze mogą pokryć mniejsze obszary w bardziej precyzyjny sposób.

Poza opisanymi metodami kontroli zaimplementowano jeszcze parę mniej istotnych. Na koniec wspomnę o możliwości określenia maksymalnej liczby neuronów w warstwie ukrytej. Ten mechanizm, jak i inne nie wspomniane w pracy, są przydatne raczej na poziomie wstępnej oceny samych danych niż we właściwym procesie uczenia.

4.3.9. Przedziały ufności, jako narzędzie analizy danych i wizualizacji wyników

Oprócz analizy istotnego podzbioru prawdopodobieństw $\{p(C^i|\mathbf{x}) : i = 1, \ldots, K\}$ dla najbardziej prawdopodobnej klasy można wyznaczyć w poszczególnych wymiarach wejściowych przedział, w którym zmienność wartości owego wymiaru nie zmieni klasyfikacji. Ściślej, zakładając, że wektor $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ został sklasyfikowany jako obiekt klasy k, przedział $[x^r_{min}, x^r_{max}]$ dla cechy r wyznaczony jest przez:

$$x^r_{min} = \min_{\bar{x}} \{C(\bar{\mathbf{x}}) = k \ \wedge \ \forall_{x_r > \hat{x} > \bar{x}} C(\hat{\mathbf{x}}) = k\} \tag{4.122}$$

$$x^r_{max} = \max_{\bar{x}} \{C(\bar{\mathbf{x}}) = k \ \wedge \ \forall_{x_r < \hat{x} < \bar{x}} C(\hat{\mathbf{x}}) = k\} \tag{4.123}$$

gdzie $\bar{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \bar{x}, x_{r+1}, \ldots, x_N]$, a $\hat{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \hat{x}, x_{r+1}, \ldots, x_N]$.

Tak wyznaczone przedziały $[x^r_{min}, x^r_{max}]$ dla $r = 1, \ldots, N$, opisują możliwe od-

chylenia wartości dla poszczególnych współrzędnych r klasyfikowanego wektora $\mathbf{x}$, podczas gdy wartości pozostałych cech pozostają niezmienne (równe odpowiednim współrzędnym wektora $\mathbf{x}$). Dalej przedziały te będą nazywane *przedziałami ufności*.

Umiejscowienie wartości współrzędnych wektora $\mathbf{x}$ w odpowiadających im przedziałach ufności, pomaga stwierdzić, czy wektor $\mathbf{x}$ jest na obrzeżu regionu decyzyjnego, czy raczej w jego centrum.

Wyznaczanie przedziałów ufności można rozbudować o próg ufności, tak aby prawdopodobieństwo zwycięskiej klasy było istotnie większe od najbardziej prawdopodobnej klasy alternatywnej. Można to zrealizować, dodając nierówność do wzorów (4.122) i (4.123):

$$x_{min}^{r,\beta} = \min_{\bar{x}} \left\{ C(\bar{\mathbf{x}}) = k \ \wedge \ \forall_{x_r > \hat{x} > \bar{x}} C(\hat{\mathbf{x}}) = k \ \wedge \ \frac{p(C^k|\bar{\mathbf{x}})}{\max_{i \neq k} p(C^i|\bar{\mathbf{x}})} > \beta \right\} \tag{4.124}$$

$$x_{max}^{r,\beta} = \max_{\bar{x}} \left\{ C(\bar{\mathbf{x}}) = k \ \wedge \ \forall_{x_r < \hat{x} < \bar{x}} C(\hat{\mathbf{x}}) = k \ \wedge \ \frac{p(C^k|\bar{\mathbf{x}})}{\max_{i \neq k} p(C^i|\bar{\mathbf{x}})} > \beta \right\} \tag{4.125}$$

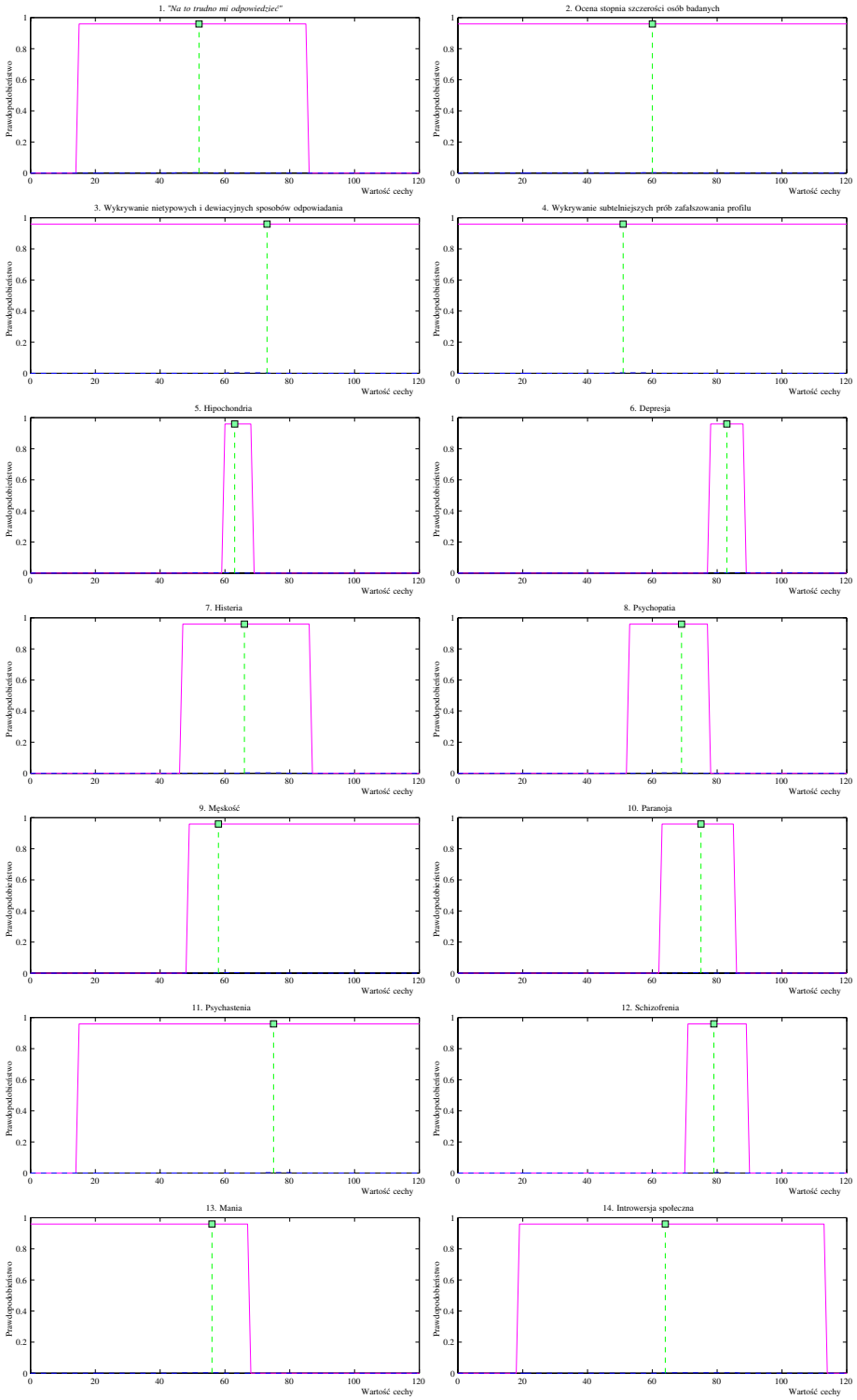
gdzie $\bar{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \bar{x}, x_{r+1}, \ldots, x_N]$, a $\hat{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \hat{x}, x_{r+1}, \ldots, x_N]$. Współczynnik β oznacza wybrany próg. Oczywiście można prowadzić obserwacje zmienności przedziałów, w zależności, od doboru wartości progu β.

Tak zdefiniowane przedziały ufności stanowią silną alternatywę dla reguł logicznych i są narzędziem, które może znacząco wspomóc proces diagnozy, co będzie można prześledzić na poniższych przykładach.

Rysunek 4.6 ilustruje przykład wyznaczonych przedziałów ufności dla jednego przypadku z psychometrycznej bazy danych, szczegółowo opisywanych w podrozdziale 5.3.1. Przedstawione przedziały ufności opisują jednoznaczny przypadek psychozy reaktywnej, dla którego prawdopodobieństwo przynależności wyniosło 0.96. Kolejne podrysunki ilustrują dopuszczalne zmiany wartości poszczególnych cech.

Zielony kwadrat odpowiada wartości cechy analizowanego przypadku, a wartość odciętej to prawdopodobieństwo najbardziej prawdopodobnej z klas (por. wzór 4.120). Kolorem karmazynowym oznaczono dopuszczalny zakres wartości danej cechy dla najbardziej prawdopodobnej klasy. Wartość na osi pionowej odpowiada wartości wyznaczonego prawdopodobieństwa. Kolorem niebieskim (linią przerywaną) oznaczono dopuszczalny zakres wartości danej cechy dla drugiej z najbardziej prawdopodobnych klas. Podobnie jak dla najbardziej prawdopodobnej klasy, wartość rzędnej odpowiada wartości wyznaczonego prawdopodobieństwa dla tej klasy. Ten przypadek jest jednak tak jednoznaczny, że zakres drugiej alternatywnej klasy po prostu niemal leży na osi rzędnych. Dzięki temu możemy nie tylko obserwować dopuszczalne zakresy zmian, ale widzimy również stosunek prawdopodobieństw najbardziej istotnych klas.

Dla odróżnienia rysunek 4.7 ilustruje przypadek, który nie jest tak jednoznaczny,



Rysunek 4.6: Przedziały ufności. Przypadek psychozy reaktywnej.

jak poprzedni. W wyniku klasyfikacji okazało się, iż najbardziej prawdopodobną klasę stanowią zmiany organiczne 0.56, drugą najbardziej prawdopodobną klasą jest schizofrenia 0.39. Pozostałe klasy nie ma już istotnego wpływu (dla rozpatrywanego przypadku). Na podstawie tego przypadku zauważyć można znacznie bardziej istotny wpływ klasy alternatywnej, schizofrenii.

Jednak tak jak i reguły logiczne, przedziały ufności pokazują stałe prawdopodobieństwo podczas gdy wartości odpowiednich współrzędnych ulegają zmianie. Dlatego też znacznie bogatsze w informacje jest zilustrowanie nie tylko przedziałów ufności, ale również wartości prawdopodobieństw towarzyszących zmieniającym się wartościom poszczególnych cech. Takie przedziały będą nazywane *probabilistycznymi przedziałami ufności*. Dzięki takiej zmianie w miejsce *prostokątów*, które symbolizowały przedziały, pojawią się krzywe, które będą pokazywały zmianę prawdopodobieństwa.

Dla powyżej omówionych dwóch przypadków: psychozy reaktywnej i zmian organicznych, wyznaczone zostały probabilistyczne przedziały ufności — patrz rysunek 4.8 i 4.9.

Zielony kwadrat odpowiada wartości cechy analizowanego przypadku, a wartość rzędnej to prawdopodobieństwo najbardziej prawdopodobnej z klas (por. wzór 4.120). Kolorem karmazynowym oznaczono krzywą zmian wartości prawdopodobieństwa zwycięskiej klasy dla zmieniających się wartości odpowiedniej cechy analizowanego przypadku. Krzywa dla cechy r jest opisana przez prawdopodobieństwo $p(C(\mathbf{x})|\bar{\mathbf{x}})$ ($C(\mathbf{x})$ jest zdefiniowane równaniem 4.121), gdzie $\bar{\mathbf{x}} = [x_1, \ldots, x_{r-1}, \bar{x}, x_{r+1}, \ldots, x_N]$ Krzywa niebieska kropkowana przedstawia prawdopodobieństwo przynależności do drugiej z najbardziej prawdopodobnych klas rozpatrywanego przypadku dla zmieniających się wartości odpowiedniej cechy analizowanego przypadku. Krzywą dla cechy r opisuje prawdopodobieństwo przez $p(C^{k_2}|\bar{\mathbf{x}})$, gdzie k_2 jest zdefiniowane jako:

$$k_2 = \arg\max_i \{p(C^i|\mathbf{x}),\ C^i \neq C(\mathbf{x})\} \tag{4.126}$$

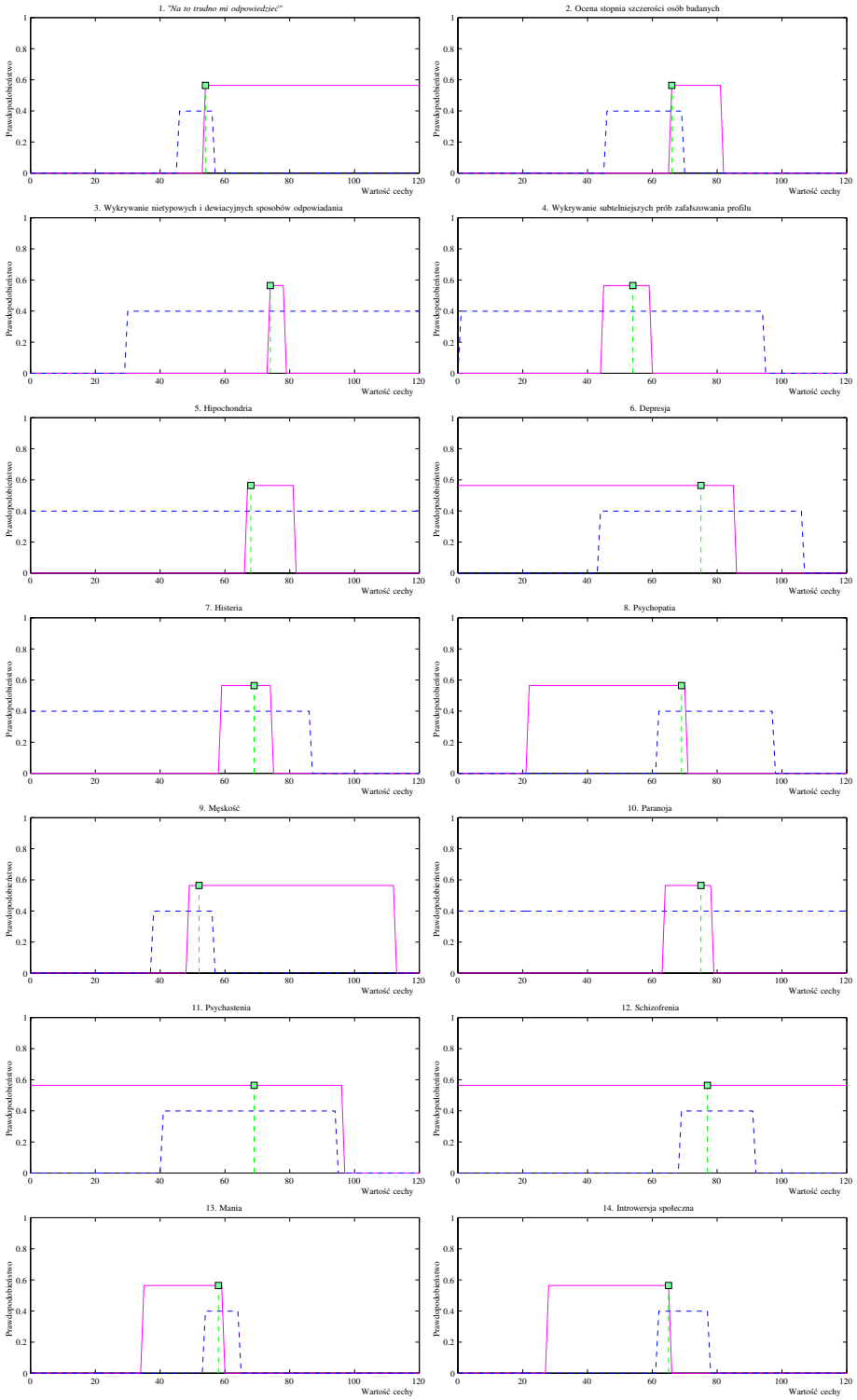
Natomiast przerywana krzywa turkusowa przedstawia prawdopodobieństwo klasy alternatywnej, najbardziej prawdopodobnej dla danej wartości prezentowanej cechy i pozostałych wartości cech zgodnych z rozpatrywanym przypadkiem (dla różnych wartości prezentowanej klasy alternatywne mogą być różne). Tą krzywą opisuje prawdopodobieństwo $p(C^{k_M}|\bar{\mathbf{x}})$, gdzie k_M jest zdefiniowane jako:

$$k_M = \arg\max_i \{p(C^i|\bar{\mathbf{x}}),\ C^i \neq C(\mathbf{x})\} \tag{4.127}$$

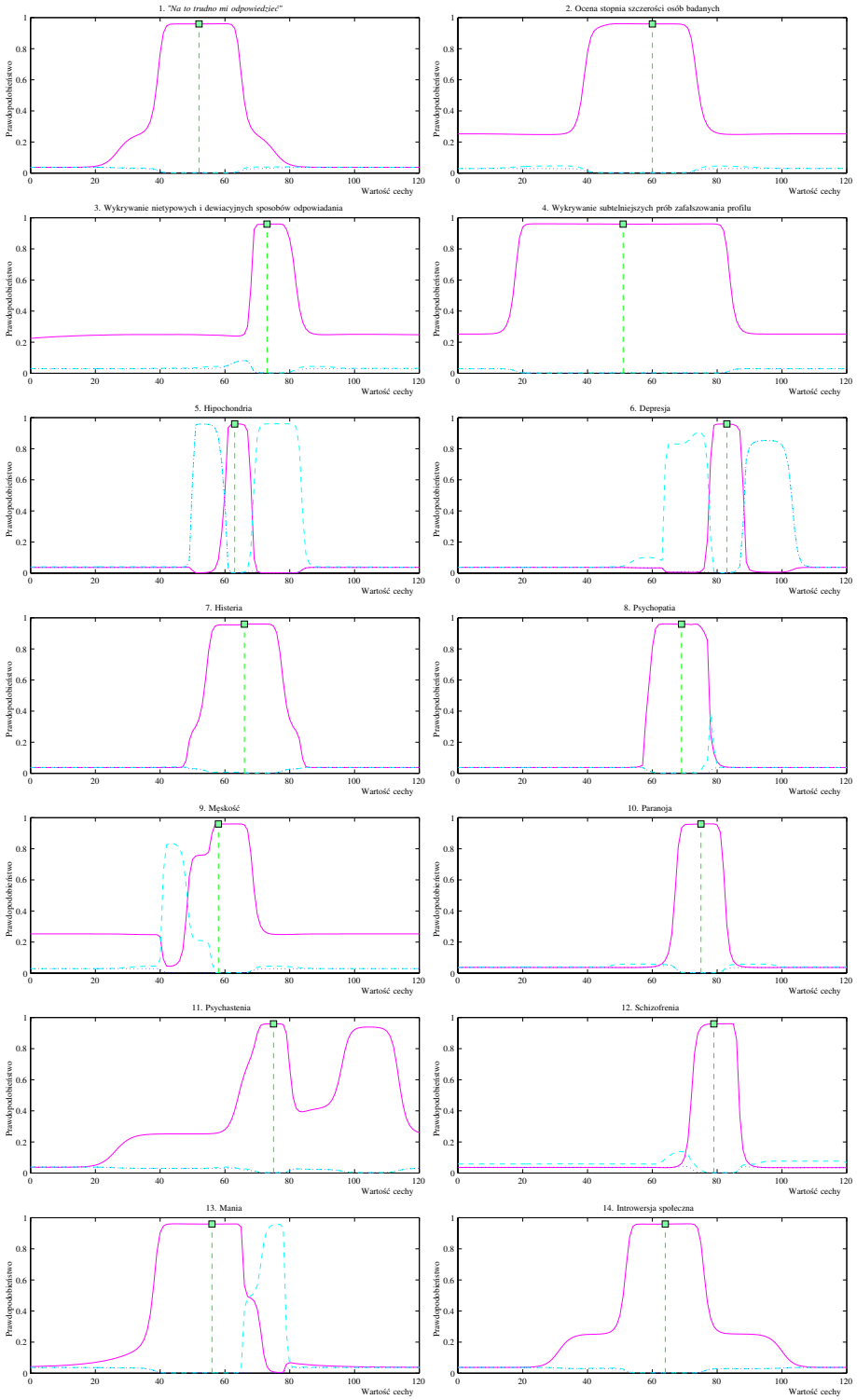
Proszę zwrócić uwagę, iż w równaniu 4.127 wyznacza się maksimum z $p(C^i|\bar{\mathbf{x}})$ w punkcie $\bar{\mathbf{x}}$, natomiast w równaniu 4.126 w punkcie $\mathbf{x}$.

Więcej przykładów probabilistycznych przedziałów ufności można znaleźć w podrozdziale 5.3.1.

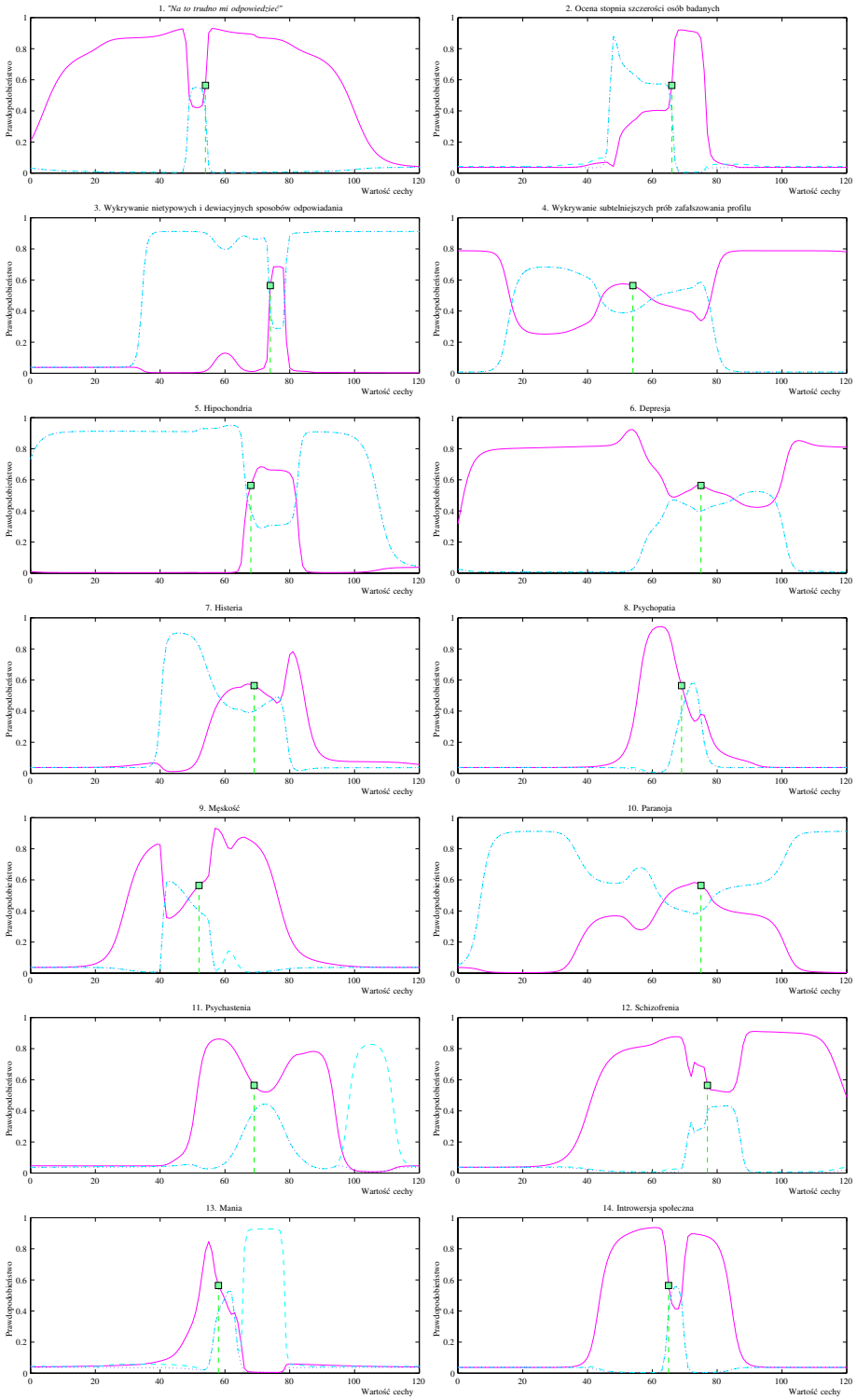
Probabilistyczne przedziały ufności znacznie wzbogacają informację, wspomagając proces klasyfikacji i diagnozy. Analizując zmienność prawdopodobieństwa dla poszczególnych cech łatwo można znaleźć wiele istotnych własności, charakterystycznych dla rozpatrywanego przypadku. Przede wszystkim poprzez analizę wpływu klas



Rysunek 4.7: Przedziały ufności. Przypadek zmian organicznych i schizofrenii.



Rysunek 4.8: Probabilistyczne przedziały ufności. Przypadek psychozy reaktywnej.



Rysunek 4.9: Probabilistyczne przedziały ufności. Przypadek zmian organicznych i schizofrenii.

alternatywnych (krzywa turkusowa) można zbadać, na ile proces klasyfikacji jest jednoznaczny i stabilny ze względu na małe zmiany wartości cech. To wydaje się najważniejszą kwestią nie tylko w medycynie. Patrząc na umiejscowienie rozpatrywanego przypadku w wyznaczonych przedziałach ufności można stwierdzić na ile jest on typowy. Z kolei obserwując charakter zmian zwycięskiej klasy można określić, czy ma ona wpływ na dany przypadek, a jeśli tak, na ile jest on istotny.

Przedziały ufności i probabilistyczne przedziały ufności, a reguły logiczne. Najbardziej widoczną i znaczącą różnicą jest rodzaj interpretacji: probabilistyczny dla przedziałów ufności i logiczny dla reguł logicznych. W konsekwencji tego odpowiedź w analizie danego przypadku przez reguły to *tak* lub *nie*, nie ma wartości pośrednich. W realnych zastosowaniach, szczególnie w medycynie, taki brak wartości pośrednich znacznie obniża zaufanie do sugerowanej diagnozy. Z kolei, gdy dochodzi do analizy trudnych przypadków, nierzadko napotyka się na wektory, które znajdują się w obszarze więcej niż jednej reguły, które należą do różnych klas lub też okazuje się, że w ogóle żadna z reguł nie pokrywa obszaru, w którym znajduje się analizowany przypadek (por. [131]).

Analiza probabilistycznych przedziałów ufności dostarcza prawdopodobieństw, czyli umożliwia ciągłą analizę, gładko opisując wszelkie zmiany analizowanych prawdopodobieństw. Umożliwia to dogłębną analizę danego przypadku, niezależnie w każdym wymiarze. Prawdopodobieństwa umożliwiają dokładne porównanie różnicy pomiędzy klasą najbardziej prawdopodobną, a klasą lub klasami alternatywnymi — każdy przypadek może mieć choćby minimalnie inny rozkład prawdopodobieństw przynależności do poszczególnych klas.

Wszystko to sprawia, że probabilistyczne przedziały ufności stanowią niezwykle silne narzędzie wspomagania procesu klasyfikacji, jego analizy i wizualizacji. W efekcie czego trudno już taki model postępowania nazwać *czarną skrzynką*, co nierzadko zarzuca się metodom sztucznych sieci neuronowych.

Rozdział 5

Zastosowanie sieci IncNet do klasyfikacji i analizy danych medycznych

5.1. Techniki porównywania różnych modeli

Rozmaite modele adaptacyjne, które uczą się z danych, cechują się przeróżnymi właściwościami. Część możliwości porównywania modeli, to analityczne możliwości porównywania podobieństw i różnic matematycznych właściwości poszczególnych modeli. Jednak najczęściej nie jest to wystarczające, co jest spowodowane zbyt ogólnym poziomem takich analiz. Dlatego też najczęstszym sposobem porównywania różnych modeli jest porównywanie błędów, jakie te modele popełniają już po procesie adaptacji. Należy jednak pamiętać, że istnieją dwa źródła błędów modelu. Pierwsze to niedoskonałość samego modelu adaptacyjnego, drugie to niedoskonałość danych.

