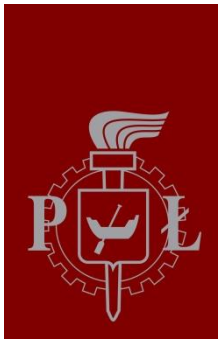


Politechnika Łódzka

Wydział Elektrotechniki, Elektroniki, Informatyki i Automatyki

Instytut Informatyki Stosowanej

Praca inżynierska

Projektowanie sztucznych sieci neuronowych z wykorzystaniem kwantowo inspirowanych algorytmów genetycznych

Szymon Grzelak

Promotor: dr hab. inż. Jacek Kucharski, prof. PŁ

Łódź, 2020

PODZIĘKOWANIA

Serdeczne podziękowania za trud włożony w moje wychowanie i edukację oraz największą dbałość o mnie w każdym innym aspekcie życia pragnę złożyć mojej Rodzinie. Dziękuję szczególnie **moim Rodzicom** za bezcenną umiejętność krytycznego formułowania myśli, którą nieustannie staram się w sobie pielęgnować, oraz żywy przykład prawdziwie partnerskiej relacji, dającej szczęście i siłę wszystkim obdarowanym płynącą z niej miłością.

Pragnę również podziękować promotorowi mojej pracy **dr hab. inż. Jackowi Kucharskiemu**, **prof. PŁ**, za poświęcony mi czas, wyrozumiałość i zaufanie a przede wszystkim za przekazaną wiedzę i nieocenione uwagi podczas tworzenia pracy. Serdecznie dziękuję wszystkim moim nauczycielom i wykładowcom, w tym kadrze Wydziału Elektrotechniki, Elektroniki, Informatyki i Automatyki, których ciężka praca przygotowała mnie do czynnego uczestnictwa we wspólnym rozszerzaniu posiadanej wiedzy.

„Jesteśmy karłami, którzy wspięli się na ramiona olbrzymów. W ten sposób widzimy więcej i dalej niż oni, ale nie dlatego, ażeby wzrok nasz był bystrzejszy lub wzrost słuszniejszy, ale dlatego, iż to oni dźwigają nas w górę i podnoszą o całą gigantyczną wysokość.”

Bernard z Chartes

STRESZCZENIE

Tytuł – Projektowanie sztucznych sieci neuronowych z wykorzystaniem kwantowo inspirowanych algorytmów genetycznych.

Streszczenie – Prezentowana praca dotyczy wykorzystania kwantowo inspirowanych algorytmów genetycznych do projektowania sztucznych sieci neuronowych, czerpiąc inspiracje zarówno z budowy i ewolucji organizmów żywych, jak i teoretycznych podstaw Informatyki Kwantowej.

W pracy dokonano przeglądu wiedzy na temat modeli sztucznych neuronów, jednokierunkowych sieci wielowarstwowych oraz zagadnień ich projektowania i uczenia. Przedstawiono klasyczne algorytmy uczenia pierwszego i drugiego rzędu, tj. algorytm wstecznej propagacji błędów oraz algorytm Levenberga-Marquardta. Następnie omówiono realizację kwantowo inspirowanych algorytmów genetycznych rzędu II oraz zaadaptowano ją na potrzeby ewolucji sztucznych sieci neuronowych. Ponadto opisano implementację wymienionych wyżej narzędzi w języku C# oraz sposób ich połączenia w celu rozwiązania realnych problemów projektowania sztucznych sieci neuronowych.

Słowa kluczowe – kwantowo inspirowane algorytmy genetyczne, neuroewolucja, optymalizacja nieliniowa, sztuczne sieci neuronowe, wsteczna propagacja.

ABSTRACT

Title – Artificial neural networks design using quantum-inspired genetic algorithms.

Abstract – The presented thesis concerns the use of quantum-inspired genetic algorithms to design artificial neural networks, drawing inspiration both from the structure and evolution of living organisms, and theoretical foundations of Quantum Computing.

The paper reviews the knowledge about artificial neuron models, feed-forward multilayer neural networks and issues connected with their design and training. Classic first- and second-order training algorithms, i.e. Error Backpropagation Training and Levenberg-Marquardt Training, are presented. Then, the idea of Quantum-Inspired Second-Order Genetic Algorithms is discussed and adapted for the needs of the artificial neural networks evolution. In addition, the implementation of the above-mentioned tools in C# was described, as well as how to combine them to solve real problems of designing artificial neural networks.

Keywords – artificial neural networks, backpropagation, neuroevolution, nonlinear optimization, Quantum-Inspired Genetic Algorithms.

SPIS TREŚCI

WYKAZ WAŻNIEJSZYCH SKRÓTÓW I OZNACZEŃ

Przyjęte oznaczenia algorytmów oraz symboli matematycznych, stosowane w pracy, zostały zaprezentowane odpowiednio w Tabeli 1 oraz 2.

Tabela 1 Oznaczenie algorytmów

Skrót	Znaczenie
EBP	Algorytm wstecznej propagacji błędów (ang. *Error Backpropagation Training)* (Werbos, 1974)
LM	Algorytm Levenberga-Marquardta (ang. *Levenberg-Marquardt Training*) (Hagan, Menhaj, 1994)
QIEA	Kwantowo inspirowany algorytm ewolucyjny (ang. *Quantum-Inspired Evolutionary Algorithm*)
QIGA	Kwantowo inspirowany algorytm genetyczny (ang. *Quantum-Inspired Genetic Algorithm*)
QIGA1	Pierwotny kwantowo inspirowany algorytm genetyczny (Han, Kim, 2000)
QIGA2	Kwantowo inspirowany algorytm genetyczny rzędu II (Nowotniak, Kucharski, 2014)
iQIEA	Algorytm QIEA z „kodowaniem impulsowym” (da Cruz et al., 2007)

Tabela 2 Oznaczenie symboli

Symbol	Znaczenie
$\mathbb{R}$	zbiór liczb rzeczywistych
$\boldsymbol{x} \in \mathbb{R}^n$	n-elementowy wektor sygnałów wejściowych sztucznego neuronu
$\boldsymbol{w} \in \mathbb{R}^n$	n-elementowy wektor wag synaptycznych
$\boldsymbol{w}^* \in \mathbb{R}^n$	skorygowany n-elementowy wektor wag synaptycznych
net: $\mathbb{R}^n$ x $\mathbb{R}^n \rightarrow \mathbb{R}$	suma ważona sygnałów wejściowych
f: $\mathbb{R} \rightarrow \mathbb{R}$	funkcja aktywacji
$y \in \mathbb{R}$	wartość wyjściowa sztucznego neuronu
$\eta \in \mathbb{R}$	współczynnik uczenia
$d \in \mathbb{R}$	wartość wzorcowa

Tabela 2 Oznaczenie symboli cd.

$\beta \in \mathbb{R}$	współczynnik funkcji sigmoidalnej
p	wzór uczący
P	ciąg wzorów uczących, również populacja klasyczna
$Q \in \mathbb{R}$	miara błędu
N	sztuczny neuron
$\boldsymbol{u} \in \mathbb{R}^n$	n-elementowy wektor sygnałów wejściowych sieci neuronowej
$\boldsymbol{o} \in \mathbb{R}^n$	n-elementowy wektor sygnałów wyjściowych sieci neuronowej
$\varepsilon \in \mathbb{R}$	błąd sieci spropagowany na wyjścia neuronu
$\delta \in \mathbb{R}$	błąd sieci spropagowany na wejścia neuronu
$\boldsymbol{J} \in \mathbb{R}^{n \times m}$	macierz Jacobiego
$\boldsymbol{e} \in \mathbb{R}^n$	n-elementowy wektor błędów
$\boldsymbol{Q} \in \mathbb{R}^{n \times n}$	pseudohesjan
$\boldsymbol{g} \in \mathbb{R}^n$	n-elementowy wektor gradientu
$\mu \in \mathbb{R}$	parametr w algorytmie Levenberga-Marquardta
$f \in \mathbb{R}$	modyfikator parametru μ w algorytmie Levenberga-Marquardta
$\hat{\varepsilon} \in \mathbb{R}$	błąd jednostkowy sieci spropagowany na wyjścia neuronu
$\hat{\delta} \in \mathbb{R}$	błąd jednostkowy sieci spropagowany na wejścia neuronu
$\boldsymbol{j} \in \mathbb{R}^n$	wiersz macierzy Jacobiego
$\boldsymbol{q} \in \mathbb{R}^{n \times m}$	podmacierz pseudohesjanu $\boldsymbol{Q}$
$\boldsymbol{\eta} \in \mathbb{R}^n$	podwektor wektora gradientu $\boldsymbol{g}$
$h \in \mathbb{R}$	wartość przesunięcia w metodzie różnic skończonych
$\alpha, \beta \in [0, 1]$	amplitudy prawdopodobieństwa binarnego genu kwantowego
$\boldsymbol{q} \in \mathbb{R}^n$	chromosom kwantowy
$\mu \in [0, 1]$	współczynnik kontrakcji amplitud prawdopodobieństwa
$b \in \{0,1\}^n$	najlepszy znaleziony osobnik klasyczny
Q	populacja kwantowa
$C \in \{0,1\}^{n \times n}$	macierz połączeń
$\boldsymbol{H} \in \mathbb{R}^{n \times n}$	macierz Hessego (hesjan)