Całkowity Błąd Modelu	=	Niedoskonałość Modelu	+	Niedoskonałość Danych

Dodatkowo ważnym jest rozróżnienie dwóch typów błędów: *błędu uczenia* i *błędu generalizacji*. Pierwszy, to błąd, który model popełnia na zbiorze, na którym był uczony. Drugi, błąd generalizacji, to błąd, jaki popełnia model na zbiorze testowym, którego elementy nie brały udziału w procesie uczenia. Obserwacja błędu uczenia pokazuje nam postęp (lub jego brak) w procesie uczenia. Końcowa wartość błędu uczenia nie jest niestety wymiernym współczynnikiem umożliwiającym określenie, tego, co najważniejsze, czyli uzyskanego poziomu generalizacji. Z analizy wielu artykułów

wynika, że nadal wiele osób lekceważy ten fakt. Istotnie więcej na temat poziomu generalizacji uzyskuje się z obserwacji błędu uzyskanego na zbiorze testowym.

Skąd bierze się zbiór testowy? Mamy tu dwa główne przypadki:

- gdy mamy do czynienia z pewnym odgórnie zdefiniowanym podziałem zbioru danych na część treningową i testową,
- gdy nie istnieje żaden odgórny podział zbioru danych.

O ile pierwszy przypadek nie pozostawia żadnych wątpliwości, to drugi staje się najczęściej przyczyną powstawania przeróżnych sposobów podziału zbioru danych na część (części) treningową i testową. To z kolei nierzadko jest powodem niemożności dalszego porównywania błędów różnych modeli. Najczęściej powodem uniemożliwiającym dalsze porównywanie wyników jest dysproporcja pomiędzy liczebnością zbioru treningowego i testowego w różnych podziałach.

Istnieje jednak parę standardowych sposobów podziału zbioru danych i wyznaczania dzięki temu poszczególnych błędów.

Pierwszym i najprostszym sposobem jest losowy podział na część testową i treningową. Najczęściej dokonuje się podziału w następujących proporcjach: 90% i 10%, odpowiednio dla zbioru treningowego i testowego; 80% i 20%; 70% i 30%; a czasem i 50% na 50%. W związku z ogromną liczbą możliwych podziałów, wynikających z losowości podziału, należy w praktyce dokonać uśrednienia po wielu takich podziałach, aby uniezależnić wyniki badań od owego losowego podziału. Wtedy, jako błąd, podaje się wartość średnią (odpowiednio dla zbioru treningowego i testowego).

Drugi sposób podziału zbioru danych nazwać można testem krzyżowym, rotacyjnym lub kroswalidacją (*ang. crossvalidation*), CV. Test ten polega na losowym podziale zbioru danych $\mathcal{S}$ na k możliwie równo licznych podzbiorów $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_k$. Następnie na podstawie dokonanego podziału tworzy się k par zbiorów treningowych i testowych:

$$\mathrm{TRS}_i = \bigcup_{j \neq i} \mathcal{S}_j \tag{5.1}$$

$$\mathrm{TES}_i = \mathcal{S}_i \tag{5.2}$$

dla $i = 1, 2, \ldots, k$.

Kolejnym krokiem jest użycie powyżej zdefiniowanych par zbiorów, kolejno do uczenia i wyznaczenia błędów treningowych i testowych. Ostatecznie umożliwia to wyznaczenie końcowych rezultatów:

$$E_{\mathrm{TRS}} = \frac{1}{k} \sum_{i=1}^{k} E_{\mathrm{TRS}_i} \tag{5.3}$$

$$E_{\mathrm{TES}} = \frac{1}{k} \sum_{i=1}^{k} E_{\mathrm{TES}_i} \tag{5.4}$$

$$\tag{5.5}$$

gdzie E_{TRS_i} jest błędem modelu uzyskanym po uczeniu na zbiorze treningowym, a E_{TES_i} jest błędem modelu, uzyskanym po uczeniu na zbiorze testowym.

Do najbardziej typowych podziałów należy podział na 10 podzbiorów (10 CV) i na n podzbiorów, gdzie n oznacza liczbę wektorów danych. Ostatni przypadek nazywany jest *leave one out* (LOO).

Czasem spotyka się również odmianę testu krzyżowego, która statystycznie powinna dawać wierniejsze rezultaty od wyżej opisanej metody. Metoda ta dotyczy danych, na których dokonuje się klasyfikacji. Jedyna różnica polega na dodatkowym warunku nakazującym utrzymywanie we wszystkich podzbiorach takich samych proporcji elementów poszczególnych klas do liczby pozostałych elementów, jakie są w całym zbiorze danych. Nazywa się ją kroswalidacją stratyfikowaną (*ang. stratified crossvalidation*), SCV.

Choć czasem różnice pomiędzy wynikami z 10 CV i LOO są nieduże, to na ogół jednak różnica pomiędzy nimi jest statystycznie istotna.

Gdy wartości 10 CV i LOO są zbliżone, może to oznaczać, że zbiór danych dobrze opisuje problem i tym samym usuniecie z niego 10% danych nie powoduje istotnych błędów wynikających z niereprezentatywności próbki w stosunku do (nieznanego) rozkładu prawdopodobieństwa. Prawdą jest również, że bardziej wartościowy jest *dobry* rezultat 10 CV niż rezultat z taka samą wartością dla LOO. Z tego wynika fakt, iż kiedy model A ma taki rezultat dla 10 CV jak model B dla LOO, to z pewnością model A nie powinien być gorszy. Podobne konkluzje znaleźć można w [16].

W problemach klasyfikacji jako miary błędu używa się współczynnika poprawności

$$WP = \frac{\text{ilość poprawnie sklasyfikowanych wektorów}}{\text{ilość wektorów}} \tag{5.6}$$

lub też błędu

$$WB = 1 - WP = \frac{\text{ilość niepoprawnie sklasyfikowanych wektorów}}{\text{ilość wektorów}} \tag{5.7}$$

Sumaryczny błąd kwadratowy (*ang. Sum Squared Error, SSE*), jak i pozostałe miary wymienione poniżej, są częściej wykorzystywane w aproksymacji, niż klasyfikacji:

$$SSE = \sum_{i=1}^{n} (F(\mathbf{x}_i) - y_i)^2 \tag{5.8}$$

Jeszcze częściej korzysta się ze średniego błędu kwadratowego (*ang. Mean Squared Error, MSE*):

$$MSE = \frac{1}{n} SSE = \frac{1}{n} \sum_{i=1}^{n} (F(\mathbf{x}_i) - y_i)^2 \tag{5.9}$$

czy też pierwiastka średniego błędu kwadratowego (*ang. Root Mean Squared Error, RMSE*):

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}} = \frac{1}{\sqrt{n}}\sqrt{\mathrm{SSE}} = \frac{1}{\sqrt{n}}\sqrt{\sum_{i=1}^{n}(F(\mathbf{x}_i) - y_i)^2} \tag{5.10}$$

Inna stosowaną miarą jest średni błąd procentowy (*ang. Average Percentage Error, APE*):

$$\mathrm{APE} = \frac{1}{N}\sum_{i=1}^{n}\left|\frac{F(\mathbf{x}_i) - y_i}{y_i}\right| * 100\% \tag{5.11}$$

Czasami spotyka się również miarę, która uwzględnia nie tylko błąd, ale i wariancję (*ang. Average Relative Variance, ARV*)

$$\mathrm{ARV} = \frac{\sum_{i=1}^{n}(F(\mathbf{x}_i) - y_i)^2}{\sum_{i=1}^{n}(\bar{y} - y_i)^2} \tag{5.12}$$

gdzie $\bar{y} = 1/n\sum_{i=1}^{n} y_i$.

5.2. Wstępne przetwarzanie danych

Jak już stwierdzono w poprzednim podrozdziale, na ostateczny całkowity błąd ma wpływ nie tylko niedoskonałość modelu, ale i niereprezentatywność danych. Dlatego też niezwykle ważne jest, aby dane były jak najlepsze. Na jakość danych wpływa wiele czynników. Do głównych należą: jakość dokonanych pomiarów, jakość zbierania i przechowywania danych (nierzadko mamy do czynienia z brakującymi informacjami), zawartość informacyjna poszczególnych cech (atrybutów) i wstępne przetwarzanie danych. Uwag na tematy wstępnego przetwarzania danych można doszukać się w różnych publikacjach, a szczególnie warto wymienić [20, 166, 10, 95].

5.2.1. Transformacje danych

Nierzadko wartości niektórych cech nie mają rozkładów liniowych czy normalnych, obserwuje się czasami eksponencjalny czy logarytmiczny rozrzut danych w pewnym wymiarze. Takie cechy najczęściej mogą wnieść więcej informacji po dokonaniu transformacji odwrotnej od tej, która została zaobserwowana w danym wymiarze. Z kolei gdy pewne cechy przyjmują wartości symboliczne należy rozważyć użycie miar heterogenicznych. Więcej informacji na ten temat znajduje się w podrozdziale 2.2.1.

W problemach klasyfikacyjnych większość metod zazwyczaj zakłada podobny rozrzut danych w poszczególnych wymiarach przestrzeni wejściowej. Stąd też najczęściej dokonuje się pewnej transformacji danych, która zniweluje zbyt duże, początkowe dysproporcje pomiędzy wartościami w poszczególnych wymiarach.

Najprostszą stosowaną transformacją jest **normalizacja** danych tak, aby po transformacji wartości mieściły się w przedziale $[0,1]$ (w podobny sposób można dokonać transformacji tak aby wartości mieściły się w przedziale$[-1,1]$). W tym celu dla każdej cechy dokonuje się poniższego przekształcenia:

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{5.13}$$

przy czym

$$x_{min} = \min_i x_i \tag{5.14}$$

$$x_{max} = \max_i x_i \tag{5.15}$$

a i zmienia się od 1 do n.

Powyższa transformacja może być czasami wręcz niebezpieczna. Gdy mamy do czynienia z błędnymi wartościami w pewnych cechach, może się okazać iż leżą one daleko poza normalnym zakresem wartości danej cechy. W takim przypadku dochodzi do nadmiernego *ściśnięcia* znormalizowanych wartości danej cechy, niosących najwięcej istotnych informacji. Z tego też powodu często stosuje się powyższą normalizację danych, ale wartości x_{min} i x_{max} wybiera się nie spośród całego zbioru $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$, lecz po odrzuceniu ze zbioru $\mathcal{S}$ k% najmniejszych i największych wartości (za k przyjmuje się najczęściej 5 lub 10). Taką normalizację najczęściej nazywa się **normalizacją z obcięciem**.

Innym sposobem jest **standaryzacja** danych:

$$x_i' = \frac{x_i - \bar{x}}{\sigma_x} \tag{5.16}$$

gdzie $\bar{x}$ jest wartością średnią, natomiast σ_x jest standardowym odchyleniem:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{5.17}$$

$$\sigma_x = \frac{1}{n-1}\sum_{i=1}^{n} (x_i - \bar{x})^2 \tag{5.18}$$

Tak jak czysta normalizacja może prowadzić do złych konsekwencji, tak stosowanie normalizacji z obcięciem, czy standaryzacji, jest bezpieczniejsze i zazwyczaj nie prowadzi do istotnych różnic w późniejszym procesie adaptacji.

Czasem, gdy rozkład danych dla pewnej cechy (czy grupy cech) nie jest normalny lub liniowy i bardziej zależy nam na zachowaniu relacji pomiędzy poszczególnymi elementami, niż odległości, jakie one wyznaczają, można skorzystać z poniższego przekształcenia (niezależnie dla wszystkich cech):

$$x_i' = 2|\{x : x < x_i\}| + |\{x : x = x_i\}| \tag{5.19}$$

$|\mathcal{Z}|$ oznacza tu moc zbioru $\mathcal{Z}$.

Powyższe przekształcenie oparte o podobieństwo, a nie o realne odległości, może być zastosowane do części lub wszystkich wymiarów. Ważną własnością tego przekształcenia jest niwelacja nieliniowości rozkładu danych w wymiarze, który poddaje się transformacji.

Wadą może okazać się łączenie się (niemal) uprzednio odległych skupisk danych o ciągłych wartościach, pomiędzy którymi nie było żadnych innych danych. Jednak można temu zaradzić, wprowadzając do powyższego przekształcenia informacje o punktach, które są istotnymi punktami podziału wartości cech w wymiarze, który aktualnie podlega transformacji. Punkty takie można wyznaczyć korzystając na przykład z kryterium używanego w drzewach CART [17], kryterium dipolowego [14, 12, 13], czy SSV [78] lub metod opartych na histogramach i dendrogramach. Wyznaczanie punktów podziałów powinno przebiegać równocześnie dla wszystkich wymiarów podlegających transformacji. Prowadzi to do optymalnego wyboru tych punktów i przypisania im wartości, które odzwierciedlają ich wpływ na polepszenie podziału.

Tak zmodyfikowaną transformację można zdefiniować poprzez:

$$x_i' = 2|\{x : x < x_i\}| + |\{x : x = x_i\}| + \sum_{p \in P^k \wedge p < x_i\}} \kappa_p \tag{5.20}$$

gdzie P^k jest zbiorem punktów podziału k-tego wymiaru. Natomiast κ_p jest współczynnikiem określającym istotność podziału w punkcie p względem innych wyznaczonych punktów.

5.2.2. Wartości nietypowe

Problemów mogą dostarczyć również wartości nietypowe poprzez zaburzenia procesu uczenia i w końcowym efekcie spowodować obniżenie poziomu generalizacji. Dobrymi przykładami takich wartości są wszelkie irracjonalne wartości w danym zagadnieniu, np. temperatura ciała człowieka wynosząca 70 stopni Celsjusza. Możne też dojść do powstania wartości nietypowych w parach cech lub grupach cech. Na przykład: wiek = 5 lat, wzrost = 166 cm. W tym przypadku wartości poszczególnych cech są jak najbardziej dopuszczalne, ale ich kombinacja jest nierealna. Pochodzenie wartości nietypowych jest zazwyczaj związane z procesem dokonywania pomiarów, lub ich kolekcjonowania i przechowywania. W miarę możliwości należy dokonać poprawek takich wartości, bądź ich usunięcia, lub usunięcia całych wektorów, w których występują takie wartości.

5.2.3. Wartości brakujące

Inny problem stanowią wartości brakujące w wektorach, opisujących różne przypadki. Niewątpliwie najlepszym lekarstwem jest właściwe uzupełnienie takich braków. Jednak w praktyce jest to niemal zawsze niewykonalne. Jeśli nie można uzupełnić, to należy wziąć pod uwagę możliwość redukcji zbioru danych o wektory, które posiadają wartości brakujące, lecz gdy zbiór danych nie jest wystarczająco duży, to i takie

rozwiązanie nie możne być zastosowane. Ważna jest przyczyna powstania wartości brakujących. W przypadku danych medycznych mamy do czynienia z dwoma istotnie różnymi przyczynami. Pierwsza najczęściej jest spowodowana decyzją, której konkluzją było niewykonanie jakiejś analizy czy pomiaru. Druga grupa to braki wynikające z zagubienia informacji lub na przykład utraty kontaktu z pacjentem.

Jedną z ciekawszych możliwości jest pominięcie wymiaru, w którym mamy do czynienia z wartością brakującą. Jednak nie każdy model umożliwia opuszczenie takiego wymiaru bez znacznych skutków ubocznych. Możliwość opuszczenia pewnego wymiaru podczas klasyfikacji mamy, gdy model korzysta z separowalnych funkcji transferu (por. podrozdział 2.4.6 i 2.4.7), na przykład z funkcji Gaussa, czy funkcji bicentralnych.

Inną możliwością postąpienia z wartościami brakującymi, jest przypisanie im specjalnych wartości spoza przedziału, w którym występują wartości w danym wymiarze. Powoduje to uwzględnienie informacji o, na przykład, niedokonaniu pewnego badania, czy niewykonaniu jakiegoś pomiaru.

Jeszcze innym spotykanym podejściem jest zastąpienie wartości brakującej wartością, która jest najbardziej prawdopodobna (np. najczęściej występującą w danym wymiarze) lub najoptymalniejszą wartością, zakładając wykorzystanie pewnego klasyfikatora $\mathcal{X}$. Jednakże w takim przypadku nie można być pewnym trafności wyboru i czasem może to zwiększyć ryzyko popełnienia błędu (choć sam sposób postępowania ma na celu znalezienie najbardziej prawdopodobnej wartości w miejsce wartości nieznanej). Dla przykładu, gdy założyć, iż wartości brakujące zastąpi się najczęściej występującą wartością danej cechy, można spodziewać się, że brakująca informacja o wzroście niemowlęcia wyniesie około 165 cm.

Stosowanie wartości średnich danego wymiaru, jako substytutu wartości brakującej, również pojawia się w literaturze. Lecz podobnie jak ostatni sposób (lub jeszcze bardziej) naraża to na popełnienie błędu w procesie uczenia, bądź klasyfikacji, choć w literaturze często spotyka się takie postępowanie podczas porównywania generalizacji różnych metod.

Wszystkie powyżej zaprezentowane podejścia do problemu wartości brakujących rozwiązują problem jedynie częściowo i mogą powodować różne konsekwencje, niekoniecznie oczekiwane. Lecz gdy wrócić do źródeł powstawania brakujących wartości, to można dojść do konstruktywnych konkluzji.

Źródła wartości brakujących to:

- wartość nie była wyznaczana — np. lekarz nie zlecił wykonania danego testu
- wartość nie została wyznaczona — np. lekarz zlecił wykonanie danego testu lecz wyniku nie ma

W pierwszym przypadku zastępowanie wartością średnią bądź niby najbardziej prawdopodobną może nieść za sobą duże ryzyko, ponieważ *nie zlecenie* wyznaczenia wartości na pewno było świadomą decyzją, popartą pewną wiedzą *a priori*, o której

trudno założyć, że istnieje o niej informacja w samych danych. Dlatego też raczej należy przypisać pewną ustaloną stałą wartość spoza zakresu występowania danych w tym wymiarze, która będzie symbolizowała owo *nie zlecenie* wyznaczenia wartości.

Natomiast w drugim przypadku wartość w ewidentny sposób została zagubiona w procesie jej wyznaczania bądź przechowywania. Nie ma w tym żadnej świadomej działalności podobnej do poprzedniego przypadku. W takim przypadku bardziej wskazanym wydaje się uzupełnienie wartości możliwie najbardziej prawdopodobną wartością, przy wykorzystaniu do tego wszelkich informacji dostępnych w danych uczących lub korzystaniu z pewnego klasyfikatora[1], wspomagającego sam proces uprawdopodobniania owej wartości.

Takie postępowanie jest połączeniem dwóch, już wcześniej opisywanych metod, ale z uwzględnieniem źródła wartości brakującej. W rezultacie otrzymuje się bardziej racjonalną metodę postępowania.

5.2.4. Selekcja cech

Bardzo pozytywnie na przebieg procesu uczenia, jak i ostateczny poziom generalizacji, może wpłynąć wybór istotnych cech, spośród wszystkich wymiarów danych wejściowych. Powstało bardzo wiele i bardzo różnych metod wyboru istotnych cech. W pracach [34, 56, 56, 10, 3] można znaleźć porównania metod selekcji cech.

Metody różnią się sposobami doboru ilości i kolejności cech w analizie. Niektóre z metod prowadzą analizę całej przestrzeni możliwości, która czasem może być duża (jej złożoność to $O(2^N)$). Inne metody dobierają lub odrzucają cechy heurystycznie (wtedy złożoność zazwyczaj nie przekracza $O(N^2)$) lub analizują różne losowe kombinacje cech (wtedy maksymalna złożoność znów wnosi $O(2^N)$, lecz liczbę losowań ogranicza się z góry).

Metody doboru cech mogą też różnić się metodami oceny cech. Można prowadzić analizę w oparciu o odległość (separowalność czy dyskryminację — np. drzewa decyzyjne). Korzysta się również z różnych miar informacji przy ocenie danego wymiaru. Innym sposobem jest też badanie zależności czy korelacji pomiędzy wymiarami. Bada się także wymiary pod kątem utrzymywania pewnego warunku zgodności, jednocześnie wymuszając korzystanie z jak najmniejszej liczby cech. Jeszcze inną rodzinę stanowią metody korzystające z pewnego klasyfikatora (czy aproksymatora), który służy do oceny dokonanego doboru cech (np. [46]).

5.3. Medyczne zastosowania sieci IncNet

Sieć IncNet można stosować zarówno do problemów klasyfikacyjnych, jak i w aproksymacji. W bieżącym podrozdziale przedstawione zostaną zastosowania sieci IncNet do klasyfikacji danych medycznych. Pierwszym zastosowaniem będzie klasyfikacja

[1] Jest to niedopuszczalne w przypadku aproksymacji.

danych psychometrycznych. Zadaniem sieci w tym przypadku jest klasyfikacja osoby do pewnej psychiatrycznej grupy nozologicznej, w oparciu o wyznaczone współczynniki dla danej osoby. W następnym podrozdziale zaprezentowane jest użycie sieci IncNet do klasyfikacji chorób: raka piersi, zapalenia wątroby, cukrzycy, zapalenia wyrostka i chorób tarczycy. Natomiast w kolejnym podrozdziale będzie można prześledzić kilka przykładów zastosowania sieci IncNet w aproksymacji.

5.3.1. Klasyfikacja i analiza danych psychometrycznych

Opis problemu

Psychometryczny test *Minnesota Multiphasic Personality Inventory (MMPI)* [24, 22, 23, 7] jest jednym z najczęściej stosowanych testów, które wspomagają dokonywanie klasyfikacji psychiatrycznych typów nozologicznych. Test MMPI składa się z ponad 550 pytań. Pytania testu dotyczą przeróżnych tematów, związanych z badaną osobą [50] (liczby w nawiasach oznaczają liczbę pytań):

- ogólnego stanu zdrowia (9 pozycji),
- symptomów neurologicznych (19),
- nerwów czaszkowych (11),
- motoryki i koordynacji ruchowej (6),
- wrażliwości (5),
- reakcji wazomotorycznych,
- zaburzeń mowy, problemów wydzielniczych (10),
- problemów systemu krążeniowo-oddechowego (5),
- problemów żołądkowo-jelitowych (11),
- problemów moczowo-płciowych (5),
- nawyków (19),
- spraw rodzinnych i małżeńskich (26),
- problemów zawodowych (18),
- problemów szkolnych (12),
- postaw wobec religii (19),
- postaw politycznych, stosunku do prawa i porządku (46),
- postaw społecznych (72),
- obniżenia nastroju (32),

- podwyższenia nastroju (24),
- stanów obsesyjnych i kompulsywnych (15),
- urojeń, poczucia mocy, halucynacji, iluzji (34),
- fobii (29),
- tendencji sadystycznych i/lub masochistycznych (7),
- morale (33),
- pozycje odnoszące się do męskości-kobiecości (55)
- pozycje wskazujące na to, czy jednostka nie próbowała przedstawić siebie w nadmiernie korzystnym świetle (15).

Na podstawie odpowiedzi na pytania testu konstruuje się skale kontrolne i kliniczne:

Skale kontrolne:

1. ”Na to trudno mi odpowiedzieć” (”?”),
2. ocena stopnia szczerości osób badanych,
3. wykrywanie nietypowych i dewiacyjnych sposobów odpowiadania,
4. wykrywanie subtelniejszych prób zafałszowania profilu

Skale kliniczne:

1. hipochondria,
2. depresja,
3. histeria,
4. psychopatia
5. męskość,
6. paranoja,
7. psychastenia,
8. schizofrenia,
9. mania,
10. introwersja społeczna

Celem testu MMPI jest, na podstawie wyżej przedstawionych cech (w postaci współczynników różnych skal), wspomożenie dokonania klasyfikacji psychiatrycznego typu nozologicznego badanej osoby. Część spośród typów jest wspólna dla kobiet i mężczyzn, natomiast inne typy są zróżnicowane. Jeden z możliwych podziałów dokonany przez J. Gomułę i T. Kucharskiego (Uniwersytet M. Kopernika w Toruniu) przedstawiony jest poniżej:

Typy dotyczące kobiet:

1. nerwica,
2. psychopatia,
3. przestępcy,
4. schizofrenia,
5. psychozy reaktywne,
6. psychozy inwolucyjne,
7. symulacja,
8. dewiacyjne style odpowiedzi (grupa składająca się z 6 klas nozologicznych)

Typy dotyczące mężczyzn:

1. nerwica,
2. psychopatia,
3. alkoholizm,
4. przestępcy,
5. schizofrenia,
6. psychozy reaktywne,
7. symulacja,
8. dewiacyjne style odpowiedzi (grupa składające się z 6 klas nozologicznych)

Typy wspólne:

1. norma,
2. psychopatia,
3. narkomania,
4. organika,

5. zespół urojeniowy,
6. psychozy reaktywne,
7. paranoja,
8. stan hipomaniakalny,
9. symulacja,
10. dyssymulacja

Dane

Ostateczna klasyfikacja typu nozologicznego na podstawie skal kontrolnych i klinicznych jest trudna i wymaga bogatej wiedzy specjalistycznej. Powstało więc pytanie, czy nie można by skonstruować systemu, który mógłby dokonywać automatycznie właściwej klasyfikacji, bazując na wyznaczonych skalach (kontrolnych i klinicznych). W tym celu psycholodzy z Uniwersytetu Mikołaja Kopernika, Jerzy Gomuła i Tomasz Kucharski, opracowali bazy danych w oparciu o liczną grupę pacjentów Akademickiej Poradni Psychologicznej. Bazy te zostały uzupełnione informacje z kilku szpitali psychiatrycznych. Starano się przy tym dobierać odpowiednio liczne grupy osób dla różnych typów nozologicznych. Przestrzegano również różnych ograniczeń, wypływających z założeń przeprowadzania testu MMPI (tj. odpowiedni wiek, nie mniej niż podstawowe wykształcenie, dobry ogólny stan zdrowia). Starano się również, aby zbliżone były do siebie rozkłady związane z takimi zmiennymi jak płeć, wiek, wykształcenie stan cywilny, środowisko, czas trwania choroby oraz charakteru leczenia.

W efekcie powstało kilka baz, które ciągle są rozbudowywane. W poniższych badaniach będą analizowane główne dwie bazy. Każda z nich ma 14 cech, na które składają się skale kontrolne i kliniczne (patrz powyższy opis). Pierwsza baza dotyczy kobiet, a druga mężczyzn. Obie baza zawierają klasy (podklasy) wspólne. Takie klasy oznaczone są poprzez dodanie *–w*. Natomiast klasy kobiet i mężczyzn oznaczone są poprzez dodanie *–k* i *–m* odpowiednio dla kobiet i mężczyzn.

Pierwsza baza składa się z 1027 wektorów, z których każdy może należeć do jednej z 27 klas: norma-w (1), nerwica-w (2), psychopatia-w (3), organika-w (4), schizofrenia-w (5), zespół urojeniowy-w (6), psychoza reaktywna-w (7), psychoza inwolucyjna-w (8), paranoja-w (9), stan (hipo)maniakalny-w (10), przestępcy-w (11), symulacja-w (12), dysymulacja-w (13), narkomania-w (14), norma-k (15), przestępcy-k (16), nerwica-k (17), psychopatia-k (18), organika-k (19), schizofrenia-k (20), symulacja-k (21), dewiacyjny styl odpowiedzi 1-k (22), dewiacyjny styl odpowiedzi 2-k (23), dewiacyjny styl odpowiedzi 3-k (24), dewiacyjny styl odpowiedzi 4-k (25), dewiacyjny styl odpowiedzi 5-k (26), dewiacyjny styl odpowiedzi 6-k(27).

Druga baza składa się z 1167 wektorów, z których każdy może należeć do jednej z 28 klas: norma-w (1), nerwica-w (2), psychopatia-w (3), organika-w (4), schizofrenia-w (5), zespół urojeniowy-w (6), psychoza reaktywna-w (7), psychoza inwolucyjna-w (8), paranoja-w (9), stan (hipo)maniakalny-w (10), przestępcy-w (11), symulacja-w (12),

dysymulacja-w (13), narkomania-w (14), norma-m (15), przestępcy-m (16), nerwica-m (17), psychopatia-m (18), alkoholizm-m (19), organika-m (20), schizofrenia-m (21), symulacja-m (22), dewiacyjny styl odpowiedzi 1-m (23), dewiacyjny styl odpowiedzi 2-m (24), dewiacyjny styl odpowiedzi 3-m (25), dewiacyjny styl odpowiedzi 4-m (26), dewiacyjny styl odpowiedzi 5-m (27), dewiacyjny styl odpowiedzi 6-m (28).