WSTĘP

Sztuczna Inteligencja i Mechanika Kwantowa, jako dwie niezależne dziedziny informatyki i fizyki będące znaczącymi osiągnięciami nauki XX wieku, nieodzownie przyczyniły się do zgłębienia tajemnic Wszechświata, nadając nowe tempo rozwojowi cywilizacji. Zaprezentowane w niniejszej pracy rozwiązanie inżynierskie wykorzystuje narzędzie **kwantowo inspirowanych algorytmów genetycznych**, usytuowane na pograniczu tych dwóch dziedzin wiedzy, do **projektowania sztucznych sieci neuronowych** dla problemów aproksymacji, klasyfikacji czy sterowania. Takie podejście pozwala na poprawę efektywności metod Sztucznej Inteligencji dzięki wprowadzeniu do nich elementów inspirowanych teorią Mechaniki Kwantowej.

Mechanika Kwantowa (ang. *Quantum Mechanics* – QM), stanowiąc jedno z najbardziej przełomowych osiągnięć współczesnej fizyki, dostarcza najdokładniejszego znanego obecnie opisu fizycznej rzeczywistości w skali mikroświata (promieniowanie ciała doskonale czarnego (Planck, 1901), efekt fotoelektryczny zewnętrzny (Einstein, 1905), zjawisko Comptona (Compton, 1923)), rzucając nowe światło nie tylko na podstawowe oddziaływania fizyczne (teoria wielkiej unifikacji (Buras et al., 1978), koncepcja pola Higgsa (Higgs, 1964), Model Standardowy (Novaes, 2000)) ale również pojmowanie natury Wszechświata (Hawking, 2007). Po ponad 50 latach od powstania teorii kwantowej pojawiły się postulaty wykorzystania kwantowomechanicznego opisu rzeczywistości w celu uzyskania zupełnie nowego spojrzenia na teorię informacji i procesy obliczeniowe (Feynman, 1982). Rodząca się w ten sposób **Informatyka Kwantowa** (ang. *Quantum Computing* – QC) (Nielsen, Chuang, 2000), zajmuje się wykorzystaniem potencjalnych możliwości miniaturowych układów kwantowych oraz praw, którym one podlegają.

Podobnie jak fizyka, tak samo **Sztuczna Inteligencja** (ang. *Artificial Intelligence* – AI) (Russell et al., 2010), stanowiąca interdyscyplinarny obszar informatyki, skupia się na opisaniu i wykorzystaniu mechanizmów natury. Czerpiąc inspiracje z obserwacji złożonych procesów w przyrodzie i zachowań uczestniczących w tych procesach istot inteligentnych, np. ewolucji biologicznej (Holland, 1975), kolektywnego zachowania zwierząt (Eberhart et al., 2001), sposobu działania ludzkiego umysłu (Kurzweil, 2012), proponuje narzędzia, m.in. systemy ekspertowe (Białko, 2005), sztuczne sieci neuronowe (Krzyśko, Wołyński, Górecki, Skorzybut, 2008), obliczenia ewolucyjne (Rutkowski, 2009),

systemy neuronowo-rozmyte (Łęski, 2008), pozwalające na efektywne rozwiązywanie problemów technicznych.

Zrozumienie natury rzeczywistości jakie przyniosła Mechanika Kwantowa, stanowiło inspiracje do podejmowania prób przeniesienia opisywanych przez nią zjawisk do dziedziny Sztucznej Inteligencji. Przykłady (Manju, Nigam, 2012) zaprezentowane w ciągu minionej dekady, pokazują, że nawet drobna „kwantowa inspiracja", rozumiana jako wprowadzenie losowości opartej na teorii kwantowej, pozwala na poprawę efektywności metod inteligencji obliczeniowej. Tak w dziedzinie Informatyki Kwantowej powstało miejsce dla metod selektywnie wykorzystujących odkrycia Mechaniki Kwantowej w implementacjach algorytmów niewymagających użycia komputera kwantowego, takich jak: **kwantowo inspirowane algorytmy ewolucyjne** (Zhang, 2010), kwantowo inspirowane sieci neuronowe (Menneer, Narayanan, 1995; Allauddin et al., 2008) czy kwantowo inspirowane metody inteligencji roju (Coelho et al., 2008; Luitel, Venayagamoorthy, 2010).