Zawartość baz przedstawiono graficznie na rysunkach 5.1 i 5.2 dla pierwszej bazy, natomiast na rysunkach 5.3 i 5.4 dla drugiej bazy. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom nozologicznym.

Proces uczenia

Jak opisano w podrozdziale 4.3.7, sieć IncNet wykorzystywana do problemów klasyfikacji, składa się z klastra podsieci, a zadaniem każdej z podsieci jest estymacja każdej z klas niezależnie, po czym ostatecznej klasyfikacji dokonuje moduł decyzyjny (który działa w oparciu o zasadę, że zwycięzca bierze wszystko), co ilustruje rys. 4.5. Należy wspomnieć również, że każda z sieci, ucząc się niezależnie, wyznacza w procesie uczenia jak najlepszą dla siebie architekturę, korzystając z mechanizmów kontroli złożoności, dzięki czemu podsieci najczęściej znacznie różnią się pod względem końcowej liczby neuronów (funkcji bazowych).

Poniższe tabele (5.1 i 5.2) prezentują, jak rozkłada się liczba neuronów sieci IncNet w poszczególnych podsieciach. Tabele zawierają informacje uzyskane na podstawie uczenia na zbiorze 27 i 28 klasowym odpowiednio. Proces uczenia trwał 5 epok.

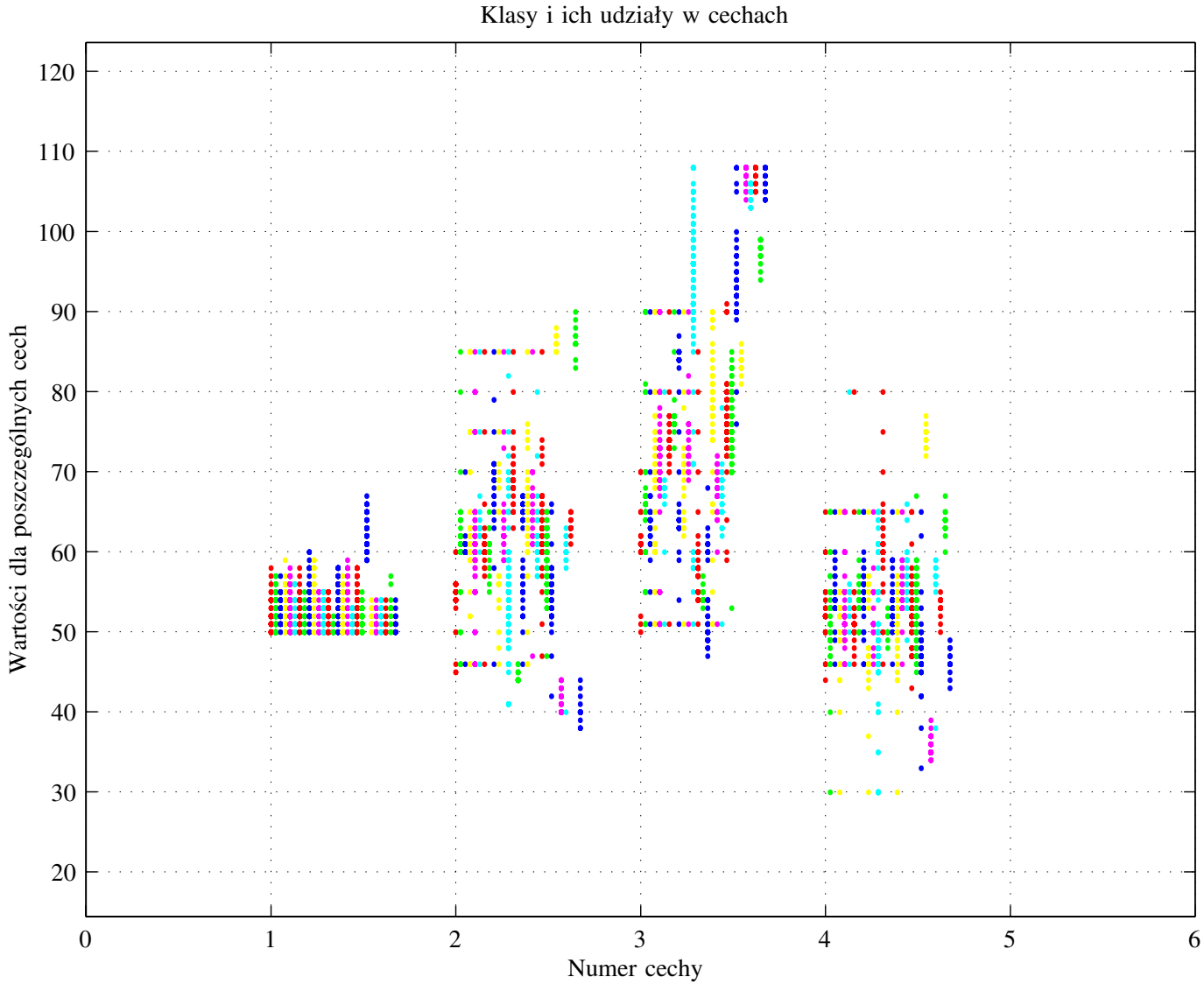
Liczby neuronów w poszczególnych podsieciach																										
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
5	1	4	9	9	6	4	3	8	8	3	11	4	1	8	4	5	8	12	8	1	2	2	1	1	1	1
Całkowita liczba neuronów: 130																										

Tabela 5.1: Rozkład złożoności sieci IncNet dla zbioru 27 klasowego.

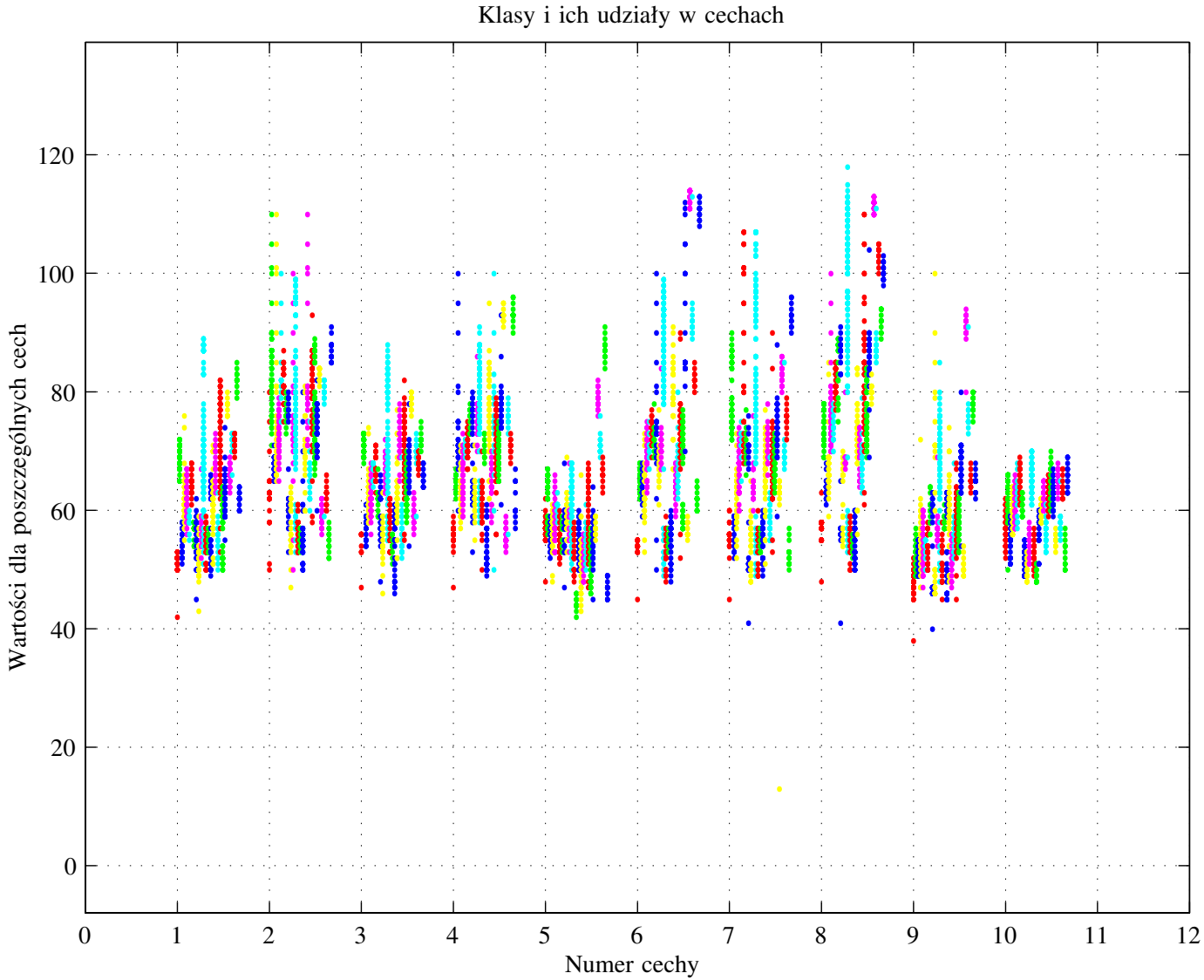
Liczby neuronów w poszczególnych podsieciach																											
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
4	5	4	13	10	2	5	4	7	6	2	15	1	1	6	4	12	6	13	11	11	14	1	1	1	1	1	1
Całkowita liczba neuronów: 162																											

Tabela 5.2: Rozkład złożoności sieci IncNet dla zbioru 28 klasowego.

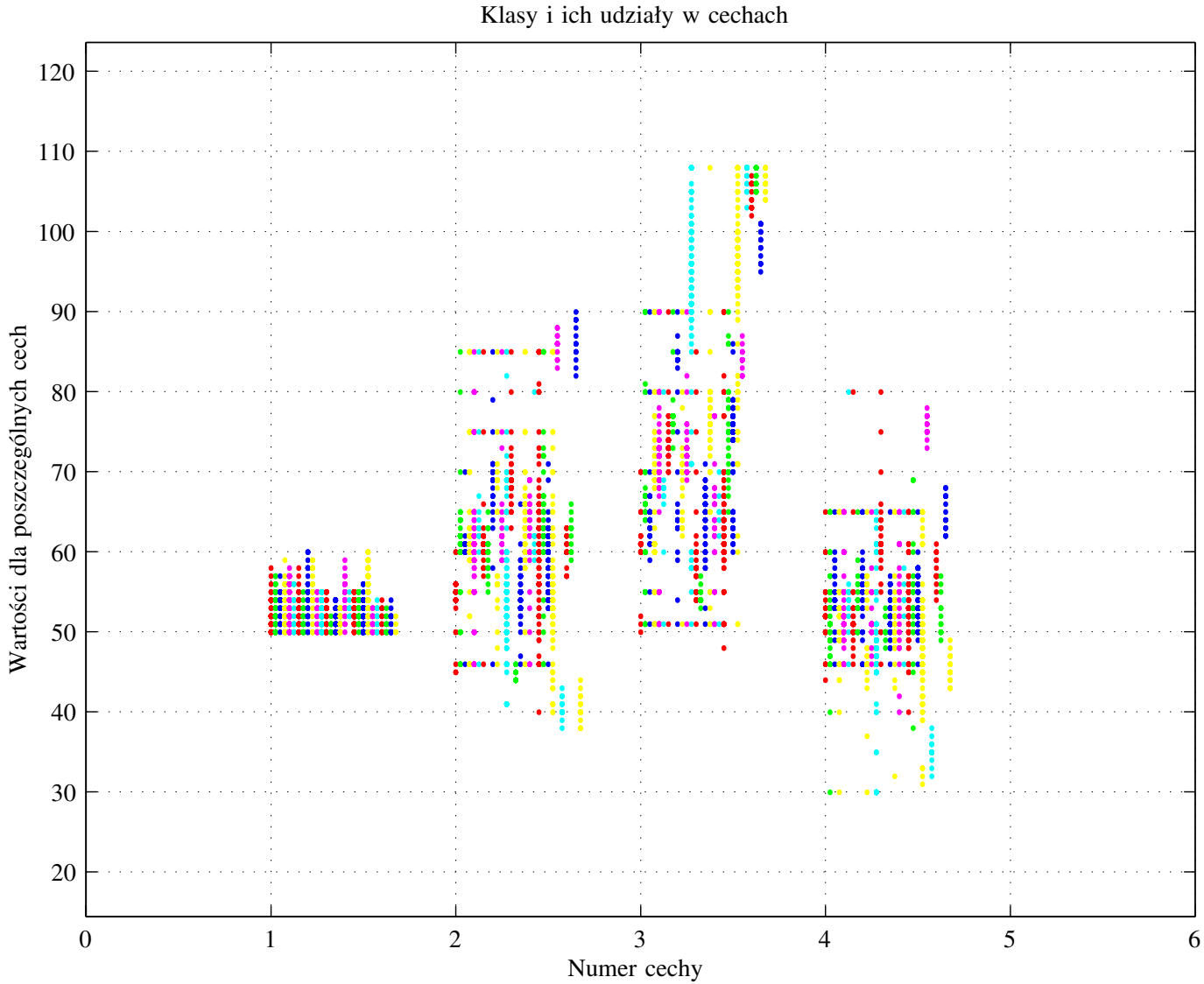
Jak widać, złożoność poszczególnych podsieci jest znacznie zróżnicowana i waha się od 1 neuronu do 15 neuronów. Dowodzi to, że kontrola złożoności powinna być wbudowana w mechanizm uczenia i działać możliwie sprawnie.



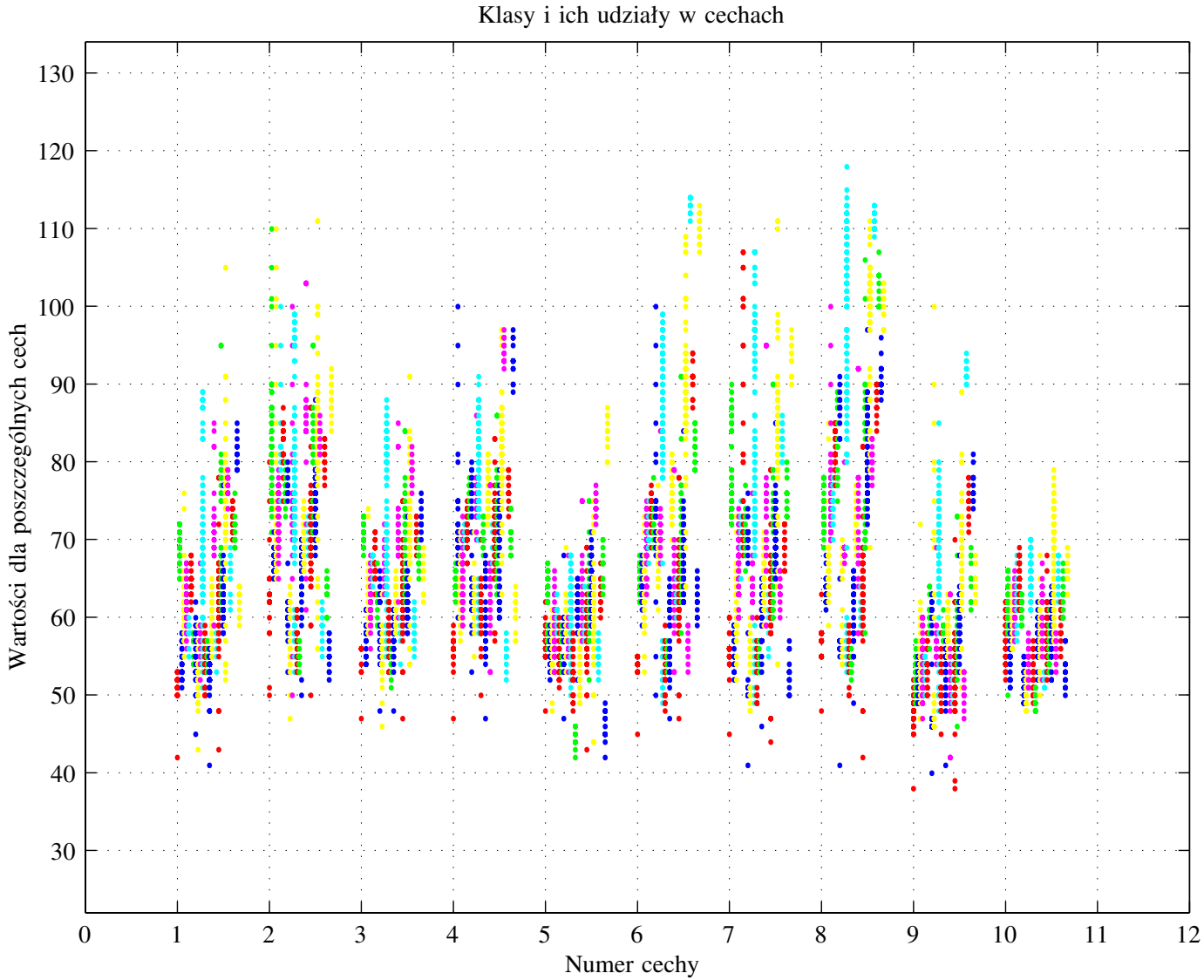
Rysunek 5.1: Pierwsza baza danych psychometrycznych (1027 wektorów, 27 klas, 14 wymiarowe wejście). Rysunek ukazuje pierwsze cztery cechy — skale kontrolne.



Rysunek 5.2: Pierwsza baza danych psychometrycznych (1027 wektorów, 27 klas, 14 wymiarowe wejście). Rysunek ukazuje 10 kolejnych cech — skale kliniczne.



Rysunek 5.3: Druga baza danych psychometrycznych (1167 wektorów, 28 klas, 14 wymiarowe wejście). Rysunek ukazuje pierwsze cztery cechy — skale kontrolne.



Rysunek 5.4: Druga baza danych psychometrycznych (1167 wektorów, 28 klas, 14 wymiarowe wejście). Rysunek ukazuje 10 kolejnych cech — skale kliniczne.

Zmienność liczby neuronów w procesie uczenia, jak i zmianę wartości błędu treningowego i testowego, dla kilku wybranych podsieci, można przeanalizować na kolejnych rysunkach. Widzimy na nich zmiany, które zostały zebrane w 25 punktach kontrolnych (czyli co około 200 iteracji, każda sieć była uczona 5 epok).

Rysunek 5.5 przedstawia przykładowy proces uczenia dla 5-tej i 16-tej klasy 27-klasowej bazy danych. Należy zwrócić uwagę, że jednostką czasu jest tu jedna iteracja, czyli prezentacja jednego wektora treningowego. Czarna krzywa obrazuje liczbę neuronów, czerwona i zielona pokazują poprawność klasyfikacji dla zbioru treningowego i testowego.

Z kolei rysunek 5.6 pokazuje proces uczenia 20-tej klasy dla zbioru 28-klasowego (tutaj kolor czerwony pokazuje poprawność dla zbioru treningowego). Kolejny rysunek – 5.7, otrzymano również na podstawie uczenia dla 28-klasowego zbioru danych. Dokładniej obrazuje on proces uczenia 9-tej klasy.

Porównanie i analiza wyników

W celach porównawczych zostały zebrane rezultaty uzyskane za pomocą różnych metod klasyfikacji, jak i metod wyciągania reguł logicznych. Sieć IncNet została porównana z siecią FSM [1, 44], która była wykorzystana jako klasyfikator (FSM z funkcjami Gaussa) i jako metoda wyciągania reguł logicznych (FSM z funkcjami prostokątnymi i FSM z funkcjami prostokątnymi i bicentralnymi z optymalizacją[2]). Do porównania użyto także metody uczenia maszynowego C 4.5 [134] do ekstrakcji reguł logicznych. Były wykonywane próby klasyfikacji przy użyciu innych metod, ale ich rezultaty były istotnie gorsze od zaprezentowanych w poniżej opisanych tabelach 5.3 i 5.4.

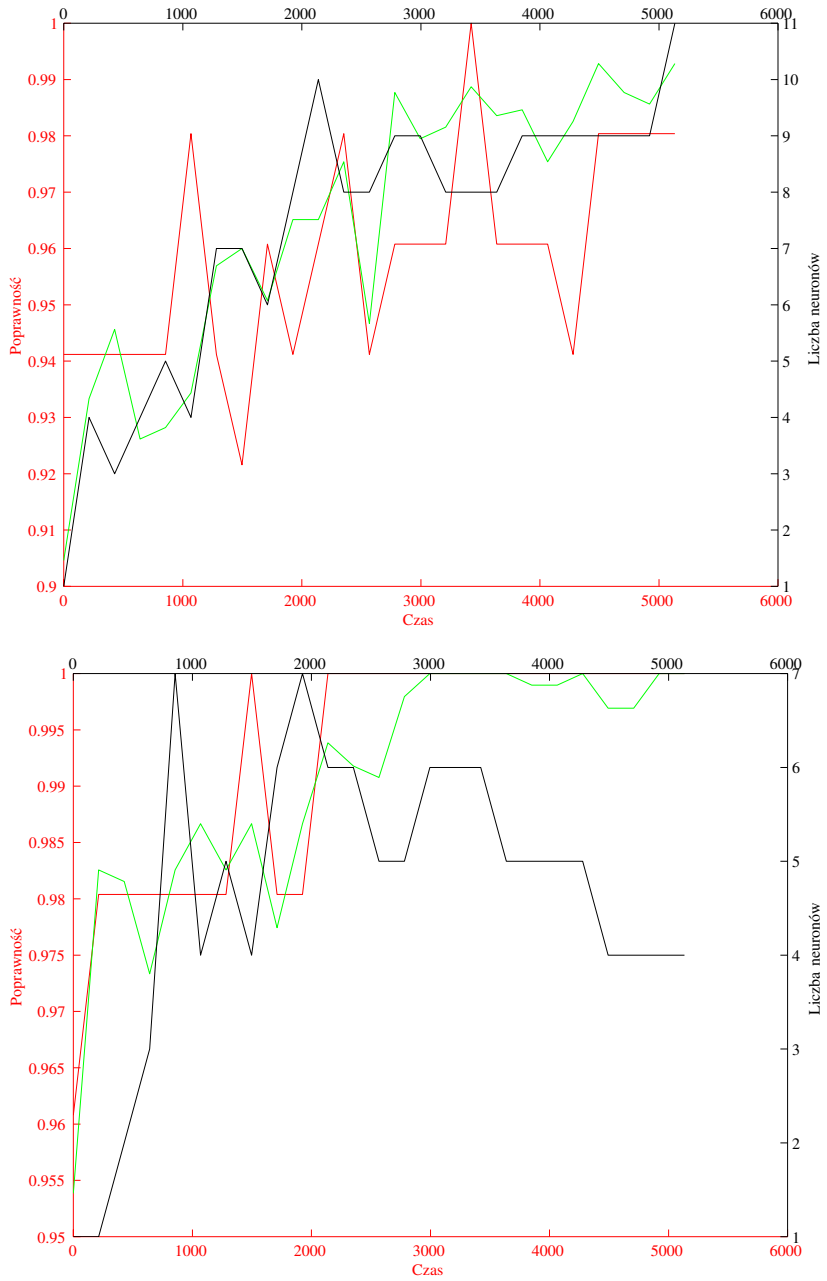
W tabeli 5.3 zostało ukazane porównanie, w którym wszystkie modele korzystały przy uczeniu z całego zbioru 27- i 28-klasowego odpowiednio.

Model	Uczenie na całym zbiorze	
	27 klasowym	28 klasowym
IncNet	**99.22**	**99.23**
C 4.5	93.67	93.06
FSM+R Opt.	97.57	96.91

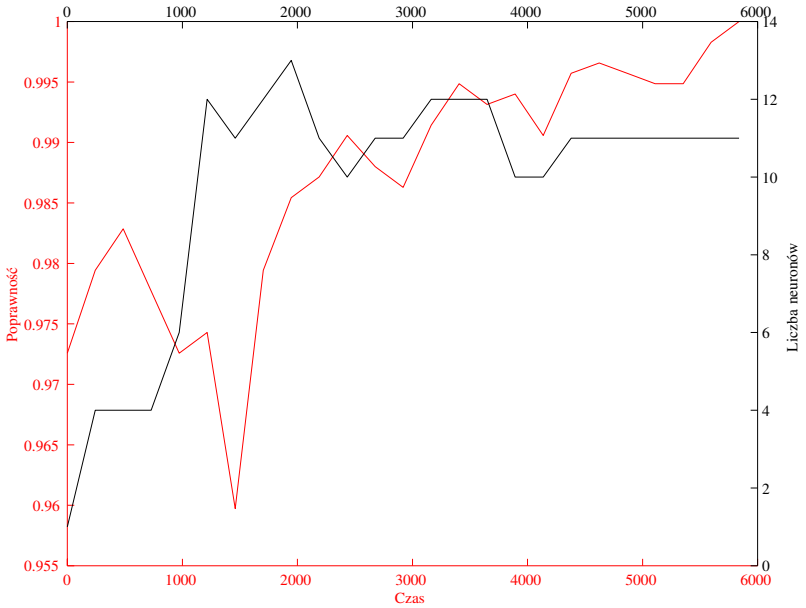
Tabela 5.3: Poprawność klasyfikacji w procentach dla różnych modeli adaptacyjnych. Modele były uczone na całym zbiorze 27- i 28-klasowym.

Tabela 5.4 porównuje możliwości generalizacji wyżej wspomnianych modeli z siecią IncNet. Tak jak i poprzednio użyto obu zbiorów (27- i 28-klasowego). Tabela prezentuje rezultaty uzyskane po uczeniu dla dwóch różnych podziałów. Pierwszy podział to

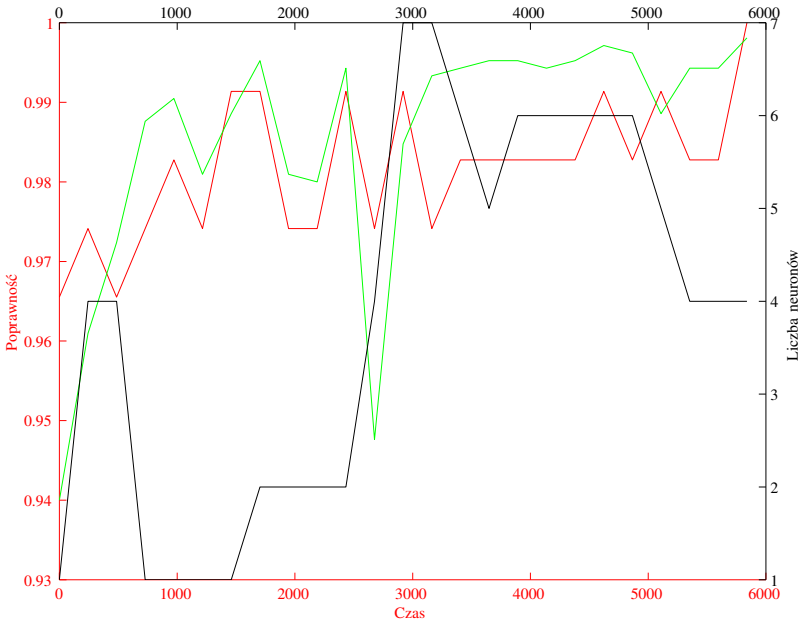
[2]FSM+R Opt. to reguły *miękkie* uzyskane w procesie optymalizacji. Takie *miękkie* reguły nie dają odpowiedzi typu TAK/NIE, lecz wartość z zakresu $[0, 1]$.



Rysunek 5.5: Wykres ilustruje zmieniającą się w czasie poprawność klasyfikacji dla zbioru treningowego (kolor zielony) i zbioru testowego (kolor czerwony), jak i liczbę neuronów (kolor czarny). Dane dla zbioru 27-klasowego, klasy 5-tej i 16-tej. [Jednostką czasu jest prezentacja pojedynczego wektora.]



Rysunek 5.6: Wykres ilustruje zmieniającą się w czasie poprawność klasyfikacji dla zbioru treningowego (kolor czerwony), jak i liczbę neuronów (kolor czarny). Dane dla zbioru 28-klasowego, klasy 20-tej.



Rysunek 5.7: Wykres ilustruje zmieniającą się w czasie poprawność klasyfikacji dla zbioru treningowego (kolor zielony) i zbioru testowego (kolor czerwony), jak i liczbę neuronów (kolor czarny). Dane dla zbioru 28-klasowego, klasy drugiej.

90% na zbiór treningowy i 10% na zbiór testowy. Drugi to 95% na zbiór treningowy i 5% na zbiór testowy.

Model	zbiór 27-klasowy				zbiór 28-klasowy			
	10%		5%		10%		5%	
	TRS	TES	TRS	TES	TRS	TES	TRS	TES
IncNet	**99.03**	**93.14**	**98.77**	**96.08**	**98.95**	**93.10**	**98.29**	**94.83**
FSM + G	97.65	92.16			97.08	90.20		
FSM + R	96.97	84.70			96.66	82.76		
C 4.5	93.22	83.70			93.13	78.90		

Tabela 5.4: Porównanie poprawność klasyfikacji w procentach danych psychometrycznych dla różnych modeli adaptacyjnych. 10% (lub 5%) oznacza, że 10% (lub 5%) wzorców branych jest do zbioru testowego, a reszta (90% lub 95%) wzorców stanowi zbiór treningowy.

Przedstawione rezultaty pokazują, że sieć IncNet nie tylko uzyskała najlepsze wyniki na zbiorze treningowym, ale przede wszystkim na zbiorze testowym. W tabeli 5.4 widać drastyczną różnice w generalizacji pomiędzy siecią IncNet i regułami logicznych uzyskanymi z modeli C 4.5 i FSM+R. Różnica w poprawności klasyfikacji na zbiorze testowym wyniosła około 10%.

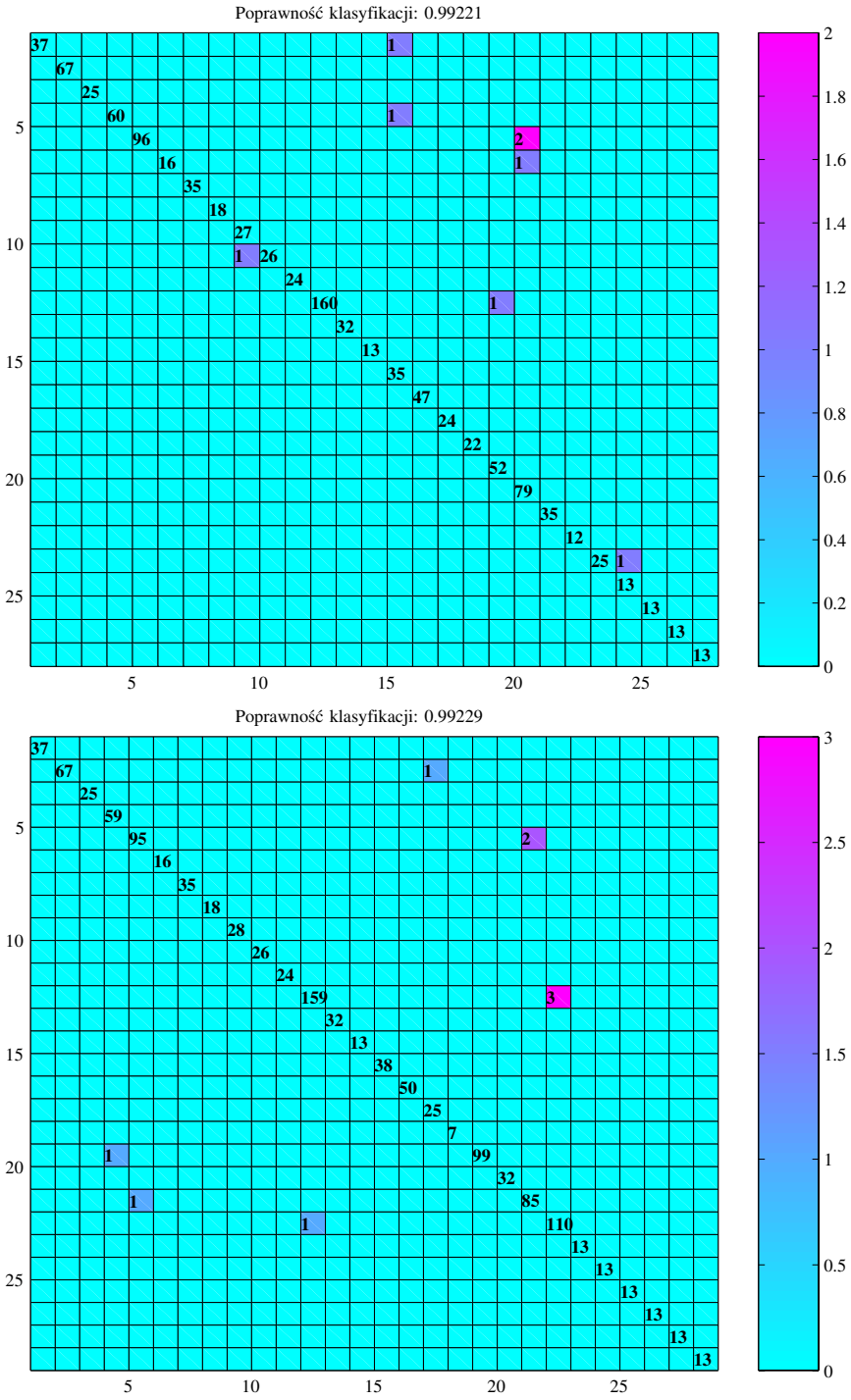
Sprawność sieci IncNet można również prześledzić, analizując macierze rozrzutu, które zostały wyznaczone dla sieci, powstałych przez uczenie na różnych danych i przy różnym podziale na część treningową i testową.

Rysunek 5.8 prezentuje macierze rozrzutu, powstałe przy uczeniu sieci IncNet na całym zbiorze 27-klasowym (u góry) i 28-klasowym (u dołu). Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 145. Oś rzędnych odpowiada numerom klas uzyskanych w wyniku klasyfikacji, natomiast oś odciętych odpowiada numerom klas oczekiwanych. Kolejne dwa rysunki 5.9 i 5.10 pokazują macierze rozrzutu zbioru testowego i treningowego dla 27-klasowego zbioru. Macierze na pierwszym rysunku są wynikiem uczenia sieci IncNet przy podziale 90% + 10% (zbiór treningowy i testowy). Natomiast macierze na drugim rysunku są wynikiem podziału 95% + 5%. Ostatnie dwa rysunki z macierzami rozrzutu 5.11 i 5.12 zostały opracowane tak, jak poprzednie dwa, z wyjątkiem, iż użyto 28-klasowego zbioru danych.

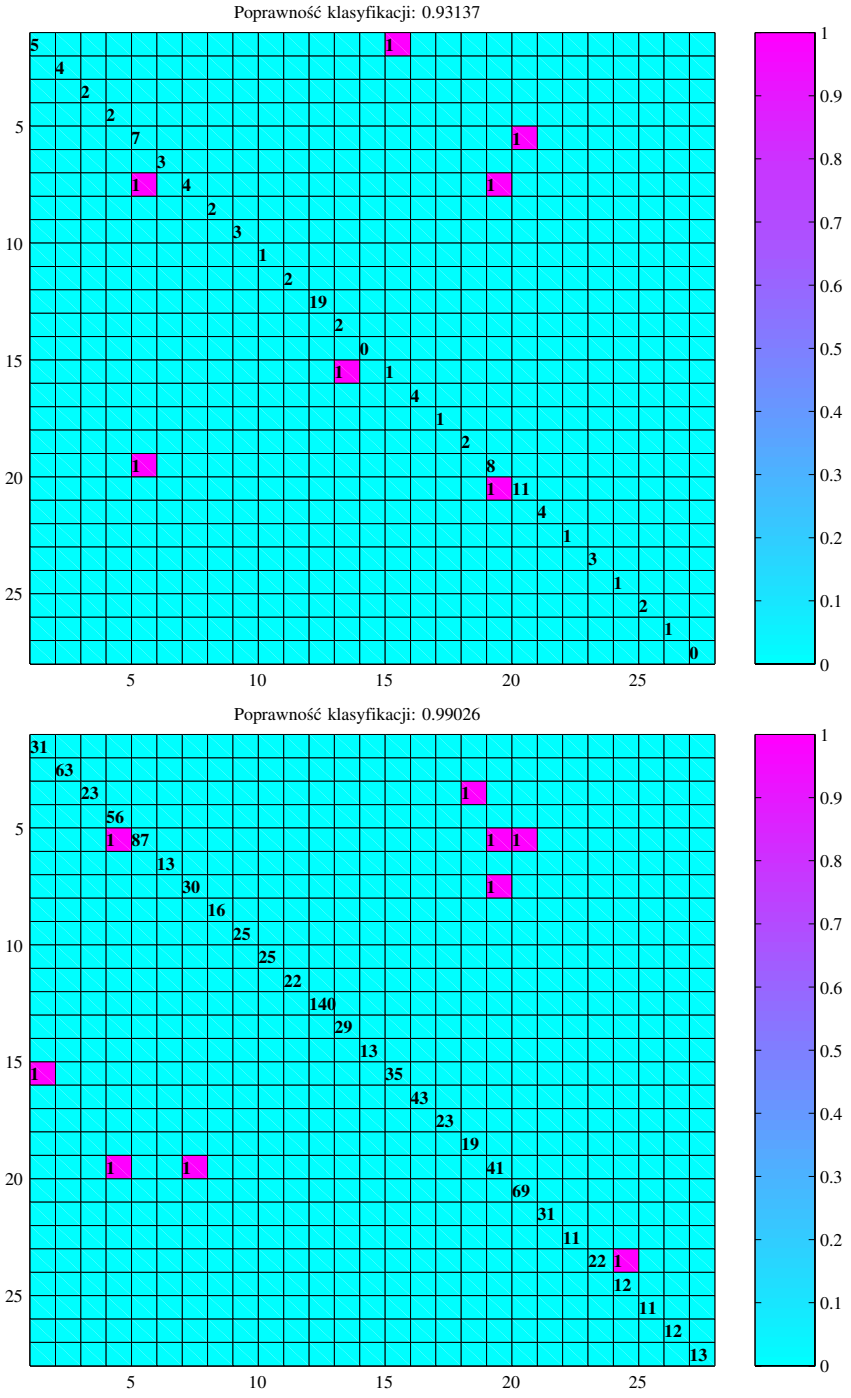
Bardzo ciekawa jest analiza wektorów, które przez sieć zostały źle sklasyfikowane. Na kilku kolejnych rysunkach będzie można porównać wartość otrzymaną na wyjściu sieci *zwycięskiej* klasy z wartością dla klasy oczekiwanej (tj. prawidłowej).

Jak już zostało opisane w podrozdziale 4.3.7, można prowadzić obserwację wartości wyjściowych $C^i(\mathbf{x})$ poszczególnych podsieci (patrz rysunek 4.5), bądź dokonać renormalizacji i obserwować (przybliżone) prawdopodobieństwa $p(C^i|\mathbf{x})$ przynależności do danych klas (patrz równanie 4.119).

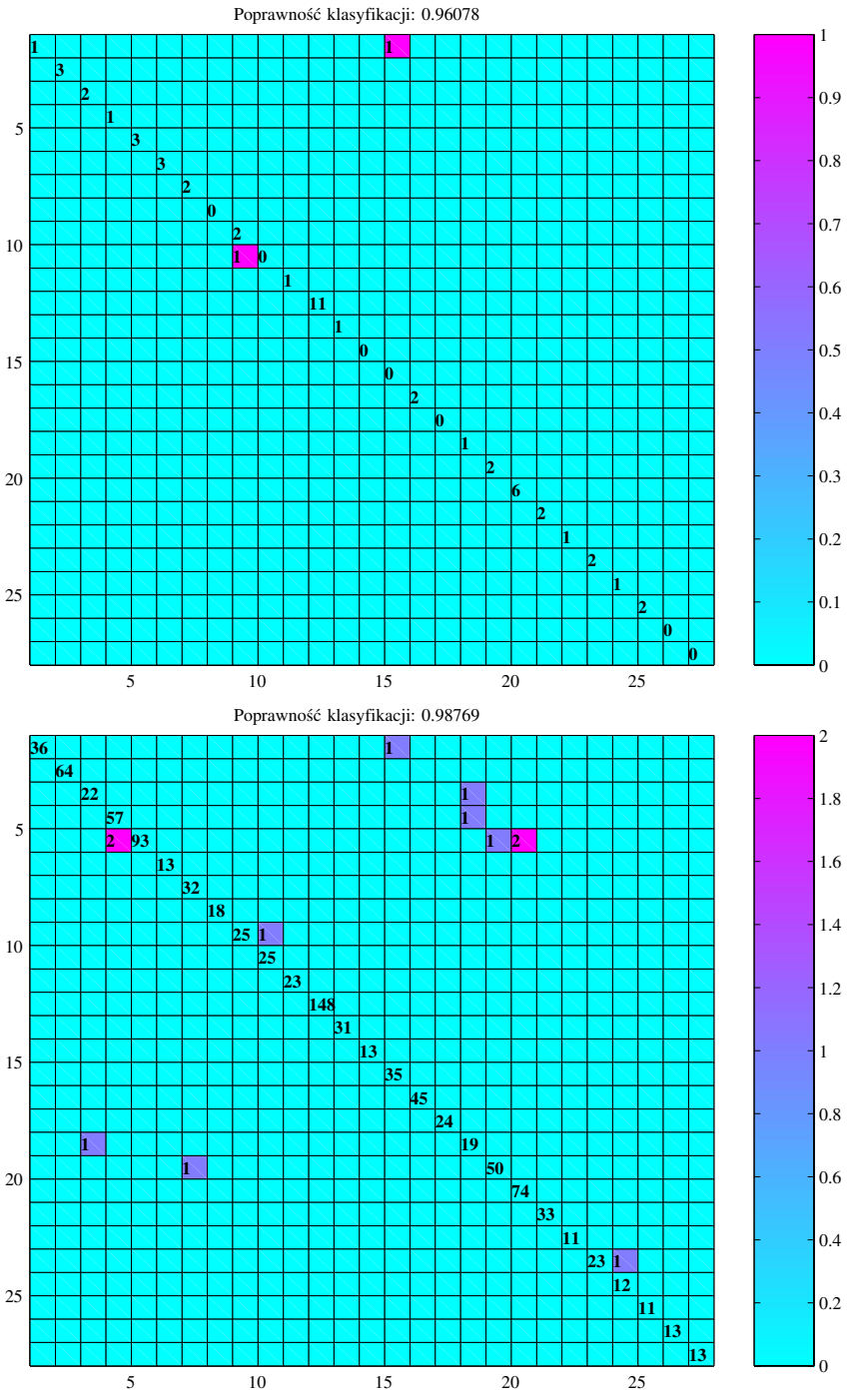
Wartości na lewej osi rysunków 5.13, 5.14, 5.15 i 5.16 odpowiadają wartością dla



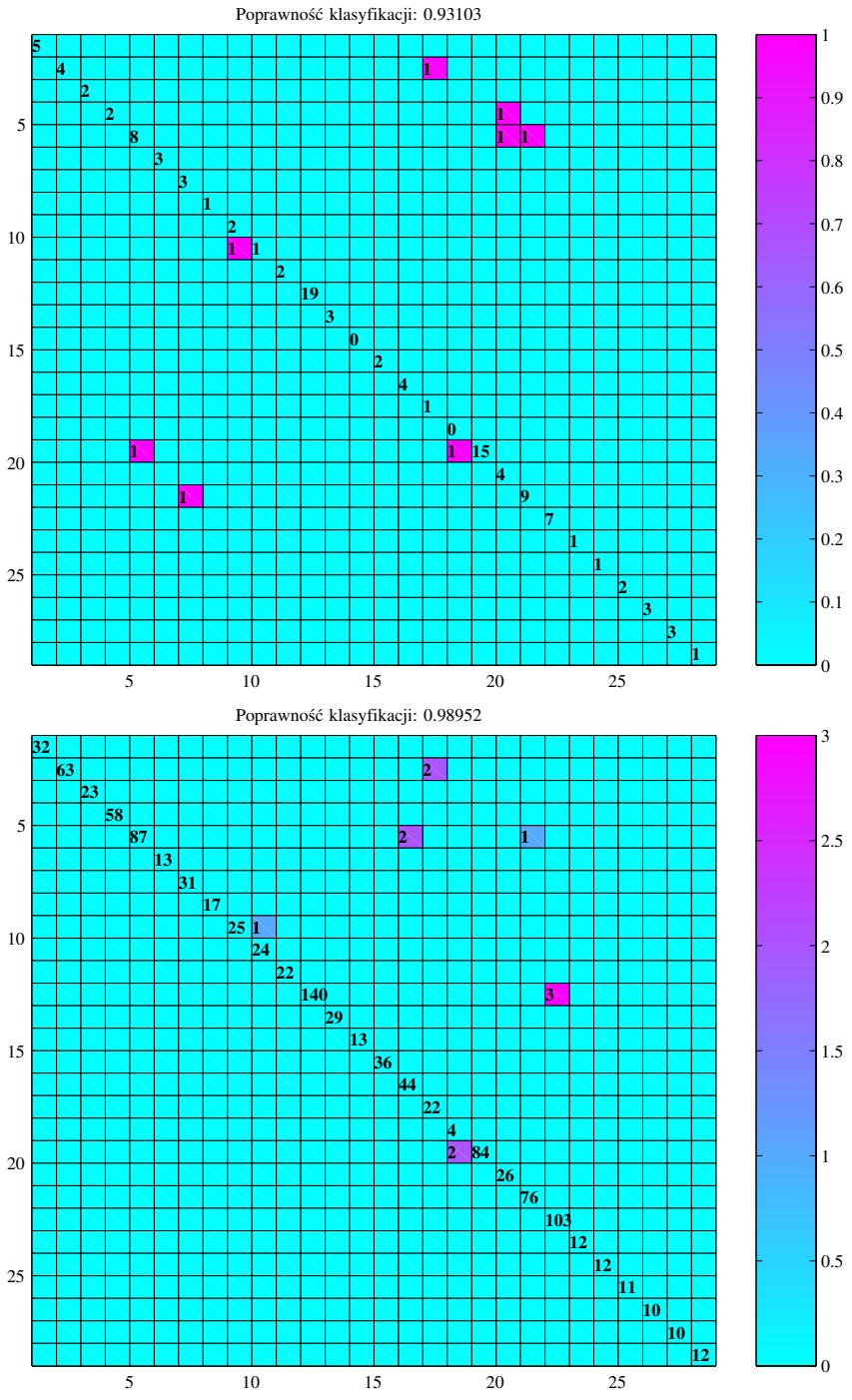
Rysunek 5.8: Macierze rozrzutu powstałe przy uczeniu na całym zbiorze. U góry dla 27-klasowego zbioru, u dołu dla 28-klasowego zbioru. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 145.



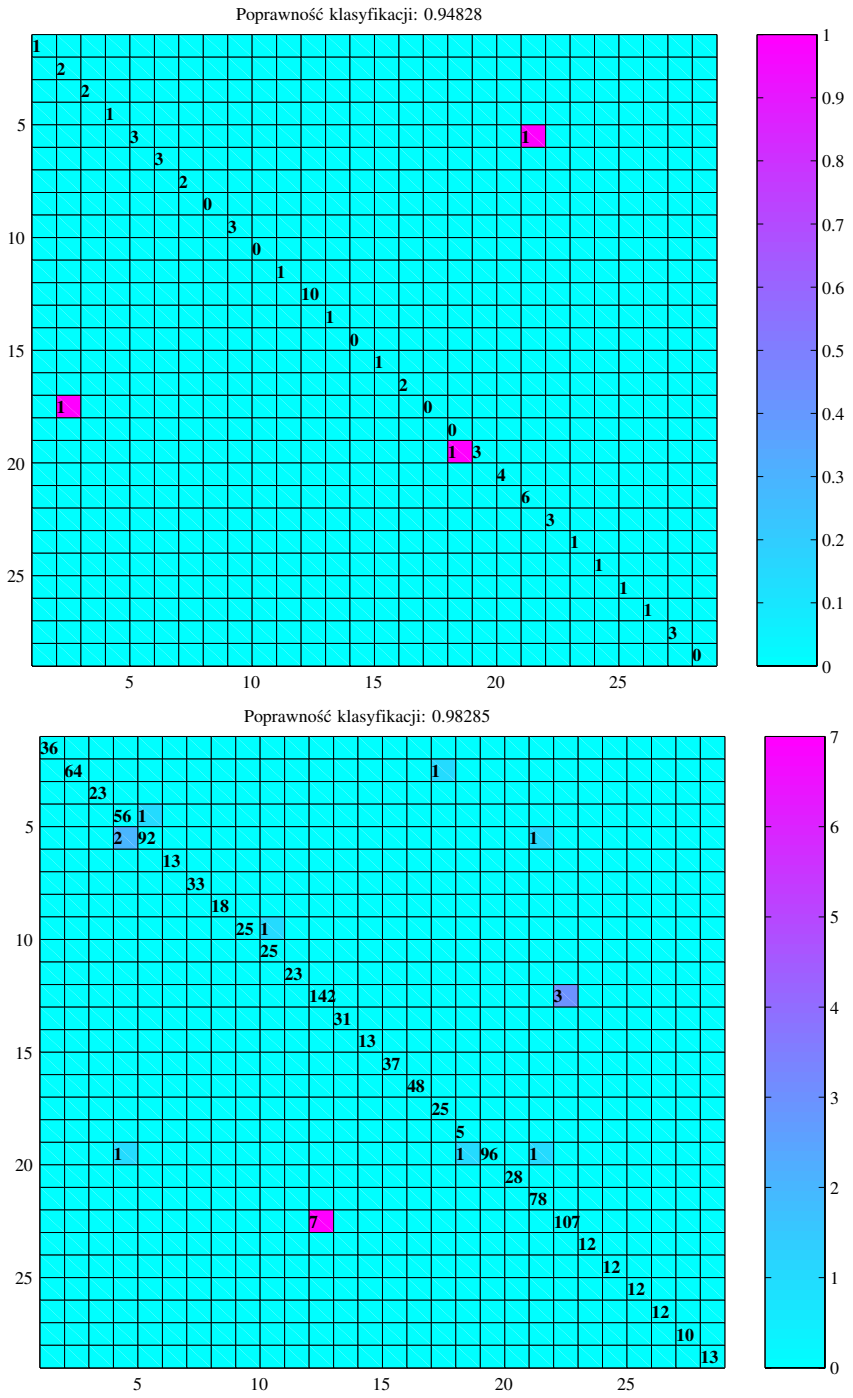
Rysunek 5.9: Macierze rozrzutu powstałe przy uczeniu na 90%-owej części zbioru 27-klasowego. U góry dla zbioru testowego, u dołu dla zbioru treningowego. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 145.



Rysunek 5.10: Macierze rozrzutu powstałe przy uczeniu na 95%-owej części zbioru 27-klasowego. U góry dla zbioru testowego, u dołu dla zbioru treningowego. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 145.



Rysunek 5.11: Macierze rozrzutu powstałe przy uczeniu na 90%-owej części zbioru 28-klasowego. U góry dla zbioru testowego, u dołu dla zbioru treningowego. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 145.



Rysunek 5.12: Macierze rozrzutu powstałe przy uczeniu na 95%-owej części zbioru 28-klasowego. U góry dla zbioru testowego, u dołu dla zbioru treningowego. Wartości rzędnych i odciętych oznaczają numery poszczególnych klas, patrz opis na str. 145.

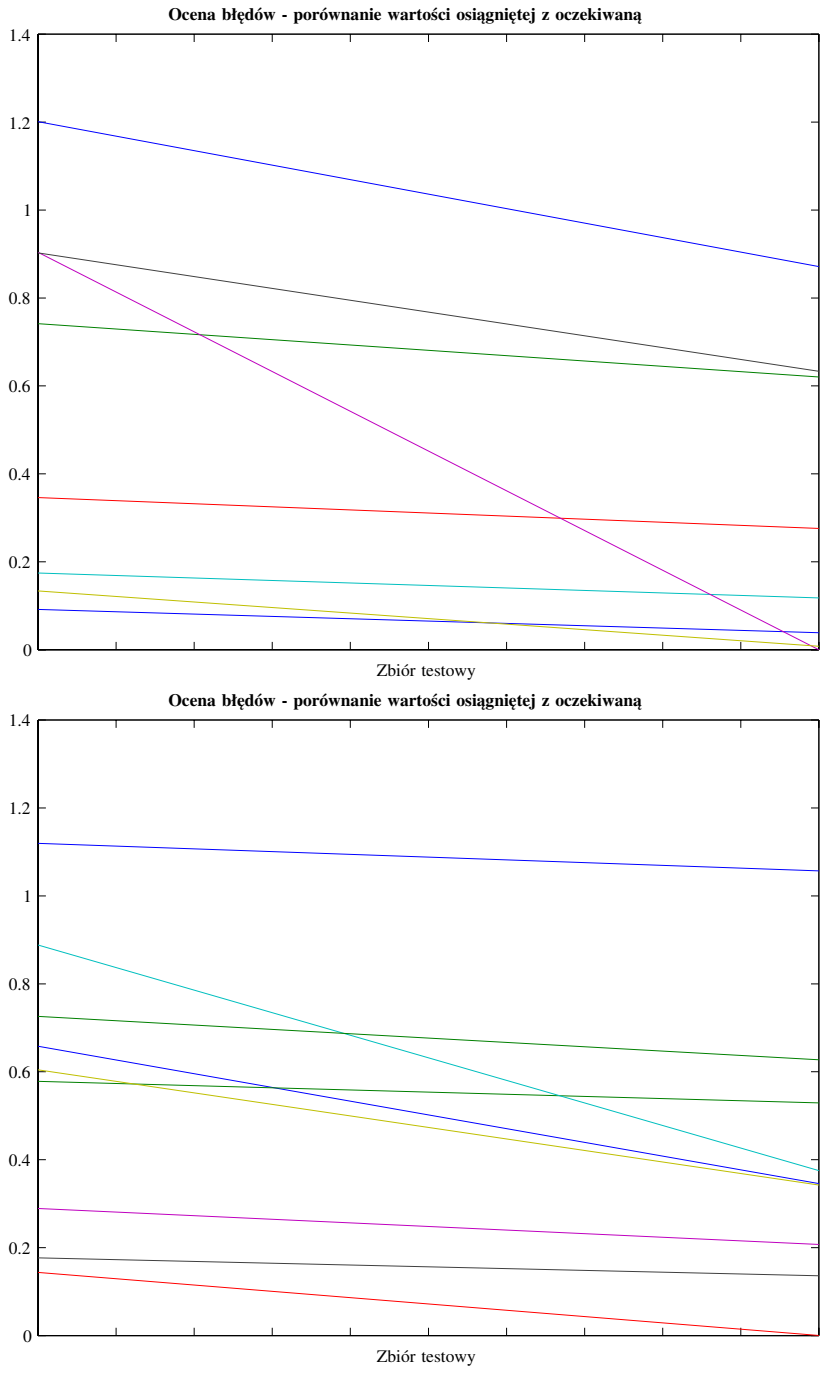
klasy zwycięskiej, a na osi prawej widać wartość dla klasy oczekiwanej. Rysunek 5.13 prezentuje wartości wyjściowe klasy zwycięskiej i oczekiwanej wektorów źle sklasyfikowanych dla 27- (u góry) i 28- (u dołu) -klasowego zbioru danych. Sieci były uczone na wszystkich danych. Drugi rysunek 5.14 w odróżnieniu od poprzedniego prezentuje wartości prawdopodobieństw dla klasy zwycięskiej i oczekiwanej. Kolejne dwa rysunki 5.15 i 5.16 prezentują również wartości prawdopodobieństw dla źle sklasyfikowanych wektorów dla zbiorów 27- i 28-klasowych. Sieci były uczone na 90% wektorów danych, a 10% danych stanowiło część testową. U góry rysunku przedstawione są źle sklasyfikowane wektory części testowej, a u dołu części treningowej.