CEL, ZAKRES I STRUKTURA PRACY

Celem niniejszej pracy jest rozwiązanie problemu inżynierskiego projektowania sztucznych sieci neuronowych z wykorzystaniem kwantowo inspirowanych algorytmów genetycznych. Problem projektowania sztucznych sieci neuronowych, zdefiniowany jako poszukiwanie optymalnej jej topologii i wag synaptycznych, został już częściowo rozwiązany. W obecnym stanie wiedzy istnieją skuteczne metody uczenia sieci neuronowych (korekcji wag), brak jednak dobrych narzędzi do doboru architektury sieci, która zwykle jest projektowana eksperymentalnie. Kwantowo inspirowane algorytmy genetyczne, czerpiąc z teorii Informatyki Kwantowej, realizują niepewny z natury charakter wielu problemów świata rzeczywistego, a ich przykłady aplikacyjne pokazują, że pozwala to na poprawę efektywności poszukiwania rozwiązania w różnorodnych zadaniach optymalizacji numerycznej i kombinatorycznej, co czyni je odpowiednie do globalnej optymalizacji struktury połączeń i wag synaptycznych sztucznych sieci neuronowych. Podejście, które adaptuje kwantowo inspirowane algorytmy genetyczne do projektowania sztucznych sieci neuronowych, pozwala stworzyć kompleksowe narzędzie do budowy i uczenia sztucznych sieci neuronowych, niewymagające doświadczenia w projektowaniu jej topologii i doborze odpowiednich metod uczenia.

W pracy zgromadzono wiedzę na temat jednokierunkowych sieci wielowarstwowych, klasycznych algorytmów uczenia pierwszego i drugiego rzędu, kwantowo inspirowanych algorytmów genetycznych rzędu II oraz algorytmów ewolucji sztucznych sieci neuronowych. Ponadto opisano implementację wymienionych wyżej narzędzi w języku C# oraz sposób ich połączenia w celu rozwiązania realnych problemów projektowania sztucznych sieci neuronowych.

Struktura pracy

Treść pracy została zamknięta w siedmiu rozdziałach:

Rozdział 1 – **Sztuczne sieci neuronowe**. Przedstawiono model sztucznego neuronu (1.1) oraz jego specyfikację w postaci neuronu sigmoidalnego (1.2). Następnie opisano architekturę jednokierunkowej sieci wielowarstwowej (1.3) wraz z jej implementacją w języku C# (1.6). Ponadto dokonano przeglądu podstawowych strategii uczenia sieci neuronowych (1.4) oraz omówiono zagadnienie doboru architektury sieci (1.5).

Rozdział 2 – **Algorytm wstecznej propagacji błędów**. Omówiono proces korekcji wag w algorytmie wstecznej propagacji błędów (2.1) i organizację uczenia sztucznej sieci neuronowej (2.2) oraz opisano implementację algorytmu wstecznej propagacji błędów w języku C# (2.3) w celu uczenia sieci neuronowej z Rozdziału 1.

Rozdział 3 – **Algorytm Levenberga-Marquardta**. Przedstawiono proces korekcji wag w algorytmie Levenberga–Marquardta (3.1) i wyznaczania potrzebnej do jego realizacji macierzy Jacobiego (3.2). Opisano także organizację uczenia sztucznej sieci neuronowej z wykorzystaniem algorytmu Levenberga-Marquardta (3.3) oraz opisano jego implementację w języku C# (3.4) do uczenia sieci neuronowej z Rozdziału 1.

Rozdział 4 – **Kwantowo inspirowane algorytmy genetyczne rzędu II**. Przedstawiono reprezentację rozwiązań w kwantowo inspirowanych algorytmach genetycznych rzędu II (4.1) i używane przez nie operatory genetyczne (4.2). Opisano schemat i sposób działania kwantowo inspirowanych algorytmów genetycznych rzędu II (4.3) oraz ich implementację w języku C#.

Rozdział 5 – **Ewolucja sztucznych sieci neuronowych**. Zaprezentowano sposób kodowania (5.1) i oceny przystosowania (5.2) sztucznej sieci neuronowej. Zaproponowano rozwiązanie problemu projektowania sieci neuronowej algorytmem kwantowo inspirowanym algorytmem genetycznym (5.3) oraz opisano implementację tego rozwiązania w języku C# (5.4) dla sieci neuronowej z Rozdziału 1.