Na rysunkach 5.13, 5.14, 5.15 i 5.16 większość linii łączących wartości dla klasy zwycięskiej i oczekiwanej nie są strome. Oznacza to, że klasyfikacja danego przypadku wcale nie była taka jednoznaczna. Tak może być nie tylko dla źle sklasyfikowanych przypadków. Taka niejednoznaczność pokazuje, że nie ma jedynego idealnego rozwiązania, wskazując alternatywne diagnozy, co nie jest odosobnionym przypadkiem w realnych problemach, z jakimi mamy do czynienia. Z niejednoznacznością można mieć do czynienia nie tylko dla źle sklasyfikowanych przypadków, ale i dla dobrze sklasyfikowanych przypadków, co również jest bardzo ważne. Niejednoznaczność przy dobrze nauczonym modelu oznacza, że mamy do czynienia z przypadkiem, który znajduje się *pomiędzy* dwoma obszarami, należącymi do różnych klas (lub znajduje się w obszarze wspólnym klas). Dlatego właśnie należy nie tylko brać pod uwagę zwycięską klasę, ale i rozwiązania alternatywne, które czasem mogą okazać się niemal tak samo prawdopodobne. W przypadku rozpatrywanych danych psychometrycznych jest to bardzo istotne (np. umożliwia płynne śledzenie zmian chorobowych). Obserwując wartości klas, dla których wyznaczone prawdopodobieństwo jest istotnie większe od zera, uwidacznia się właściwy rozkład udziału klasyfikowanego wektora na poszczególne klasy. W przypadku wyraźnego odstępstwa wartości oczekiwanych od otrzymanych (strome krzywe) jest całkiem prawdopodobne, że to psycholog dokonał niewłaściwej diagnozy.

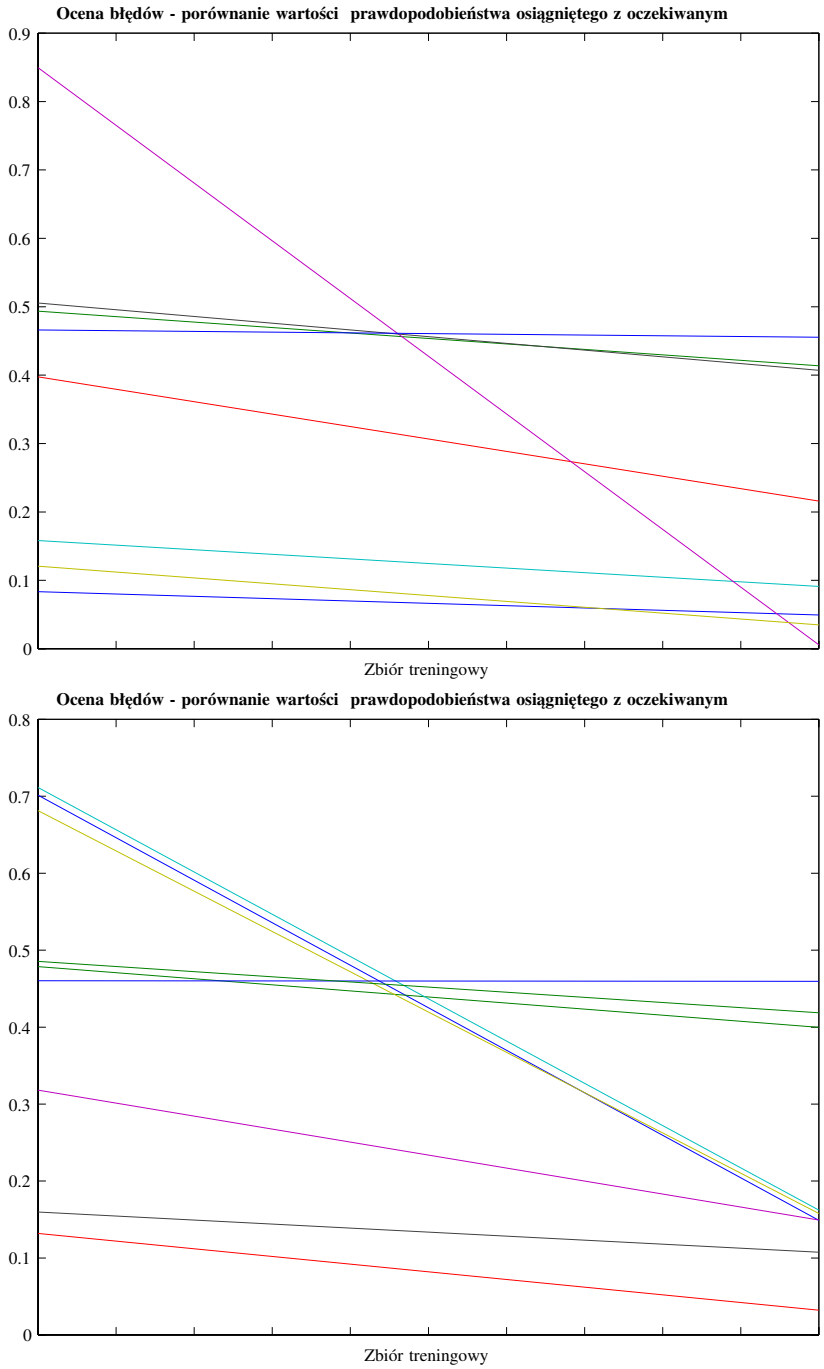
Niżej opisany przypadek pokazuje, że w pewnych skrajnych przypadkach oprócz prawdopodobieństw $p(C^i|\mathbf{x})$ (por. równanie 4.119) przydatna może okazać się także maksymalna spośród wartości $C^i(\mathbf{x})$ ($i = 1, 2, \ldots, K$). Gdy dla pewnego wektora prawdopodobieństwo nie jest skumulowane wokół jednej klasy lub kilku klas, może to oznaczać, że wektor jest spoza obszaru aktywności wszystkich klas. Czyli leży daleko od wektorów treningowych. Odpowiada to przypadkowi całkiem odmiennemu od tych, jakie obecne są w bazie treningowej. W takim przypadku maksymalna wartości spośród $C^i(\mathbf{x})$ ($i = 1, 2, \ldots, K$) będzie miała małą wartość.

Inne, bardzo ciekawe wyniki można uzyskać wyznaczając przedziały ufności i probabilistyczne przedziały ufności, opisane w podrozdziale 4.3.9. Jak już zostało wspomniane wcześniej, przedziały ufności i ich probabilistyczna odmiana są bardzo efektywnym narzędziem wspomagania procesu diagnozy i wizualizacji procesu klasyfikacji.

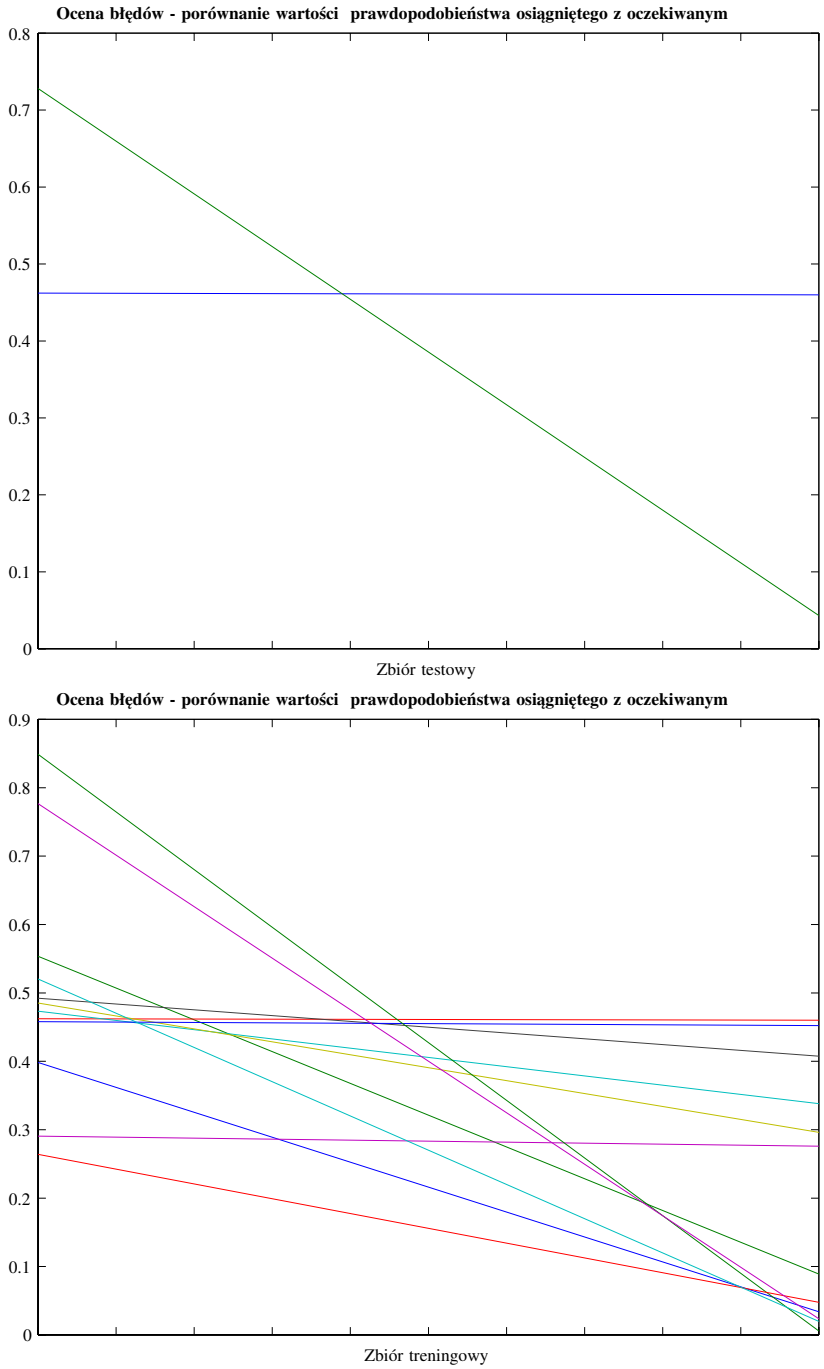
Poniżej przedstawione zostaną różne przykłady z psychometrycznych baz danych. Pierwszy niech będzie nawiązaniem do pierwszego przypadku z podrozdziału 4.3.9. Ten przypadek okazał się mniej jednoznaczny, ale zastał dobrze sklasyfikowany przez



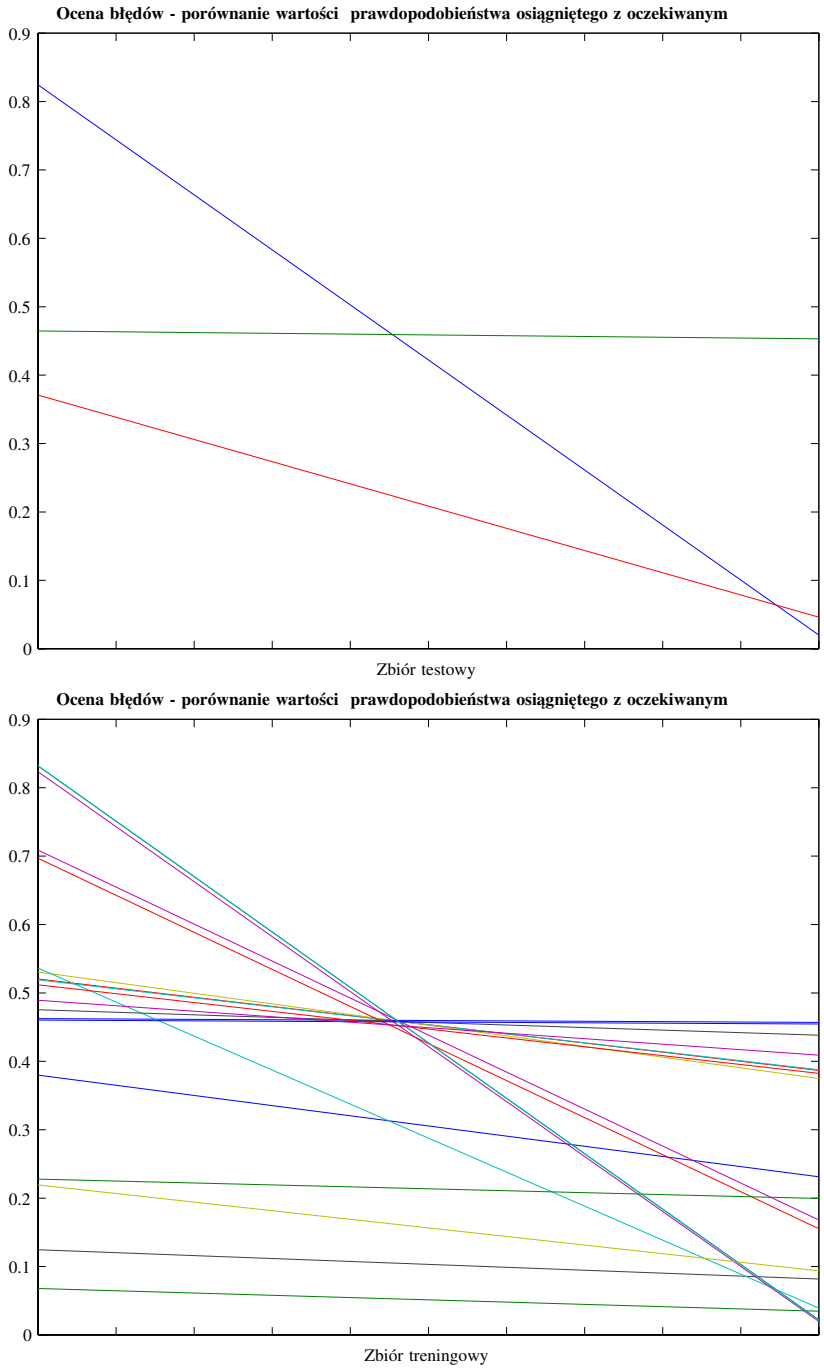
Rysunek 5.13: Porównanie wartości uzyskanych i oczekiwanych na podstawie błędnie sklasyfikowanych wektorów dla 27- i 28-klasowej bazy.



Rysunek 5.14: Porównanie wartości prawdopodobieństw uzyskanych i oczekiwanych na podstawie błędnie sklasyfikowanych wektorów dla 27- i 28-klasowej bazy.



Rysunek 5.15: Porównanie wartości prawdopodobieństw uzyskanych i oczekiwanych na podstawie błędnie sklasyfikowanych wektorów dla 27-klasowej bazy. Sieć uczona była na 95%-wej części danych. U góry dla zbioru testowego, u dołu dla zbioru treningowego.

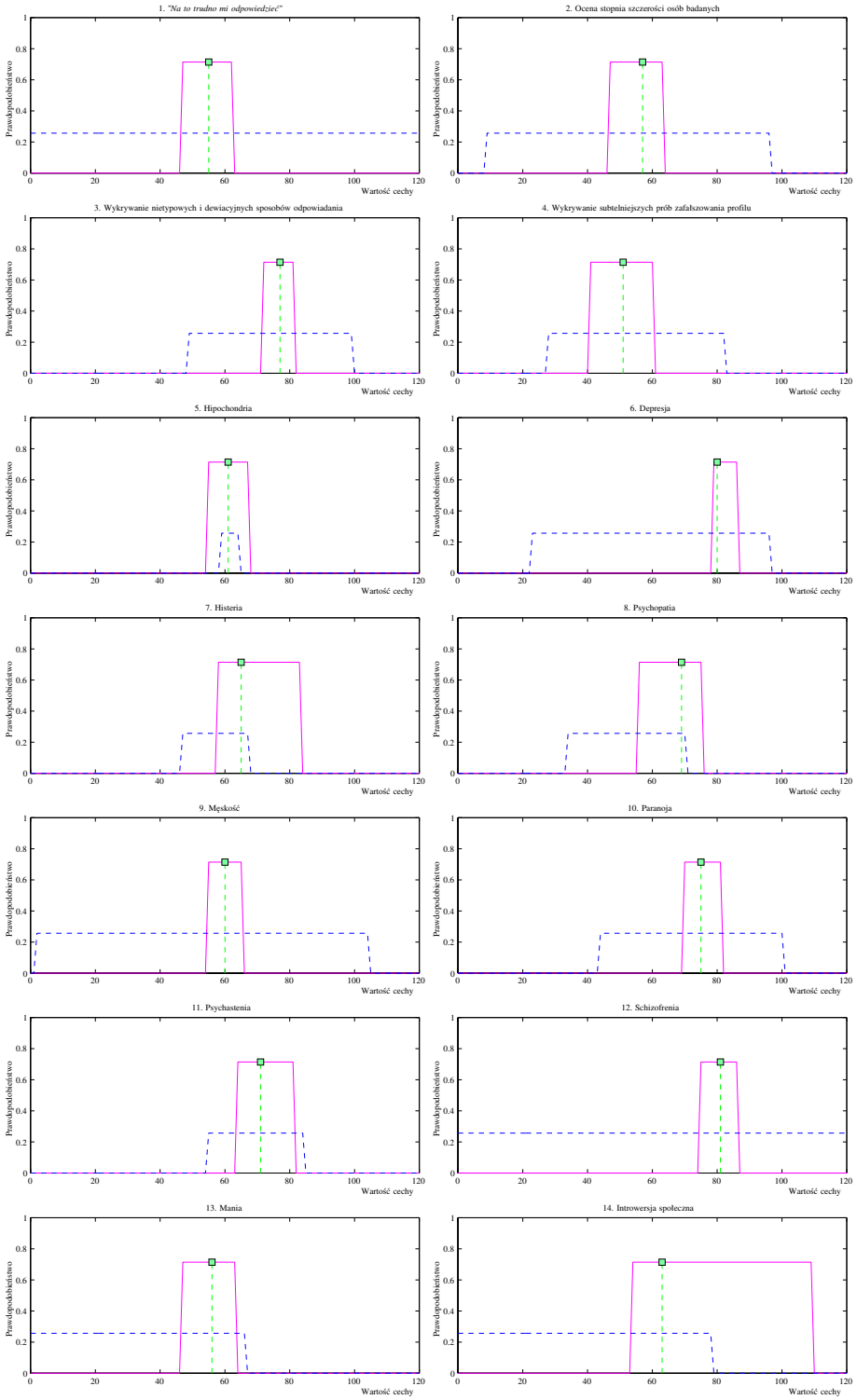


Rysunek 5.16: Porównanie wartości prawdopodobieństw uzyskanych i oczekiwanych na podstawie błędnie sklasyfikowanych wektorów dla 28-klasowej bazy. Sieć uczona była na 95%-wej części danych. U góry dla zbioru testowego, u dołu dla zbioru treningowego.

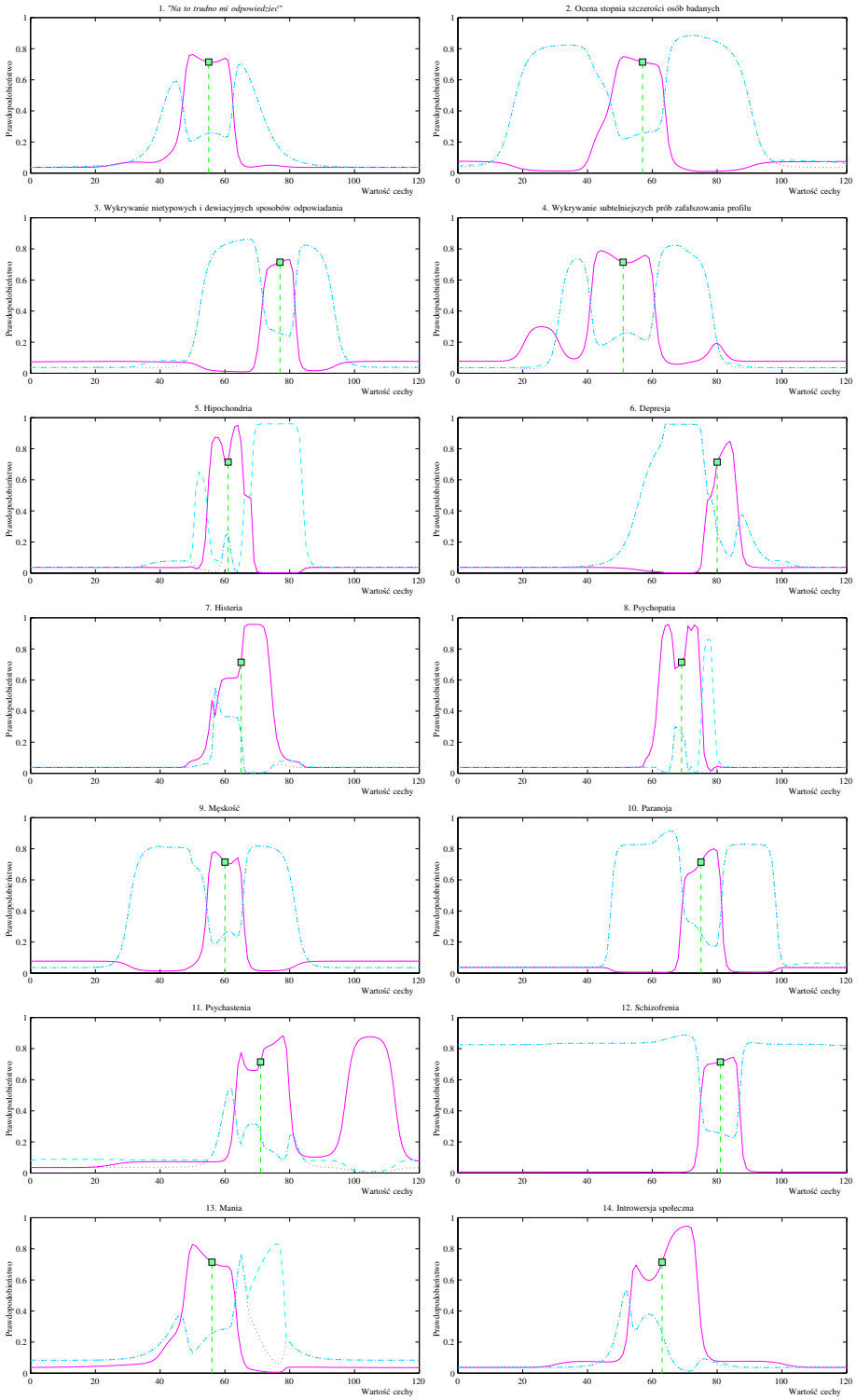
sieć. Przypadek został zaklasyfikowany do psychoz reaktywnych, uzyskując prawdopodobieństwo 0.71, natomiast do klasy schizofrenii z prawdopodobieństwem 0.26. Rysunek 5.17 przedstawia przedziały ufności, a rysunek 5.18 przedstawia probabilistyczne przedziały ufności. Szczególnie wyraźnie widać różnicę pomiędzy rysunkiem 4.8, a rysunkiem 5.18. Bez jakichkolwiek trudności można ocenić jednoznaczność procesu klasyfikacji i wpływ poszczególnych cech. Rysunek 5.18 uwidacznia nakładanie się rozkładów klas psychoz reaktywnych i schizofrenii.

Kolejny przykład został opracowany na podstawie przypadku, który został jednoznacznie zaklasyfikowany do zespołów urojeniowych. Prawdopodobieństwo klasyfikacji wyniosło 0.96. Przedziały ufności zostały przedstawione na rysunku 5.19 i 5.20.

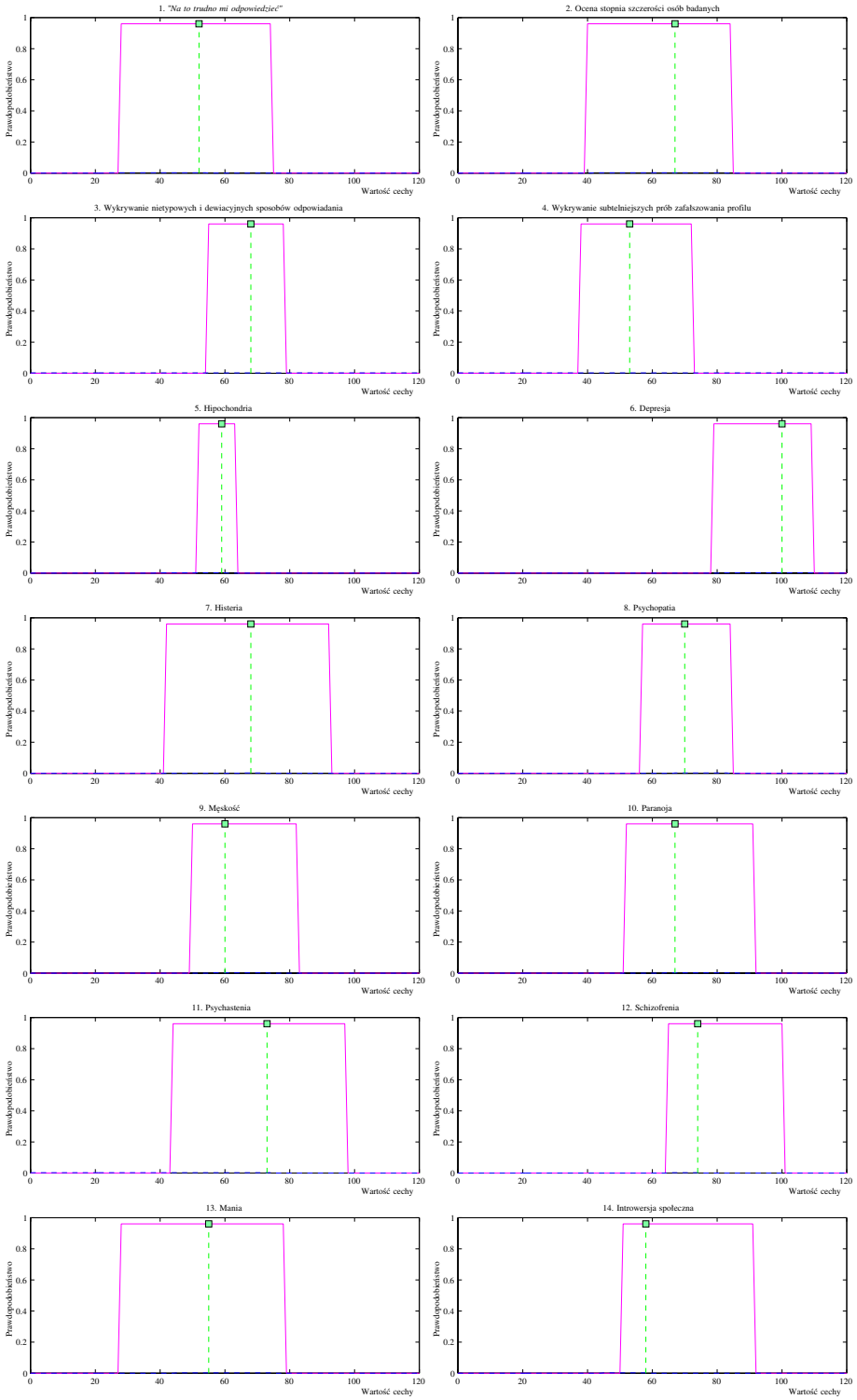
Rysunki 5.21 i 5.22 ilustrują przedziały ufności przypadku, który choć niezupełnie jednoznacznie, ale został zaklasyfikowany do schizofrenii wspólnej (mężczyzn i kobiet), a przez psychologów został on sklasyfikowany jako schizofrenia kobiet. Prawdopodobieństwa tych dwóch klas wyniosły: 0.63 i 0.023. Jednak dziś trudno powiedzieć, gdzie tak naprawdę leży błąd, czy po stronie modelu adaptacyjnego, czy też błąd został popełniony przez psychologa.



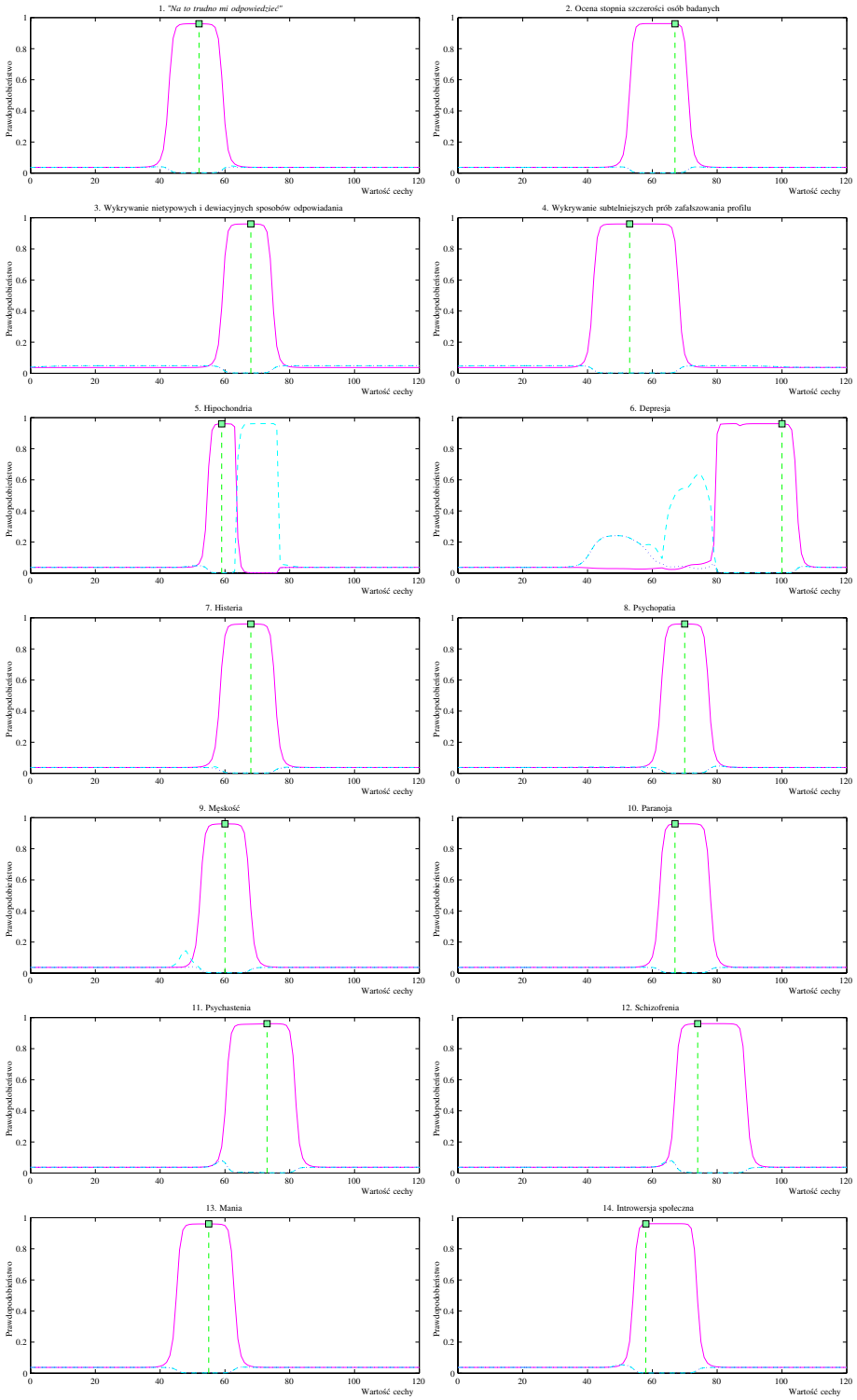
Rysunek 5.17: Przedziały ufności. Przypadek psychozy reaktywnej.



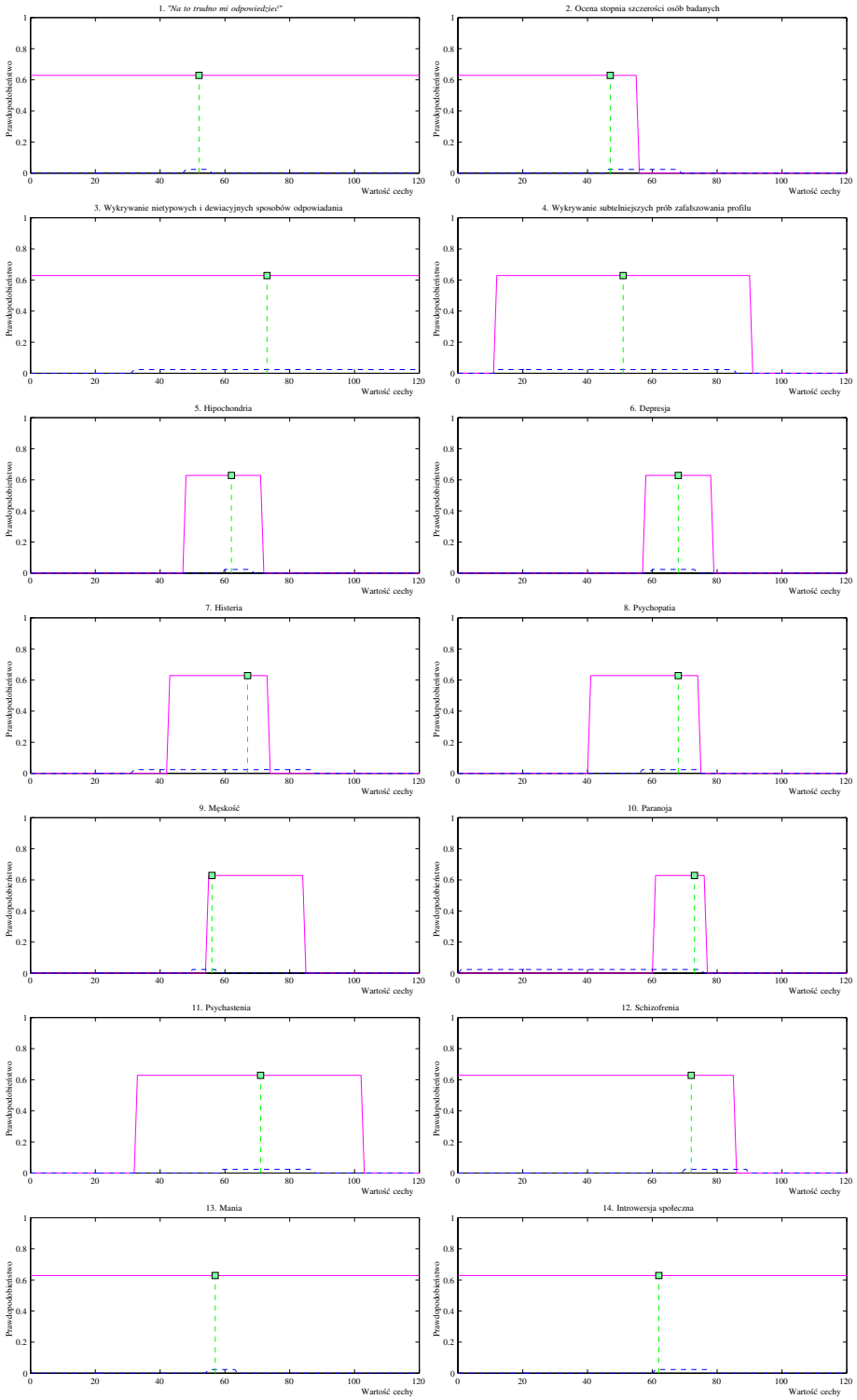
Rysunek 5.18: Probabilistyczne przedziały ufności. Przypadek psychozy reaktywnej.



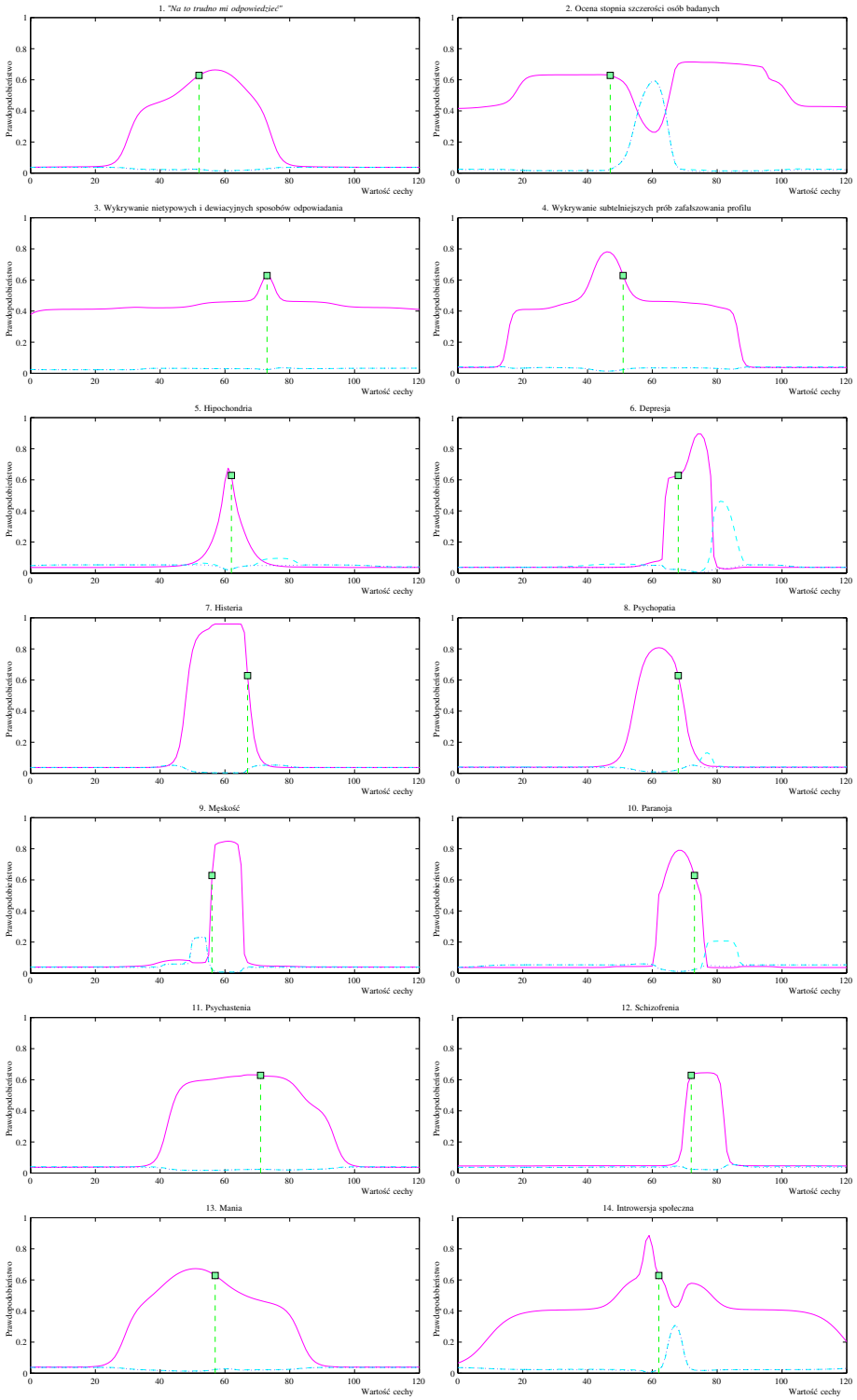
Rysunek 5.19: Przedziały ufności. Zespół urojeniowy.



Rysunek 5.20: Probabilistyczne przedziały ufności. Zespół urojeniowy.



Rysunek 5.21: Przedziały ufności. Przypadek schizofrenii.



Rysunek 5.22: Probabilistyczne przedziały ufności. Przypadek schizofrenii.

5.3.2. Typowe medyczne dane porównawcze

Aby porównać działanie sieci IncNet z innymi znanymi z literatury metodami adaptacyjnymi należało użyć danych, które są ogólnie dostępne i były używane do porównań przeróżnych modeli.

W związku z tym, iż większość ogólnie dostępnych baz danych nie zawiera pełnych opisów źródeł danych, jak i opisu samych danych (opis cech, zakresy i znaczenie skal) przedstawione zostaną wszystkie dostępne informacje.

Zapalenie wyrostka robaczkowego

Dane dotyczące zapalenia wyrostka robaczkowego (*ang. appendicitis*) otrzymano od prof. Shalom Weiss z Rutgers University. Dane składają się ze 106 wektorów. Każdy wektor opisany jest przez 8 cech i może być przypisany do jednej z dwóch klas (80.2% wektorów należy do pierwszej klasy opisującej przypadki ostrego zapalenia, natomiast 19.8% do drugiej klasy opisującej inne problemy).

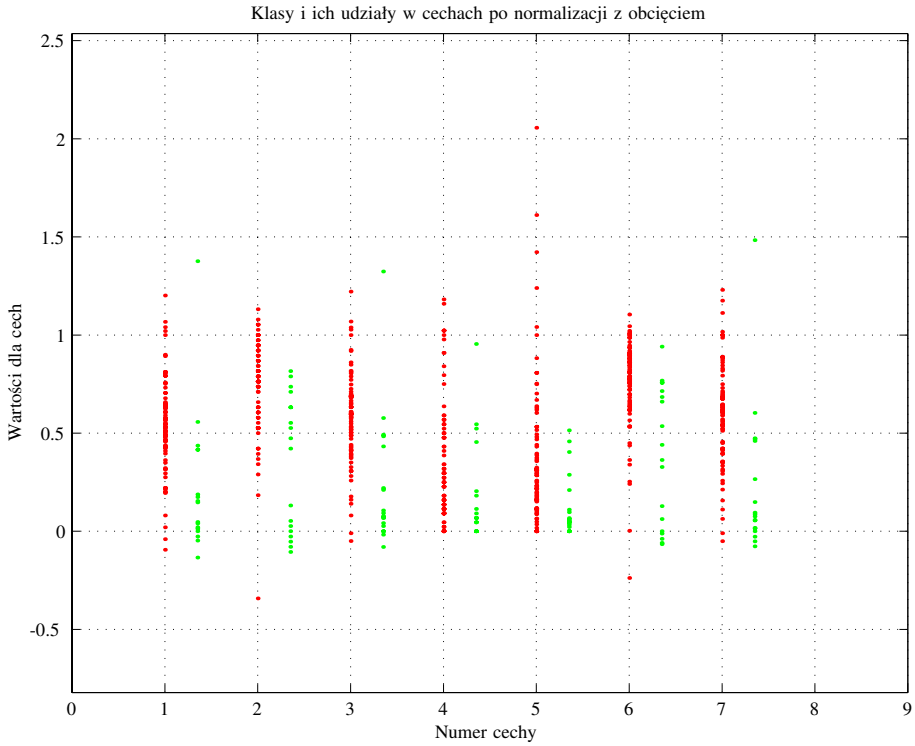
W tabeli 5.5 zostały umieszczone rezultaty uzyskane przy użyciu testu 10 CV. Sieć IncNet uzyskała poprawność 90.11% na zbiorze treningowym i 90.90% na zbiorze testowym dla testu CV 10. Uczenie wynosiło 1100 iteracji. Średnio sieć składała się z 30 neuronów (30 neuronów na dwie podsieci). Do uzyskania takiego samego rezultatu w teście CV 10 można było użyć jednej sieci IncNet, która uczyła się rozpoznawania wektorów klasy pierwszej, wtedy średnia liczba neuronów spada do 20. Dane były normalizowane z 5% obcięciem. Dane można obejrzeć na rysunku 5.23. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

KMK w opisie źródła oznacza, że wynik został otrzymany w naszym zespole przy użyciu ogólnie dostępnej wersji jakiegoś symulatora, bądź symulatorem opracowanym w naszym zespole.

Metoda	Poprawność	Źródło wyniku
IncNet	90.9	KMK
6-NN	88.0	KMK
FSM	84.9	KMK [44]
MLP+BP (Tooldiag)	83.9	KMK
RBF (Tooldiag)	80.2	KMK

Tabela 5.5: Zapalenie wyrostka robaczkowego — porównanie rezultatów dla CV 10.

Natomiast w tabeli 5.6 zostały umieszczone rezultaty otrzymane przy użyciu testu LOO dla różnych metod. Pomimo faktu, że zazwyczaj rezultaty testu LOO są lepsze, to wartość testu CV 10 dla sieci IncNet jest lepsza od każdej wartości testu LOO otrzymanych dla różnych modeli zaprezentowanych w tabeli 5.6.



Rysunek 5.23: Baza danych wyrostka robaczkowego.

Dane dotyczące raka piersi.

Dane dotyczące raka piersi zebrane zostały w szpitalu Uniwersytetu Wisconsin [127, 178] przez Dr. William H. Wolberga i są obecnie dostępne w Instytucie Informacji i Informatyki na Uniwersytecie Kalifornijskim w Irvine [129] (*Wisconsin breast cancer*). Baza składa się z 699 przypadków opisywanych przez 9 cech. Cechy opisują m. in. rozmiar (grupy) guzów, rozmiar komórek i ich kształt, przyleganie, pewne cechy krwi. Wektory pierwszej z dwóch klas stanowią 65.5% wektorów całego zbioru. Pierwsza klasa opisuje nowotwory niezłośliwe, natomiast druga złośliwe.

Sieć IncNet na zbiorze treningowym uzyskała poprawność 97.6% i 97.1% na zbiorze testowym. Średnio końcowa sieć składała się z około 40 neuronów. Uczenie wynosiło 3000 iteracji. Dane zostały znormalizowane. Dane można obejrzeć na rysunku 5.24. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

W tabeli 5.7 zostały umieszczone rezultaty otrzymane przy użyciu testu 10 CV.

Metoda	Poprawność	Źródło wyniku
PVM (reguły logiczne)	89.6	Weiss, Kapouleas [173]
C-MLP2LN (reguły logiczne)	89.6	KMK
k-NN	88.7	KMK
RIAC	86.9	Hamilton m. in. [79]
MLP+BP	85.8	Weiss, Kapouleas [173]
CART, C4.5	84.9	Weiss, Kapouleas [173]
FSM	84.9	KMK [44]

Tabela 5.6: Zapalenie wyrostka robaczkowego — porównanie rezultatów dla testu LOO.

Dane dotyczące zapalenia wątroby.

Baza danych opisująca przypadki zapalenia wątroby (*ang. hepatitis*) pochodzi od G. Gong z Uniwersytetu Carnegie-Mellon i także jest dostępna w UCI [129]. Na dane składa się 155 wektorów, a przestrzeń wejściowa opisywana jest przez 19 cech. Cechy opisują wiek, płeć, steryd, zużycie, samopoczucie, anoreksje, czy wątroba jest duża, twardość, bilirubinę i inne. Każdy z wektorów może przynależeć do jednej z dwóch klas (bardziej liczna klasa zawiera 79.4% wszystkich przypadków), które opisują przeżywalność.

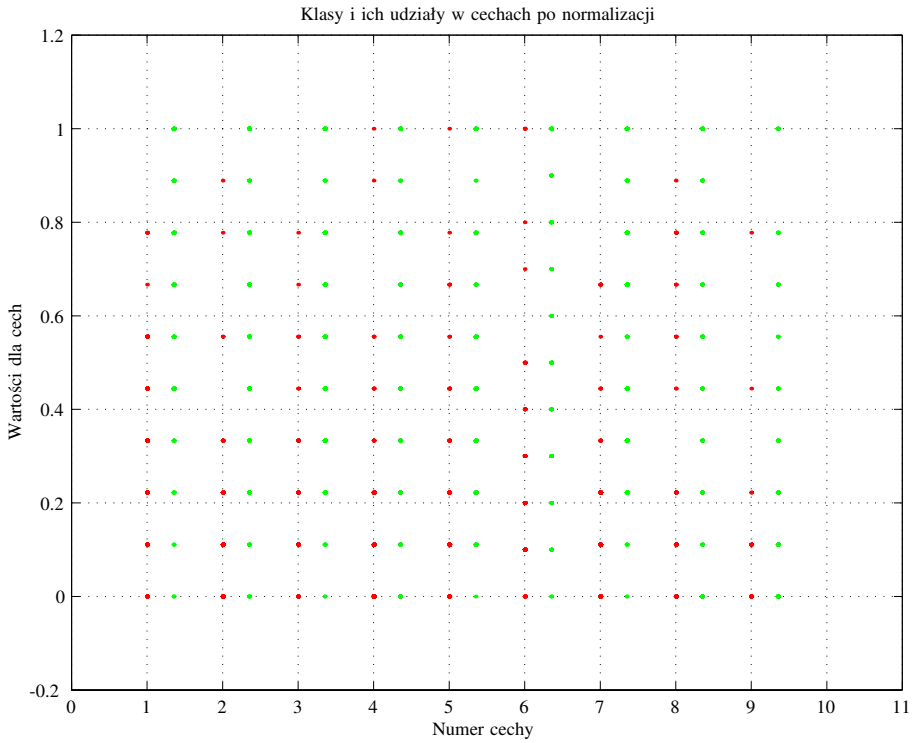
Sieć IncNet uzyskała poprawność 99.36% na zbiorze treningowym i 86% na zbiorze testowym dla testu CV 10. Użyto bicentralnych funkcji transferu z możliwością obrotu. Uczenie wynosiło 2200 iteracji. Średnio sieć składała się z 34 neuronów (34 neurony na dwie podsieci dla każdej klasy). Dane były normalizowane z 5% obcięciem. Dane można obejrzeć na rysunku 5.25. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

W tabeli 5.8 zostały umieszczone rezultaty otrzymane przy użyciu testu 10 CV.

Dane dotyczące cukrzycy

Baza danych chorób cukrzycy pochodzi z National Institute of Diabetes and Digestive and Kidney Diseases [162]. Można ją także znaleźć w UCI [129] (*Pima Indian diabetes*). Na całość zbioru składa się 768 przypadków, w tym 65.1% stanowią przypadki pierwszej klasy. Przynależność do klasy drugiej oznacza iż osoba jest chora na cukrzycę. Każdy przypadek jest opisywany przez 8 cech. Cechy opisują, ile razy pacjentka była w ciąży, test tolerancji glukozy, ciśnienie rozkurczowe, poziom insuliny, masa ciała, czy ktoś w rodzinie był chory na cukrzycę, wiek.

Sieć IncNet uzyskała poprawność 77.2% na zbiorze treningowym i 77.6% na zbiorze testowym dla testu CV 10. Uczenie wynosiło 5000 iteracji. Średnio sieć składała się z 100 neuronów. Dane zostały znormalizowane. Dane można obejrzeć na rysunku 5.26.



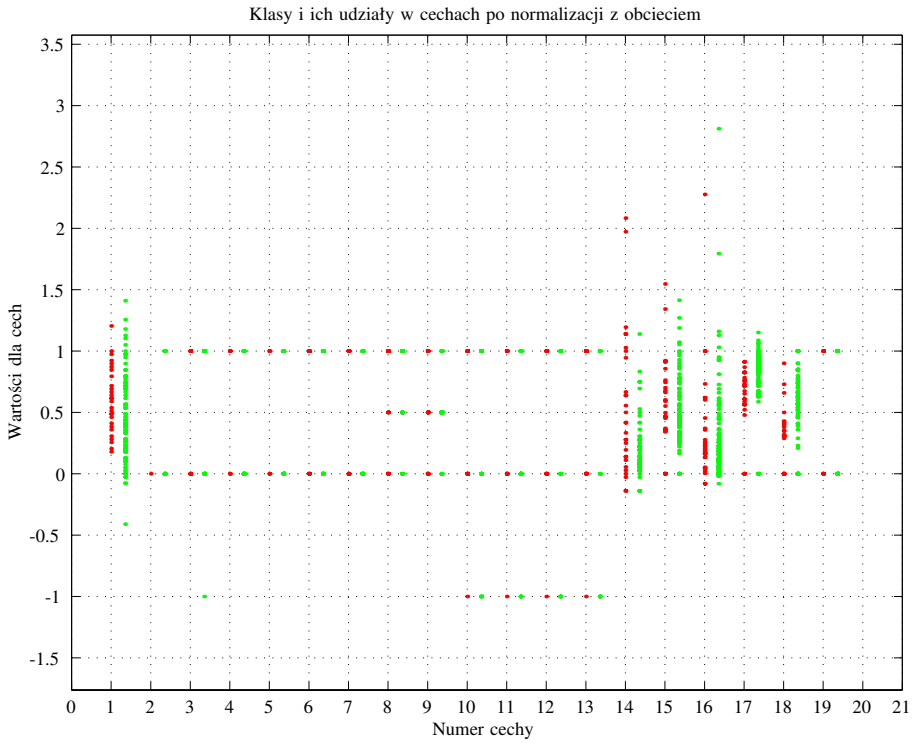
Rysunek 5.24: Baza danych raka piersi.

Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

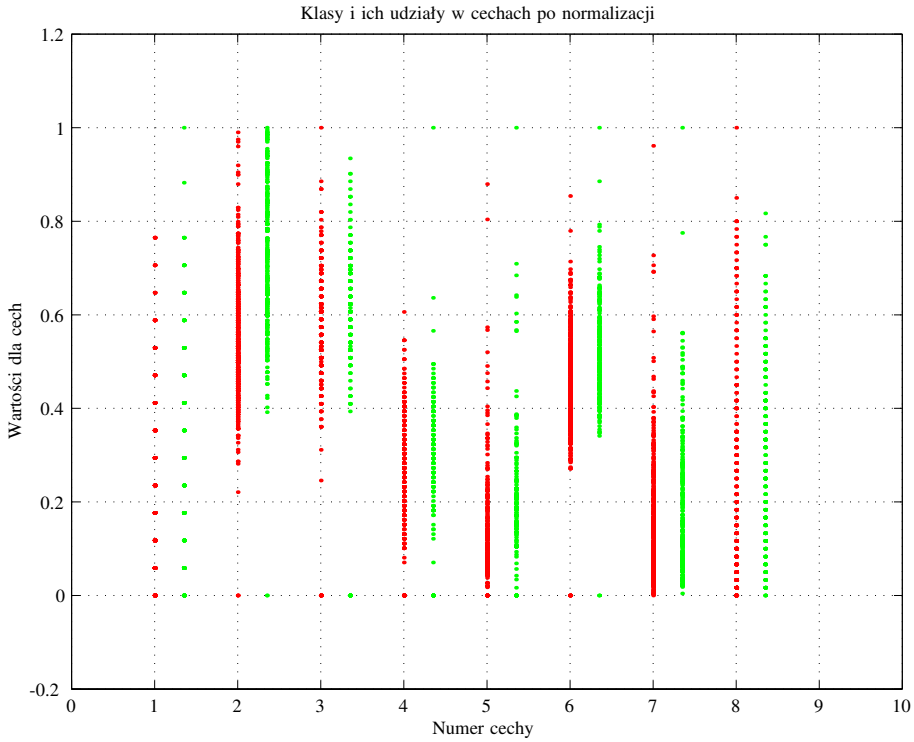
W tabeli 5.9 zostały umieszczono rezultaty uzyskane przy użyciu testu 10 CV.

Choroby tarczycy

Dane dotyczące chorób tarczycy również można znaleźć w UCI [129] (*Thyroid disease*). Dane chorób tarczycy składają się z dwóch zbiorów: treningowego, który składa się z 3772 przypadków (2.47% – pierwsza klasa, 5.06% – druga klasa i 92.47% – trzecia klasa) i testowego na który składa się 3428 przypadków (2.13% – pierwsza klasa, 5.16% – druga klasa i 92.71% – trzecia klasa). Trzy klasy opisują niedoczynność, nadczynność i normalną czynność tarczycy. Dominują przypadki zdrowej tarczycy, gdyż dane pochodzą z dwóch kolejnych lat badań przesiewowych. Przestrzeń wejściowa składa się z 21 cech (15 binarnych i 6 ciągłych). Cechy opisują wiek, płeć, czy pacjent jest leczony tyroksyną, być może pacjent jest leczony tyroksyną, czy pacjent bierze leki przeciwtarczycowe, czy pacjent jest chory (czy źle się czuje), czy pacjentka była w ciąży, czy były wykonywane operacje tarczycy, czy pacjent był leczony jodem J131, test na niedoczynność, test na nadczynność, czy jest stosowany cytrynian litowy, czy występuje wole, czy jest nowotwór, czy występuje niedoczynność



Rysunek 5.25: Baza danych zapalenia wątroby.



Rysunek 5.26: Baza danych cukrzycy.