Rozdział 6 – **Badania doświadczalne**. Przedstawiono zastosowaną procedurę badawczą (6.1) oraz uzyskane w wyniku przeprowadzonych badań wyniki (6.2).

Rozdział 7 – **Podsumowanie i wnioski**.

Informacje uzupełniające zostały przedstawione w załącznikach. W Załączniku A przedstawiono ideę algorytmów uczących pierwszego rzędu. W Załączniku B wyprowadzono regułę korekcji wag neuronu sigmoidalnego. W Załączniku C opisano wyprowadzenie reguły korekcji wag w algorytmie wstecznej propagacji błędów. Złącznik D zawiera opis idei algorytmów uczących drugiego rzędu. W Załączniku E wyprowadzono regułę korekcji wag w algorytmie Levenberga-Marquardta. W Załączniku F, Załączniku G oraz Załączniku H omówiono wyprowadzenia odpowiednio metod: wyznaczania jawnej formy macierzy Jacobiego metodą propagacji w tył, wyznaczania niejawnej formy macierzy Jacobiego metodą propagacji w tył i aproksymacji jawnej formy macierzy Jacobiego metodą propagacji w przód.

Rozdział 1

SZTUCZNE SIECI NEURONOWE

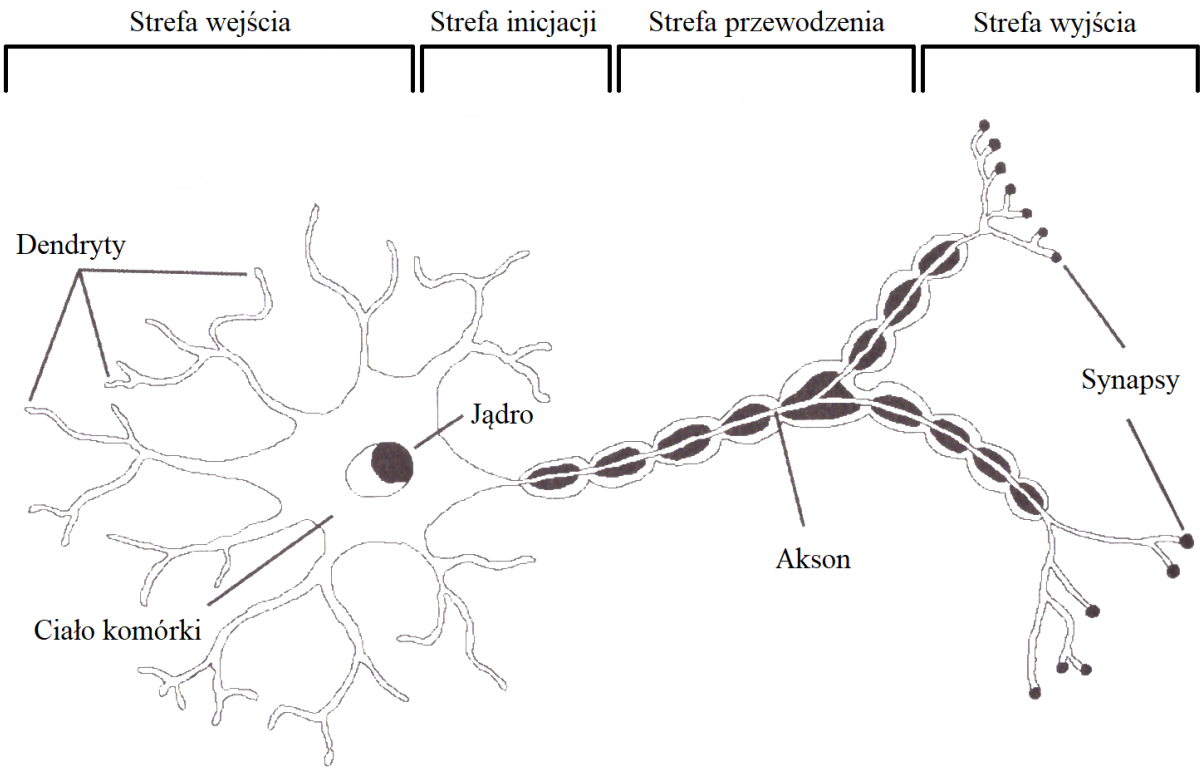
Interdyscyplinarne podejście do prowadzenia badań naukowych zastosowane podczas pierwszych prób wykorzystania mechanizmów świata natury do rozwiązania problemów sterowania i komunikacji (Rosenblueth, Wiener, Bigelow, 1943), doprowadziło w pracy (Wiener, 1948) do narodzin nowej dziedziny nauki – *Cybernetyki* (od gr. κυβερνήτης kybernetes – sternik, zarządca). W myśl towarzyszącej jej idei prowadzono badania nad połączeniami synaptycznymi włókien nerwowych, polegające na stosowaniu logiki matematycznej do modelowania procesów przełączania, zwieńczone pracą (McCulloch, Pitts, 1943) definiującą **model neuronu** działającego według zasady „wszystko albo nic" oraz pokazującą możliwość aproksymacji dowolnej funkcji przez **sieć połączonych komórek nerwowych**.