Metoda	Poprawność	Źródło wyniku
IncNet	97.1	KMK
3-NN, miara Manhattan	97.1	KMK
20-NN, miara euklidesowa	96.9	KMK
Analiza dysk. Fishera	96.8	Ster& Dobnikar [163]
MLP + Wsteczna propagacja	96.7	Ster& Dobnikar [163]
LVQ (*ang. Learning vector quantization*)	96.6	Ster& Dobnikar [163]
KNN	96.6	Ster& Dobnikar [163]
Analiza bayesowska	96.6	Ster& Dobnikar [163]
FSM - (*ang. Feature Space Mapping*)	96.5	KMK [44]
Analiza bayesowska + niezależność cech	96.4	Ster& Dobnikar [163]
DB-CART (drzewa decyzyjne)	96.2	Shang & Breiman [160]
Liniowa analiza dysk.	96.0	Ster& Dobnikar [163]
RBF (Tooldiag)	95.9	KMK
ASI (*ang. Assistant-I*)	95.6	Ster& Dobnikar [163]
ASR (*ang. Assistant-R*)	94.7	Ster& Dobnikar [163]
LFC (*ang. Lookahead feature constructor*)	94.4	Ster& Dobnikar [163]
CART (drzewa decyzyjne)	94.2	Ster& Dobnikar [163]
Kwadratowa analiza dysk.	34.5	Ster& Dobnikar [163]

Tabela 5.7: Dane dotyczące raka piersi — porównanie rezultatów.

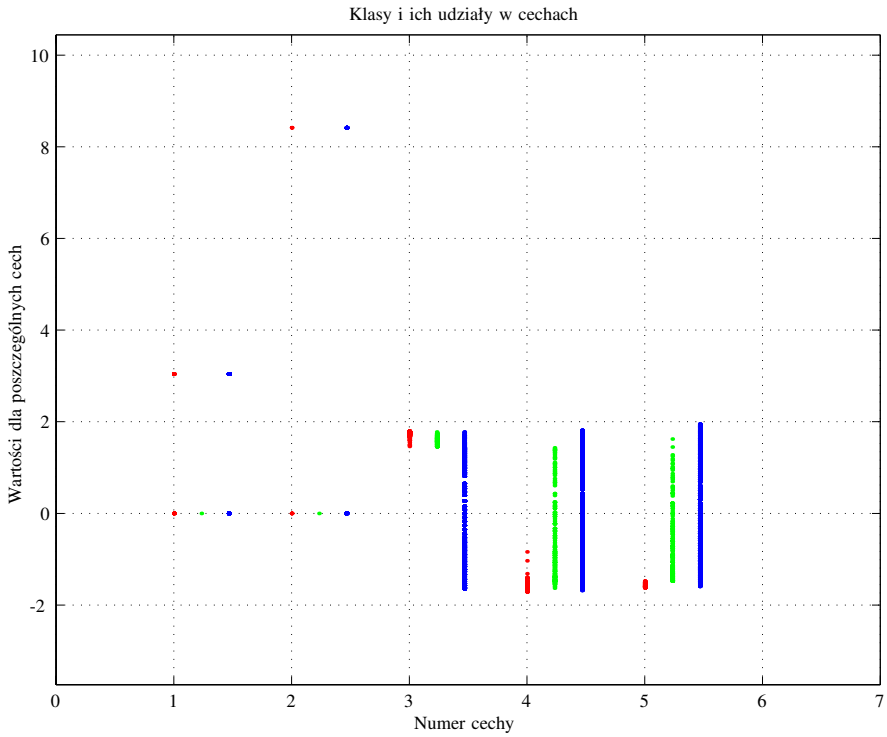
przysadki mózgowej, symptomy psychologiczne, poziom TSH, poziom T3, poziom TT4, poziom T4U, poziom FTI.

W pierwszym etapie dokonano selekcji istotnych cech z pomocą specjalnie do tego celu opracowanej wersji kNN [46]. Istotne okazały się cechy które opisują czy pacjent zażywa tyroksynę, czy przeszedł operacje tarczycy, poziom TSH, poziom TT4, poziom FTI. Po selekcji dane zostały poddane transformacji opisanej przez równanie 5.19, a następnie dane zostały poddane standaryzacji.

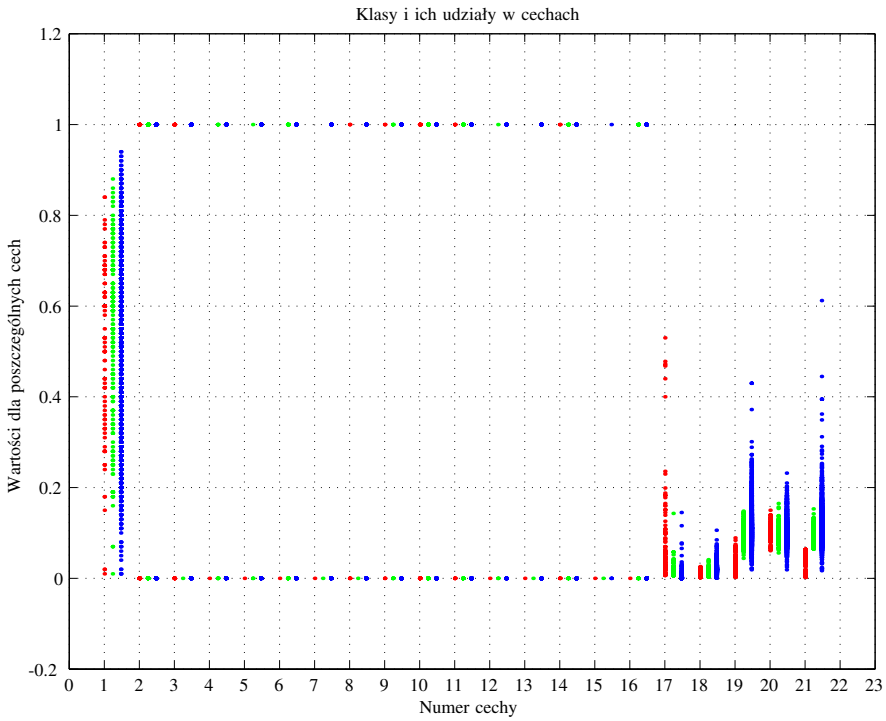
Dane można obejrzeć na rysunku 5.28. Dla każdej z cech różnokolorowe kolumny punktów (horyzontalnie nieco przesunięte względem siebie) odpowiadają różnym klasom.

Sieć IncNet uzyskała poprawność 99.68% na zbiorze treningowym i 99.24% na zbiorze testowym, czyli jedynie 0.12% gorzej niż najlepszy model. Uczenie wynosiło 200000 iteracji. Końcowa sieć składała się z 9 neuronów. Pierwsza podsieć składała się z 2 neuronów, druga z 1 neuronu, a ostatnia z 6 neuronów. Macierze rozrzutu dla danych treningowych i testowych można zobaczyć na rys. 5.29.

W tabeli 5.10 zostały umieszczone rezultaty otrzymane w oparciu o zbiór treningowy i testowy dla różnych modeli.



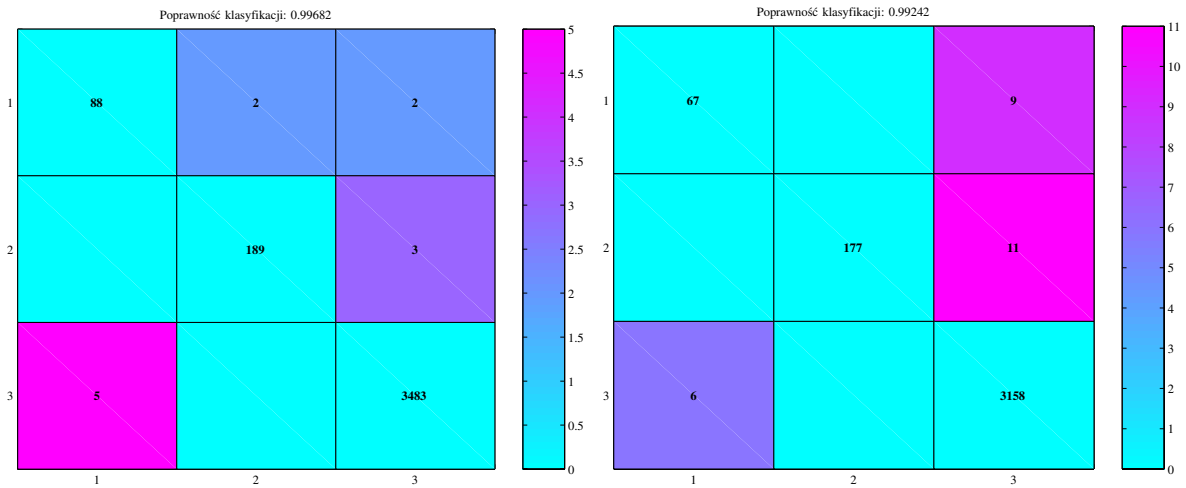
Rysunek 5.27: Baza danych nadczynności i niedoczynności tarczycy po selekcji istotnych cech i transformacji.



Rysunek 5.28: Baza danych nadczynności i niedoczynności tarczycy.

Metoda	Poprawność	Źródło wyniku
18-NN, miara Manhattan	90.2	KMK
FSM z rotacją f. transferu	89.7	KMK [44]
15-NN	89.0	KMK
FSM	88.5	KMK [44]
Liniowa analiza dysk.	86.4	Stern & Dobnikar [163]
Analiza bayesowska + niezależność cech	86.3	Stern & Dobnikar [163]
IncNet Rot	86.0	KMK
Kwadratowa analiza dysk.	85.8	Stern & Dobnikar [163]
1-NN	85.3	Stern & Dobnikar [163]
ASR (*ang. Assistant-R*)	85.0	Stern & Dobnikar [163]
Analiza dysk. Fishera	84.5	Stern & Dobnikar [163]
LVQ (*ang. Learning vector quantization*)	83.2	Stern & Dobnikar [163]
CART (drzewa decyzyjne)	82.7	Stern & Dobnikar [163]
MLP+BP	82.1	Stern & Dobnikar [163]
ASI (*ang. Assistant-I*)	82.0	Stern & Dobnikar [163]
LFC (*ang. Lookahead feature constructor*)	81.9	Stern & Dobnikar [163]
RBF (Tooldiag)	79.0	KMK
MLP+BP (Tooldiag)	77.4	KMK

Tabela 5.8: Zapalenie wątroby — porównanie rezultatów.



Rysunek 5.29: Macierze rozrzutu dla bazy danych chorób tarczycy. Po lewej dla zbioru treningowego, po prawej dla zbioru testowego.

Metoda	Poprawność	Źródło wyniku
Logdisc	77.7	Statlog [132]
IncNet	77.6	KMK
DIPOL92	77.6	Statlog [132]
Liniowa analiza dysk.	77.5	Statlog [132], Stern & Dobnikar [163]
SMART	76.8	Statlog [132]
ASI (*ang. Assistant-I*)	76.6	Stern & Dobnikar [163]
Liniowa analiza Fishera	76.5	Stern & Dobnikar [163]
MLP+BP	76.4	Stern & Dobnikar [163]
MLP+BP	75.2	Statlog [132]
LVQ (*ang. Learning vector quantization*)	75.8	Stern & Dobnikar [163]
RBF	75.7	Statlog [132]
LFC (*ang. Lookahead feature constructor*)	75.8	Stern & Dobnikar [163]
Analiza bayesowska + niezależność cech	75.5	Stern & Dobnikar [163]; Statlog
Analiza bayesowska	75.4	Stern & Dobnikar [163]
CART	74.5	Stalog [132]
DB-CART	74.4	Shang & Breiman [160]
ASR (*ang. Assistant-R*)	74.3	Stern & Dobnikar [163]
CART	72.8	Stern & Dobnikar [163]
C 4.5	73.0	Statlog [132]
Kohonen	72.7	Statlog [132]
kNN	71.9	Stern & Dobnikar [163]
kNN	67.6	Statlog [132]
Kwadratowa analiza dysk.	59.5	Ster& Dobnikar [163]

Tabela 5.9: Choroby cukrzycy — porównanie rezultatów.

Metoda	**Poprawność**		**Źródło wyniku**
	treningowa	testowa	
C-MLP2LN rules + ASA	99.9	99.36	KMK
CART	99.8	99.36	Weiss [173]
PVM	99.8	99.33	Weiss [173]
IncNet	99.68	99.24	KMK
MLP init+ a,b opt.	99.5	99.1	KMK
C-MLP2LN rules	99.7	99.0	KMK
Cascade correlation	100.0	98.5	Schiffmann [157]
Local adapt. rates	99.6	98.5	Schiffmann [157]
BP+genetic opt.	99.4	98.4	Schiffmann [157]
Quickprop	99.6	98.3	Schiffmann [157]
RPROP	99.6	98.0	Schiffmann [157]
3-NN, Euclides, 3 features used	98.7	97.9	KMK
1-NN, Euclides, 3 features used	98.4	97.7	KMK
Best backpropagation	99.1	97.6	Schiffmann [157]
1-NN, Euclides, 8 features used	–	97.3	KMK
Bayesian classif.	97.0	96.1	Weiss [173]
BP+conj. gradient	94.6	93.8	Schiffmann [157]
1-NN Manhattan, std data	100	93.8	KMK
default: 250 test errors		92.7	
1-NN Manhattan, raw data	100	92.2	KMK

Tabela 5.10: Choroby tarczycy — porównanie rezultatów.

5.4. Aproksymacja

Innymi bardzo dobrymi przykładami ilustrującymi efektywność sieci IncNet, są problemy aproksymacji funkcji. W poniższym podrozdziale zostaną przedstawione aproksymacje kilku, najczęściej spotykanych w literaturze funkcji.

5.4.1. Funkcja Hermita

Pierwszą prezentowaną funkcją jest funkcja Hermita, zdefiniowana przez:

$$f_{her}(x) = 1.1(1 - x + 2x^2)\exp(-1/2x^2) \tag{5.21}$$

Na dane treningowe składa się 40 losowych punktów z przedziału $[-4, 4]$, natomiast zbiór testowy składa się ze 100 losowych wartości, pochodzących z tego samego przedziału. Wszystkie sieci były uczone przez 800 iteracji (tj. prezentacji jednego wektora treningowego).

Tablica 5.11 prezentuje rezultaty uzyskane dla sieci IncNet z funkcjami bicentralnymi (IncNet), jak i z funkcjami Gaussa (IncNet G). Przedstawione są także rezultaty dla sieci RAN-EKF [104, 107] i RAN [146] uzyskanymi przez Kadirkamanathana. Do porównania użyto miary błędu MSE (5.9).

Błąd MSE			
IncNet	IncNet G	RAN-EKF	RAN
0.000225	0.0029	0.0081	0.0225

Tabela 5.11: Aproksymacja funkcji Hermita (5.21).

5.4.2. Funkcja Gabora i Girosiego

Problemem aproksymacji obu poniżej zdefiniowanych funkcji, zajmował się Girosi, Jones i Poggio [75]. Funkcja Gabora jest zdefiniowana przez:

$$f_{gab}(x, y) = e^{-||\mathbf{x}||^2}\cos(.75\pi(x + y)) \tag{5.22}$$

a druga (Girosiego) przez:

$$f_{gir}(x, y) = \sin(2\pi x) + 4(y - 0.5)^2 \tag{5.23}$$

Proces uczenia jest bardzo trudny, ponieważ opiera się jedynie o 20 wektorów treningowych, losowo wybranych z przedziału $[-1, 1] \times [-1, 1]$ dla funkcji Gabora i z przedziału $[0, 1] \times [0, 1]$ dla funkcji Girosiego. Natomiast na zbiór testowy składa się 10,000 losowych punktów z tych samych przedziałów.

Model1	$\sum_{i=1}^{20} c_i [e^{-\left(\frac{(x-x_i)^2}{\sigma_1}+\frac{(y-y_i)^2}{\sigma_2}\right)} + e^{-\left(\frac{(x-x_i)^2}{\sigma_2}+\frac{(y-y_i)^2}{\sigma_1}\right)}]$	$\sigma_1 = \sigma_2 = 0.5$
Model2	$\sum_{i=1}^{20} c_i [e^{-\left(\frac{(x-x_i)^2}{\sigma_1}+\frac{(y-y_i)^2}{\sigma_2}\right)} + e^{-\left(\frac{(x-x_i)^2}{\sigma_2}+\frac{(y-y_i)^2}{\sigma_1}\right)}]$	$\sigma_1 = 10, \sigma_2 = 0.5$
Model3	$\sum_{i=1}^{20} c_i [e^{\frac{(x-x_i)^2}{\sigma}} + e^{-\frac{(y-y_i)^2}{\sigma}}]$	$\sigma = 0.5$
Model4	$\sum_{\alpha=1}^{7} b_\alpha e^{-\frac{(x-t_x^\alpha)^2}{\sigma}} + \sum_{\beta=1}^{7} c_\beta e^{-\frac{(y-t_y^\beta)^2}{\sigma}}$	$\sigma = 0.5$
Model5	$\sum_{\alpha=1}^{n} c_\alpha e^{-(\boldsymbol{W}_\alpha \cdot \boldsymbol{X} - t_\alpha)^2}$	
Model6	$\sum_{i=1}^{20} c_i [\sigma(x - x_i) + \sigma(y - y_i)]$	
Model7	$\sum_{\alpha=1}^{7} b_\alpha \sigma(x - t_x^\alpha) + \sum_{\beta=1}^{7} c_\beta \sigma(y - t_y^\beta)$	
Model8	$\sum_{\alpha=1}^{n} c_\alpha \sigma(\boldsymbol{W}_\alpha \cdot \boldsymbol{X} - t_\alpha)$	

Tabela 5.12: Definicje modeli użytych do aproksymacji funkcji Gabora i Girosiego.

Funkcja Girosiego — Błąd MSE treningowy/testowy									
IncNet Rot	**IncNet**	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
.00000133	.00000232	.000036	.000067	.000001	.000001	.000170	.000001	.000003	.000743
0.000859	.000082	.011717	.001598	.000007	.000009	.001422	.000015	.000020	.026699

Funkcja Gabora — Błąd MSE treningowy/testowy									
IncNet Rot	**IncNet**	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8
.000006	.000025	.000000	.000000	.000000	.345423	.000001	.000000	.456822	.000044
0.015316	0.025113	.003818	.344881	67.9523	1.22211	.033964	98.4198	1.39739	.191055

Tabela 5.13: Aproksymacja funkcji Gabora (5.22) i Girosiego (5.23).

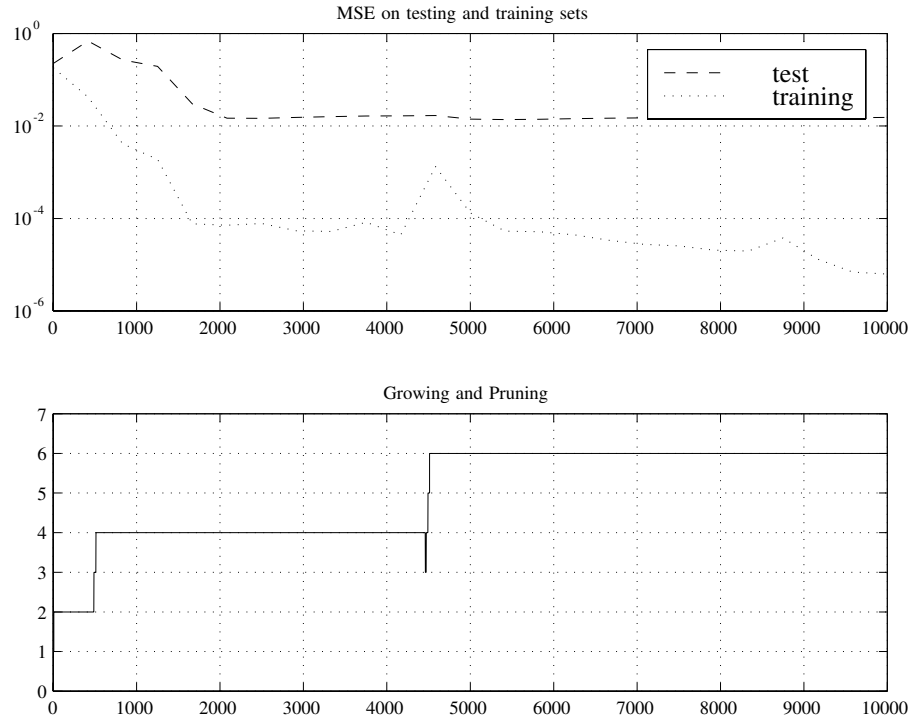
Tablica 5.12 opisuje modele testowane przez Girosiego i in. [75]. Natomiast tabela 5.13 opisuje rezultaty, uzyskane przy użyciu tych modeli i przy użyciu sieci IncNet z bicentralnymi funkcjami transferu (IncNet), jak i funkcjami bicentralnymi z rotacją (IncNet Rot). Do porównania użyto miary błędu MSE (5.9).

Choć sieć IncNet nie zawsze jest najlepsza, to jednak po uśrednieniu rezultatów, wysuwa się na czoło hierarchii, dzięki swej elastyczności (patrz tablica 5.13). Uzyskane rezultaty pokazują, że funkcje bicentralne z rotacją również mogą być efektywnie wykorzystane przez sieć IncNet, pomimo wprowadzenia $N-1$ dodatkowych parametrów adaptacyjnych.

Dla funkcji Gabora (5.22) końcowa sieć IncNet składała się z 4 neuronów dla bicentralnych funkcji transferu (zmieniając parametry uczenia tak, aby końcowa liczba neuronów była większa, uzyskiwało się gorszy poziom generalizacji) i 6 neuronów dla funkcji bicentralnych z rotacją.

Dla funkcji Girosiego (5.23) sieć IncNet składała się z 8 neuronów dla funkcji bicentralnych i bicentralnych z rotacją.

Przykładowy proces uczenia można prześledzić na rysunku 5.30. Jak widać, na



Rysunek 5.30: Adaptacja sieci IncNet dla problemu aproksymacji funkcji Sugeno. Błąd MSE dla zbioru treningowego i testowego (u góry). Liczba neuronów (u dołu).

początku sieć dodaje i usuwa neurony, aż do ustalenia ostatecznej architektury.

5.4.3. Funkcja Sugeno

Do kolejnego testu zostanie użyta funkcja zaproponowana przez Sugeno [164], zdefiniowana przez:

$$f(x, y, z) = (1 + x^{0.5} + y^{-1} + z^{-1.5})^2 \tag{5.24}$$

Poniżej zostaną zaprezentowane rezultaty uzyskane dla sieci IncNet z funkcjami bicentralnymi i bicentralnymi z rotacją. Zostaną one porównane z rezultatami uzyskanymi przez Sugeno [164], Kosińskiego i in. [115] i Horikawy i in. [88].

Dane treningowe stanowi 216 punktów o wartościach losowych z przedziału $[1, 6]$. Dane testowe składają się ze 125 punktów, również o wartościach losowych, z przedziału $[1.5, 5.5]$. Wszystkie testy były wykonywane przy podobnych parametrach początkowych. Do porównań użyto miary błędu APE zdefiniowanej wzorem 5.11.

Model	Błąd APE	
	treningowy	testowy
GMDS model Kongo [115]	4.7	5.7
Fuzzy model 1 Sugeno [164]	1.5	2.1
Fuzzy model 2 Sugeno [164]	0.59	3.4
FNN Type 1 Horikawa [88]	0.84	1.22
FNN Type 2 Horikawa [88]	0.73	1.28
FNN Type 3 Horikawa [88]	0.63	1.25
M - Delta model [115]	0.72	0.74
Fuzzy INET [115]	0.18	0.24
Fuzzy VINET [115]	0.076	0.18
IncNet	**0.119**	**0.122**
IncNet Rot	**0.053**	**0.061**

Tabela 5.14: Porównanie rezultatów aproksymacji funkcji Sugeno (5.24).

Końcowa sieć IncNet składała się z 11 neuronów w warstwie ukrytej. Rezultaty zostały zaprezentowane w tabeli 5.14.

Rozdział 6

Konkluzje

W pracy przedstawiono wiele nowych rozwiązań na różnych poziomach modelowania systemów adaptacyjnych.

Zebrane zostały i omówione informacje o funkcjach transferu, metodach adaptacji sieci RBF i przeróżnych metodach ontogenicznych.

Dokonano możliwie spójnego opisu wielu funkcji transferu, stosowanych przez różne sztuczne sieci neuronowe, jak i takich, które jeszcze nie były używane przez jakiekolwiek modele. Wskazano na wiele różnic, zalet i wad poszczególnych funkcji. Przedstawiono bardzo ciekawą taksonomię funkcji aktywacji i funkcji wyjścia, co upraszcza analizę różnych własności tych funkcji, a tym samym upraszcza dokonanie świadomego i dobrego wyboru funkcji transferu do danej sieci neuronowej.

Zaproponowano całą rodzinę nowych, bardziej uniwersalnych funkcji transferu, które okazały się znacząco bardziej efektywne, przy jednoczesnym niedużym wzroście liczby parametrów adaptacyjnych. Funkcje te mogą być stosowane w różnych modelach sieci neuronowych.

Zebrano, omówiono i porównano wiele metod inicjalizacji (proponując pewne nowe rozwiązania) i uczenia sieci neuronowych z radialnymi funkcjami bazowymi.

Zaproponowano obliczeniowo tańszą metodę rozszerzonego filtru Kalmana, dzięki której można rozwiązywać znacznie trudniejsze problemy, nie zmniejszając znacząco możliwości generalizacji powstających sieci. Szybką, jak i normalną wersję filtru EKF zastosowano do adaptacji parametrów sieci IncNet o architekturze zbliżonej do sieci z radialnymi funkcjami bazowymi.

Opracowano nowe metody kontroli złożoności sieci neuronowych, które w przeciwieństwie do wszystkich innych umożliwiają rozbudowę i zmniejszanie architektury sieci podczas procesu uczenia. Badania dowiodły, że metody te są skuteczne. Użycie sieci IncNet dla problemów wieloklasowych pokazało, że niemal do każdej z klas

optymalna architektura sieci składa się z różnej liczby neuronów.

Głównym celem było zbudowanie modelu, który w sekwencyjnym procesie uczenia starałby się utrzymywać złożoność modelu adekwatnie do złożoności danych uczących. Dlatego też sieć IncNet zbudowano przy użyciu elastycznych funkcji bicentralnych, które są wykorzystywane jako funkcje transferu warstwy ukrytej architektury zbliżonej do sieci RBF; wykorzystano filtr Kalmana jako efektywny algorytm uczenia; zastosowano metody kontroli złożoności sieci, które kontrolują złożoność sieci na bieżąco podczas procesu uczenia.

Do celów klasyfikacyjnych zaproponowano używanie klastra podsieci z modułem decyzyjnym. Taka sieć umożliwia niezależną estymację rozkładów poszczególnych klas, a następnie wykorzystywanie tych informacji w module decyzyjnym, który dokonuje klasyfikacji. Z tak skonstruowanego modelu można wyznaczyć prawdopodobieństwa przynależności danego wektora do poszczególnych cech, co wzbogaca możliwości diagnostyki.

Przedstawiono także nowe narzędzie — probabilistyczne przedziały ufności, które są silną alternatywą dla reguł logicznych. Umożliwiają dokładne porównanie wpływów istotnych klas dla danego przypadku. Stanowią także bardzo dobrą metodę wizualizacji.

Część teoretyczna poparta została wieloma ciekawymi aplikacjami sieci IncNet dla klasyfikacji problemów medycznych i aproksymacji funkcji. Przedstawiono liczne porównania z najlepszymi modelami dla wielu baz danych. Przeprowadzono dużo różnych analiz powstałych modeli.

Spis literatury

[1] R. Adamczak, W. Duch, N. Jankowski. New developments in the feature space mapping model. *Third Conference on Neural Networks and Their Applications*, s. 65–70, Kule, Poland, 1997.

[2] J. Allison. Multiquadratic radial basis functions for representing multidimensional high energy physics data. *Computer Physics Communications*, 77:377–395, 1993.

[3] H. Almuallim, T. G. Dietterich. Efficient algorithms for identifying relevant features. *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, 1992.

[4] J. A. Anderson. Logistic discrimination. *Handbook of statistics 2: Classification, pattern recognition and reduction of dimensionality*, s. 169–191. North Holland, Amsterdam, 1982.

[5] J. A. Anderson. *An Introduction to Neural Networks*. Bradford Book, 1995.

[6] A. R. Barron. Universal approximation bounds for superpositions of a sigmoid function. *IEEE Transactions on Information Theory*, 39:930–945, 1993.

[7] Y. S. Ben-Porath, N. Sherwood. *The MMPI-2 content component scales: Development, psychometric characteristics, and clinical applications*. University of Minnesota Press, Minneapolis, 1993.

[8] C. M. Bishop. Improving the generalization properties of radial basis function neural networks. *Neural Computation*, 3(4):579–588, 1991.

[9] C. M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, 1991.

[10] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[11] C. M. Bishop, M. Svensén, C. K. I. Williams. Em optimization of latent-variable density models. D. S. Touretzky, M. C. Mozer, M. E. Hasselmo, redaktorzy, *Advances in Neural Information Processing Systems 8*, Cambridge, MA, 1996. MIT Press.

[12] L. Bobrowski. Piecewise–linear classifiers, formal neurons and separability of learning sets. *Proceedings of ICPR*, s. 224–228, 1996.

[13] L. Bobrowski. Generowanie sieci neuropodobnych oraz drzew decyzyjnych w oparciu o kryterium dipolowe. *Symulacja w badaniach i rozwoju*, Jelenia Góra, 1997.

[14] L. Bobrowski, M. Krętowska, M. Krętowski. Design of neural classifying networks by using dipolar criterions. *Third Conference on Neural Networks and Their Applications*, Kule, Poland, 1997.

[15] L. Bottou, V. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[16] L. Breiman. Bias-variance, regularization, instability and stabilization. C. M. Bishop, redaktor, *Neural Networks and Machine Learning*, s. 27–56. Springer-Verlag, 1998.

[17] L. Breiman, J. H. Friedman, A. Olshen, C. J. Stone. *Classification and regression trees*. Wadsworth, 1984.

[18] J. S. Bridle. Probabilistic interpretation of feedforward classification network outputs with relationships to statistical pattern recognition. F. Fogelman Soulié, J. Hérault, redaktorzy, *Neurocomputing: Algorithms, Architectures and Applications*, s. 227–236. Springer-Verlag, New York, 1990.

[19] D. S. Broomhead, D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[20] I. Bruha. From machine learning to knowledge discovery: Preprocessing and postprocessing. *Machine Learning and Applications. Workshop on Preprocessing and Postprocessing on machine learning and data mining: theoretical aspects and applications*, s. 1–17, Greece, 1999.

[21] J. M. Buhman, N. Tishby. A statistical learning theory of data clustering. C. M. Bishop, redaktor, *Neural Networks and Machine Learning*, s. 57–68. Springer-Verlag, 1998.

[22] J. N. Butcher, J. R. Graham, C. L. Williams, Y. Ben-Porath. *Development and use for the MMPI-2 Content Scales*. University of Minnesota Press, Minneapolis, University of Minnesota Press.

[23] J. N. Butcher, C. L. Williams. *Essential of MMPI-2 and MMPI-A interpretation*. University of Minnesota Press, Minneapolis, 1992.

[24] J. N. Buther, W. G. Dahlstrom, J. R. Graham, A. Tellegen, B. Kaem-mer. *Minnesota Multiphasic Personality Inventory-2 (MMPI-2): Manual for administration and scoring*. University of Minnesota Press, Minneapolis, 1989.

[25] J. V. Candy. *Signal processing: The model based approach.* McGraw-Hill, New York, 1986.

[26] Mu-Song Chen. *Analyses and Design of Multi-Layer Perceptron Using Polynomial Basis Functions*. Praca doktorska, The University of Texas at Arlington, 1991.

[27] S. Chen, S. A. Billings, W. Luo. Orthogonal least squares methods and their application to non-linear system identification. *International Journal of Control*, 50(5):1873–1896, 1989.

[28] S. Chen, C. F. N. Cowan, P. M. Grant. Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, 2(2):302–309, 1991.

[29] V. Cherkassky, F. Mulier. *Learning from data*. Adaptive and learning systems for signal processing, communications and control. John Wiley & Sons, Inc., 1998.

[30] M. Cottrell, B. Girard, Y. Girard, M. Mangeas, C. Muller. Neural modeling for time series: a statistical stepwise method for weight elimination. *IEEE Transaction on Neural Networks*, 6(6):1355–1364, 1995.

[31] Y. Le Cun, B. Boser, J. Denker, D. Henderson, W. Hubbart, L. Jackel. Handwritten digit recognition with a back-propagation network. D. S. Touretzky, redaktor, *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kauffman.

[32] Y. Le Cun, J. Denker, S. Solla. Optimal brain damage. D. S. Touretzky, redaktor, *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kauffman.

[33] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[34] M. Dash, H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3), 1997.

[35] T. Denoeoux, R. Lengellé. Initializing back propagation networks with prototypes. *Neural Networks*, 6(3):351–363, 1993.

[36] G. Dorffner. A unified framework for MLPs and RBFNs: Introducing conic section function networks. *Cybernetics and Systems*, 25(4):511–554, 1994.

[37] W. Duch, R. Adamczak, G. H. F. Diercksen. Distance-based multilayer perceptrons. *International Conference on Computational Intelligence for Modelling Control and Automation*, s. 75–80, Vienna, Austria, 1999.

[38] W. Duch, R. Adamczak, G. H. F. Diercksen. Neural networks in non-euclidean spaces. *Neural Processing Letters*, 10:1–10, 1999.

[39] W. Duch, R. Adamczak, K. Grąbczewski. Extraction of crisp logical rules using constrained backpropagation networks. *International Conference on Artificial Neural Networks (ICANN'97)*, s. 2384–2389, 1997.

[40] W. Duch, R. Adamczak, K. Grąbczewski. Extraction of logical rules from backpropagation networks. *Neural Processing Letters*, 7:1–9, 1998.

[41] W. Duch, R. Adamczak, K. Grąbczewski. Methodology of extraction, optimization and application of logical rules. *Intelligent Information Systems VIII*, s. 22–31, Ustroń, Poland, 1999.

[42] W. Duch, R. Adamczak, N. Jankowski. Initialization of adaptive parameters in density networks. *Third Conference on Neural Networks and Their Applications*, s. 99–104, Kule, Poland, 1997.

[43] W. Duch, R. Adamczak, N. Jankowski. Initialization and optimization of multilayered perceptrons. *Third Conference on Neural Networks and Their Applications*, s. 105–110, Kule, Poland, 1997.

[44] W. Duch, R. Adamczak, N. Jankowski. New developments in the feature space mapping model. Raport instytutowy CIL-KMK-2/97, Computational Intelligence Lab, DCM NCU, Toruń, Poland, 1997. (long version).

[45] W. Duch, G. H. F. Diercksen. Feature space mapping as a universal adaptive system. *Computer Physics Communications*, 87:341–371, 1994.

[46] W. Duch, K. Grudziński. The weighted k-NN with selection of features and its neural realization. *4th Conference on Neural Networks and Their Applications*, s. 191–196, Zakopane, Poland, 1999.

[47] W. Duch, N. Jankowski. New neural transfer functions. *Journal of Applied Mathematics and Computer Science*, 7(3):639–658, 1997.

[48] W. Duch, N. Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 1999. (accepted), (PDF).

[49] W. Duch, N. Jankowski, A. Naud, R. Adamczak. Feature space mapping: a neurofuzzy network for system identification. *Proceedings of the European Symposium on Artificial Neural Networks*, s. 221–224, Helsinki, 1995.

[50] W. Duch, T. Kucharski, J. Gomuła, R. Adamczak. *Metody uczenia maszynowego w analizie danych psychometrycznych. Zastosowanie do wielowymiarowego kwestionariusza osobowości MMPI–WISKAD.* Toruń, Poland, 1999.

[51] R. O. Duda, P. E. Hart. *Patter Classification and Scene Analysis*. Wiley, 1973.

[52] N. Dyn. Interpolation and approximation by radial and related functions. C. K. Chiu, L. L. Schumaker, J. D. Watts, redaktorzy, *Approximation Theory VI*. Academic Press, San Diego, 1989.

[53] S. E. Fahlman. The recurrent cascade-correlation architecture. Raport instytutowy, Carnegie Mellon University, School of Computer Science, 1991.

[54] S. E. Fahlman, C. Lebiere. The cascade-correlation learning architecture. D. S. Touretzky, redaktor, *Advances in Neural Information Processing Systems 2*, s. 524–532. Morgan Kaufmann, 1990.

[55] S. E. Fahlman, C. Lebiere. The cascade-correlation learning architecture. Raport instytutowy CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[56] M. Fernández, C. Hernández. How to select the inputs for a multilayer feedforward by using the training set. *5th International Work Conference on Artificial an Natural Neural Networks*, s. 477–486, Alicante. Spain, 1999.

[57] E. Fiesler. Comparative bibliography of ontogenic neural networks. *Proceedings of the International Conference on Artificial Neural Networks*, 1994.

[58] W. Finnoff, F. Hergert, H. G. Zimmermann. Improving model detection by nonconvergent methods. *Neural Networks*, 6(6):771–783, 1993.

[59] W. Finnoff, H. G. Zimmermann. Detecting structure in small datasets by network fitting under complexity constrains. *Proceedings of the second annual workshop on computational learning theory and natural learning systems*, Berkeley, CA, 1991.

[60] R. Franke. Scattered data interpolation: test of some methods. *Math Computation*, 38:181–200, 1982.

[61] M. Frean. The upstart algorithm: a method for constructing and training feedforward neural networks. *Neural Computation*, 2(2):198–209, 1990.

[62] N. de Freitas, M. Milo, P. Clarkson, M. Niranjan, A. Gee. Sequential support vector machines. 1999.

[63] N. de Freitas, M. Niranjan, A. Gee. Hierarchical bayesian-kalman models for regularisation and ard in sequential learning. Raport instytutowy CUED/F-INFENG/TR 307, Cambridge University Engineering Department, 1998.

[64] J. H. Friedman. Multivariate adaptive regression splines (with discussion). *Ann. Stat.*, 19:1–141, 1991.

[65] J. H. Friedman. Flexible metric nearest neighbor classification. Raport instytutowy, Department of Statistics, Stanford University, 1994.

[66] B. Fritzke. Fast learning with incremental RBF networks. *Neural Processing Letters*, 1(1):2–5, 1994.

[67] B. Fritzke. Supervised learning with growing cell structures. J. D. Cowan, G. Tesauro, J. Alspector, redaktorzy, *Advances in Neural Information Processing Systems 6*, San Mateo, CA, 1994. Morgan Kaufman.

[68] B. Fritzke. A growing neural gas network lerns topologies. G. Tesauro, D. S. Touretzky, T. K. Leen, redaktorzy, *Advances in Neural Information Processing Systems 7*, Cambridge, MA, 1995. MIT Press.

[69] B. Fritzke. A self-organizing network that can follow non-stationary distributions. W. Gerstner, A. Germond, M. Hasler, J. Nicoud, redaktorzy, *7th International Conference on Artificial Neural Networks*, s. 613–618, Lausanne, Switzerland, 1997. Springer-Verlag.

[70] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, San Diego, 1972.

[71] S. Fusi, M. Mattia. Collective behavior of networks with linear (VLSI) integrate and fire neurons. *Neural Computation*, 1997.

[72] S. I. Gallant. Three constructive algorithms for network learning. *Proceedings of the eigth annual conference of the cognitive science society*, s. 652–660, Hillsdale, NJ, 1986. Lawrence Erlbaum.

[73] B. G. Giraud, A. Lapedes, L. C. Liu, J. C. Lemm. Lorentzian neural nets. *Neural Networks*, 8(5):757–767, 1995.

[74] F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, 1998.

[75] F. Girosi, M. Jones, T. Poggio. Priors stabilizers and basis functions: From regularization to radial, tensor and additive splines. Raport instytutowy, MIT, Cambridge, Massachusetts, 1993.

[76] F. Girosi, T. Poggio. Networks and the best approximation property. AI Lab. Memo, MIT, 1989.

[77] G. H. Golub, M. Heath, G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–213, 1979.

[78] K. Grąbczewski, W. Duch. A general purpose separability criterion for classification systems. *4th Conference on Neural Networks and Their Applications*, s. 203–208, Zakopane, Poland, 1999.

[79] H. J. Hamilton, N. Shan, N. Cercone. Riac: a rule induction algorithm based on approximate classification. Raport instytutowy CS 96-06, Regina University, 1996.

[80] E. Hartman, J. D. Keeler. Predicting the future: Advantages of semilocal units. *Neural Computation*, 3(4):566–578, 1991.

[81] E. J. Hartman, J. D. Keeler, J. M. Kowalski. Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2(2):210–215, 1990.

[82] B. Hassibi, D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in Neural Information Processing Systems 5*. Morgan Kaufmann, 1993.

[83] B. Hassibi, D. G. Stork, G. J. Wolff. Optimal brain surgeon and general network pruning. Raport instytutowy CRC-TR-9235, RICOH California Research Center, Menlo Park, CA, 1992.

[84] T. Hastie, R. Tibshirani. Discriminant adaptive nearest neighbor classification. *IEEE PAMI 18*, s. 607–616, 1996.

[85] S. Haykin. *Neural Networks - A Comprehensive Foundation.* Maxwell MacMillian Int., New York, 1994.

[86] S. Haykin. *Adaptive filter theory*. Printice-hall international, 1996.

[87] G. E. Hinton. Learning translation invariant recognition in massively parallel networks. J. W. de Bakker, A. J. Nijman, P. C. Treleaven, redaktorzy, *Proceedings PARLE Conference on Parallel Architectures and Languages Europe*, s. 1–13, Berlin, 1987. Springer-Verlag.

[88] S. Horikawa, Takeshi Furuhashi, Yoshiki Uchikawa. On fuzzy modeling using fuzzy neural networks with the back-propagation algorithm. *IEEE Transactions on Neural Networks*, 3(5):801–806, 1992.

[89] K. Hornik, M. Stinchcombe, H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[90] K. Hornik, M. Stinchcombe, H. White, P. Auer. Degree of approximation results for feedforward networks approximating unknown mappings and their derivatives. *Neural Computation*, 6(6):1262–1275, 1994.

[91] C. C. Hsu, D. Gubovic, M. E. Zaghloul, H. H. Szu. Chaotic neuron models and their VLSI implementations. *IEEE Transactions on Neural Networks*, 7(6):1339–1350, 1996.

[92] J. M. Hutchinson. *A Radial Basis Function Approach to Financial Time Series Analysis*. Praca doktorska, MIT, Cambridge, MA, 1993.

[93] J. M. Hutchinson, A. W. Lo, T. Poggio. A nonparametric approach to pricing and hedging derivative securities via learning networks. *Journal of Finance*, 49(3):851–889, 1994.

[94] Institute of Parallel and Distributed High-Performance Systems (IPVR). *Stuttgart Neural Networks Simulator*. http://www.informatik.uni-stuttgart.de/ipvr-/bv/projekte/snns/snns.html.

[95] K. Jajuga. *Statystyczna analiza wielowymiarowa*. Wydawnictwo Naukowe PWN, Warszawa, 1993.

[96] N. Jankowski. Controlling the structure of neural networks that grow and shrink. *Second International Conference on Cognitive and Neural Systems*, Boston, USA, 1998.

[97] N. Jankowski. Approximation and classification in medicine with IncNet neural networks. *Machine Learning and Applications. Workshop on Machine Learning in Medical Applications*, s. 53–58, Greece, 1999. (PDF).

[98] N. Jankowski. Approximation with RBF-type neural networks using flexible local and semi-local transfer functions. *4th Conference on Neural Networks and Their Applications*, s. 77–82, 1999. (PDF).

[99] N. Jankowski. Flexible transfer functions with ontogenic neural. Raport instytutowy, Computational Intelligence Lab, DCM NCU, Toruń, Poland, 1999. (PDF).

[100] N. Jankowski, V. Kadirkamanathan. Statistical control of RBF-like networks for classification. *7th International Conference on Artificial Neural Networks*, s. 385–390, Lausanne, Switzerland, 1997. Springer-Verlag.

[101] N. Jankowski, V. Kadirkamanathan. Statistical control of growing and pruning in RBF-like neural networks. *Third Conference on Neural Networks and Their Applications*, s. 663–670, Kule, Poland, 1997.

[102] M. Jordan. Why the logistic function? A tutorial discussion on probabilities and neural networks. Raport instytutowy 9503, Computational Cognitive Science, MIT, Cambridge, MA, 1995.

[103] V. Kadirkamanathan. *Sequential learning in artificial neural networks*. Praca doktorska, Cambridge University Engineering Department, 1991.

[104] V. Kadirkamanathan. A statistical inference based growth criterion for the RBF networks. *Proceedings of the IEEE. Workshop on Neural Networks for Signal Processing*, 1994.

[105] V. Kadirkamanathan, M. Niranjan. Nonlinear adaptive filtering in nonstationary environments. *Proceedings of the international conference on acoustic, speech and signal processing*, Toronto, 1991.

[106] V. Kadirkamanathan, M. Niranjan. Application of an architecturally dynamic network for speech pattern classification. *Proceedings of the Institute of Acoustics*, 14:343–350, 1992.

[107] V. Kadirkamanathan, M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5(6):954–975, 1993.

[108] V. Kadirkamanathan, M. Niranjan, F. Fallside. Sequential adaptation of radial basis function neural networks and its application to time-series prediction. D. S. Touretzky, redaktor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, 1990.

[109] T. Kasahara, M. Nakagawa. A study of association model with periodic chaos neurons. *Journal of Physical Society*, 64:4964–4977, 1995.

[110] M. J. Kirby, R. Miranda. Circular nodes in neural networks. *Neural Computations*, 8(2):390–402, 1996.

[111] K. Kobayasji. On the capacity of neuron with a non-monotone output function. *Network*, 2:237–243, 1991.

[112] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[113] T. Kohonen. *Self-organizing maps*. Springer-Verlag, Heidelberg Berlin, 1995.

[114] J. Korbicz, A. Obuchowicz, D. Uciński. *Sztuczne sieci neuronowe, podstawy i zastosowania*. Akademicka Oficyna Wydawnicza PLJ, Warszawa, 1994.

[115] W. Kosiński, M. Weigl. Mapping neural networks and fuzzy inference systems for approximation of multivariate function. E. Kącki, redaktor, *System Modeling Control, Artificial Neural Networks and Their Applications*, wolumen 3, s. 60–65, Łódź, Poland, 1995.

[116] B. Kosko. *Neural Networks and Fuzzy Systems*. Prentice Hall International, 1992.

[117] P. R. Krishnaiah, L. N. Kanal. *Handbook of statistics 2: Classification, pattern recognition and reduction of dimensionality*. North Holland, Amsterdam, 1982.

[118] V. Kurkova. Approximation of functions by perceptron networks with bounded number of hidden units. *Neural Networks*, 8(5):745–750, 1995.

[119] V. Kurkova, P. C. Kainen, V. Kreinovich. Estimates of the number of hidden units and variation with respect to half-spaces. *Neural Networks*, 10(6):1061–1068, 1997.

[120] H. Leung, S. Haykin. Rational neural networks. *Neural Computation*, 5(6):928–938, 1993.

[121] W. A. Light. Some aspects of radial basis function approximation. S. P. Singh, redaktor, *Approximation theory, spline functions and applications*, wolumen 256, s. 163–190. Kluwer Academic Publishers, Boston, MA, 1992.

[122] D. Lowe. Adaptive radial basis function nonlinearities, and the problem of generalization. *1st IEE International Conference on Artificial Neural Networks*, s. 171–175, London, UK, 1989.

[123] D. Lowe. On the iterative inversion of RBF networks: A statistical interpretation. *2nd IEE International Conference on Artificial Neural Networks*, s. 29–33, London, UK, 1991.

[124] D. Lowe. Novel "topographic" nonlinear. *3rd IEE International Conference on Artificial Neural Networks*, s. 29–33, London, UK, 1993.

[125] D. Lowe. Radial basis function networks. M. A. Arbib, redaktor, *The handbook of brain theory and neural networks*. MIT Press, Cambridge, MA, 1995.

[126] W. Maas. Lower bounds for the computational power of networks of spiking neurons. *Neural Computations*, 8:1–40, 1996.

[127] O. L. Mangasarian, W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18, 1990.

[128] W. S. McCulloch, W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[129] C. J. Merz, P. M. Murphy. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[130] M. Mezard, J. P. Nadal. Learning in feedforward layered networks: the tiling algorithm. *Journal of physics*, 22:2191–2203, 1989.

[131] R. Michalski. Concept learning and natural induction. *Machine Learning and Applications*, Chania, Greece, 1999.

[132] D. Michie, D. J. Spiegelhalter, C. C. Taylor. *Machine learning, neural and statistical classification*. Elis Horwood, London, 1994.

[133] T. Miki, M. Shimono, T. Yamakawa. A chaos hardware unit employing the peak point modulation. *Proceedings of the International Symposium on Nonlinear Theory and its Applications*, s. 25–30, Las Vegas, Nevada, 1995.

[134] T. Mitchell. *Machine learning*. McGraw Hill, 1997.

[135] J. Moody, C. J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[136] M. Morita. Associative memory with nonmonotone dynamics. *Neural Networks*, 6:115–126, 1993.

[137] M. T. Musavi, R. J. Bryant, M. Qiao, M. T. Davisson, E. C. Akeson, B. D. French. Mouse chromosome classification by radial basis function networks with fast orthogonal search. *Neural Networks*, 11(4):769–777, 1998.

[138] M. Nakagawa. An artificial neuron model with a periodic activation function. *Journal of Physical Society*, 64:1023–1031, 1995.

[139] N.J. Nillson. *Learning machines: Foundations of trainable pattern classifying systems*. McGraw-Hill, New York, 1965.

[140] M Niranjan, F. Fallside. Neural networks and radial basis functions in classifying static speech patterns. *Computer speech and language*, 4:275–289, 1990.

[141] P. Niyogi, F. Girosi. On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions. *Neural Computation*, 8(4):819–842, 1996.

[142] S. J. Nowlan, G. E. Hinton. Simplifying neural networks by soft weight sharing. *Neural Computation*, 4(4):473–493, 1992.

[143] M. Orr. Introduction to radial basis function networks. Raport instytutowy, Centre for Cognitive Science, University of Edinburgh, 1996.

[144] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley, Reading, MA, 1989.

[145] J. Park, I. W. Sandberg. Universal approximation using radial basis function networks. *Neural Computation*, 3(2):246–257, 1991.

[146] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3:213–225, 1991.

[147] T. Poggio. Learning sparse representations for vision. *Second International Conference on Cognitive and Neural Systems*, Boston, USA, 1998.

[148] T. Poggio, S. Edelman. A network that learns to recognize three three-dimentional objects. *Nature*, 343:263–266, 1990.

[149] T. Poggio, F. Girosi. A theory of networks for approximation and learning. Raport instytutowy A. I. Memo 1140, MIT, Massachusetts, 1989.

[150] T. Poggio, F. Girosi. Network for approximation and learning. *Proceedings of the IEEE*, 78:1481–1497, 1990.

[151] M. J. D. Powell. Radial basis functions for multivariable interpolation: A review. J. C. Mason, M. G. Cox, redaktorzy, *Algorithms for Approximation of Functions and Data*, s. 143–167, Oxford, 1987. Oxford University Press.

[152] S. Qian, Y. C. Lee, R. D. Jones, C. W. Barnes, K. Lee. Function approximation with an orthogonal basis net. *Proceedings of the IJCNN*, wolumen 3, s. 605–619, 1990.

[153] S. Ridella, S. Rovetta, R. Zunino. Circular backpropagation networks for classification. *IEEE Transaction on Neural Networks*, 8(1):84–97, 1997.

[154] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, 1996.

[155] L. Rutkowski. *Filtry adaptacyjne i adaptacyjne przetwarzanie sygnałów*. Wydawnictwa Naukowo–Techniczne, Warszawa, 1994.

[156] A. Saranli, B. Baykal. Complexity reduction in radial basis function (RBF) networks by using radial b-spline functions. *Neurocomputing*, 18(1-3):183–194, 1998.

[157] W. Schiffman, M. Joost, R. Werner. Comparison of optimized backpropagation algorithms. *Proceedings of ESANN'93*, s. 97–104, Brussels, 1993.

[158] B. Schölkopf, C. Burges, A. Smola. *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[159] N. N. Schraudolph. A fast, compact approximation of the exponential function. Raport instytutowy, IDSIA, Lugano, Switzerland, 1998.

[160] N. Shang, L. Breiman. Distributioon based trees are more accurate. *Proceedings of ICONIP'96*, 1996.

[161] S. Singhal, L. Wu. Training feed-forward networks with the extended kalman filter. *IEEE International Conference on Acoustic, Speech and Signal Processing*, s. 1187–1190, Glasgow, UK, 1989.

[162] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler nad R. S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. *Proceedings of the Symposium on Computer Applications and Medical Care*, s. 261–265. IEEE Computer Society Press, 1988.

[163] B. Šter, A. Dobnikar. Neural networks in medical diagnosis: Comparison with other methods. *Proceedings of the International Conference EANN '96*, s. 427–430, 1996.

[164] M. Sugeno, G. T. Kang. Structure identification of fuzzy model. *Fuzzy Sets and Systems*, 28, 1988.

[165] J. A. Sutherland. Holographic model of memory, learning and expression. *International Journal of Neural Systems*, 1:256–267, 1990.

[166] R. Tadeusiewicz, A. Izworski, J. Majewski. *Biometria*. Wydawnictwa AGH, Krak w, 1993.

[167] Ryszard Tadeusiewicz. *W stronę uśmiechniętych maszyn*. Wydawnictwa ALFA, Warszawa, 1989.

[168] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.

[169] V. Vapnik. The support vector method of function estimation. C. M. Bishop, redaktor, *Neural Networks and Machine Learning*, s. 239–268. Springer-Verlag, 1998.

[170] A. S. Weigend, D. E. Rumelhart, B. A. Huberman. Back–propagation, weight elimination and time series prediction. *Proceedings of the 1990 Connectionist Models Summer School*, s. 65–80. Morgan Kaufmann, 1990.

[171] A. S. Weigend, D. E. Rumelhart, B. A. Huberman. Generalization by weight elimination with application to forecasting. R. P. Lipmann, J. E. Moody, D. S. Touretzky, redaktorzy, *Advances in Neural Information Processing Systems 3*, s. 875–882, San Mateo, CA, 1991. Morgan Kaufmann.

[172] A. Weigned, H. Zimmermann, Ralph Neuneier. Clearing. P. Refenes, Y. Abu-Mostafa, J. Moody, A. Weigend, redaktorzy, *Proceedings of NNCM, Neural Networks in Financial Engineering*, Singapore, 1995. World Scientific.

[173] S.M. Weiss, I. Kapouleas. An empirical comparison of pattern recognition, neural nets and machine learning classification methods. J.W. Shavlik, T.G. Dietterich, redaktorzy, *Readings in Machine Learning*. Morgan Kauffman, 1990.

[174] D. Wettschereck, T. Dietterich. Improving the performance of radial basis function networks by learning center locations. J. E. Moody, S. J. Hanson, R. P. Lipmann, redaktorzy, *Advances in Neural Information Processing Systems 4*, s. 1133–1140, San Mateo, CA, 1992. Morgan Kaufman.

[175] D. R. Wilson, T. R. Martinez. Heterogeneous radial basis function networks. *Proceedings of the International Conference on Neural Networks*, wolumen 2, s. 1263–1276, 1996.

[176] D. R. Wilson, T. R. Martinez. Value difference metrics for continuously valued attributes. *Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks*, s. 11–14, 1996.

[177] D. R. Wilson, T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[178] William H. Wolberg, O.L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences*, wolumen 87, s. 9193–9196, U.S.A., 1990.

[179] T. Yamakawa, M. Shimono, T. Miki. Design criteria for robust associative memory employing non-equilibrium network. *Proceedings of the 4th International Conference on Soft Computing (IIZUKA'96)*, s. 688–691, Iizuka, 1996.

[180] H. F. Yanai, S. Amari. A theory of neural net with non-monotone neurons. *Proceedings of IEEE International Conference on Neural Networks*, s. 1385–1390, 1993.

[181] H. F. Yanai, S. Amari. Auto-associative memory with two-stage dynamics of non-monotonic neurons. *IEEE Transactions on Neural Networks*, 1998. (submitted).

[182] H. F. Yanai, Y. Sawada. Associative memory network composed of neurons with histeretic property. *Neural Networks*, 3:223–228, 1990.

[183] H. F. Yanai, Y. Sawada. Integrator neurons for analogue neural networks. *IEEE Transactions on Circuits and Systems*, 37:854–856, 1990.

[184] S. Yoshizawa, M. Morita, S. Amari. Capacity of associative memory using a nonmonotonic neuron model. *Neural Networks*, 6:167–176, 1993.