Realizacja funkcjonalności komórki nerwowej w postaci programowej symulacji lub elektronicznego wielowejściowego układu sumującego prowadzi do stworzenia **sztucznych neuronów**, które połączone w strukturę utworzą **sztuczną sieć neuronową**. Podstawowymi własnościami sztucznego neuronu są: zbieranie i przetwarzanie sygnałów wejściowych (bodźców) oraz wysyłanie sygnału wyjściowego (pobudzenia). Sztuczne sieci neuronowe, zorganizowane w warstwy, przetwarzają sygnały propagując je między kolejnymi połączonymi ze sobą neuronami, z wejść na wyjścia sieci. Współczesne sieci neuronowe, których przeglądu dokonano w pracy (Łęski, 2008), dzięki takim właściwościom jak uogólnianie, adaptacyjność, nieliniowość czy tolerancja błędów, znalazły szerokie zastosowanie w rozwiązywaniu m.in. problemów sterowania, aproksymacji i predykcji funkcji oraz klasyfikacji obiektów.

W niniejszym rozdziale przedstawiono model sztucznego neuronu, jego specyfikację w postaci neuronu sigmoidalnego oraz ideę jednokierunkowej sieci wielowarstwowej wraz z opisem jej implementacji w języku C#. Ponadto dokonano przeglądu podstawowych strategii uczenia sieci neuronowych oraz omówiono zagadnienie doboru architektury sieci.

1.1 MODEL SZTUCZNEGO NEURONU

Podstawowym elementem budującym układ nerwowy jest komórka nerwowa, nazywana neuronem (Bear, Connors, Paradiso, 2007). Neuron jest rodzajem komórki pobudliwej elektrycznie, przetwarza i przekazuje informacje w postaci sygnałów elektrycznych. Uproszczony schemat neuronu został przedstawiony na Rysunku 1.1. W neuronie można wyróżnić ciało komórki (perikarion, somę), zawierające organelle komórkowe, oraz odchodzące od niego dwa rodzaje wypustek (neurytów): wprowadzające informację do neuronu dendryty i wyprowadzający informację z neuronu akson. Informacje w postaci impulsów nerwowych (sygnałów elektrycznych) propagują z zakończeń aksonu jednego neuronu do dendrytów innych komórek poprzez połączenia nazywane synapsami.



Rysunek 1.1 Budowa komórki nerwowej – neuronu

Synapsy w sposób chemiczny lub elektryczny przewodzą impulsy nerwowe, które podczas tego procesu mogą ulec wzmocnieniu lub osłabieniu. W wyniku działania synaps do neuronów docierają sygnały pobudzające oraz hamujące. Gdy ciało danej komórki zostanie pobudzone przez odpowiednio silny bodziec, na początkowym odcinku aksonu dochodzi do powstania potencjału czynnościowego, który następnie propaguje do jego zakończeń, gdzie jest przekazywany dalej przez kolejne synapsy. Przewodzenie sygnałów przez komórkę

nerwową podlega zasadzie wszystko albo nic tzn. neuron wytwarza potencjał czynnościowy lub nie, a wszystkie powstające potencjały czynnościowe w danej komórce nerwowej mają tę samą wielkość, niezależnie od wielkości bodźca (o ile tylko jest on wystarczająco intensywny, aby aktywować neuron).

Z perspektywy funkcjonalnej neuron można podzielić na cztery strefy, zaznaczone na Rysunku 1.1:

- strefa wejścia – dendryty i ciało komórki, miejsce odbierania i przetwarzania impulsów nerwowych odbieranych od innych neuronów.
- strefa inicjacji – początkowy odcinek aksonu, miejsce powstawania potencjału czynnościowego.
- strefa przewodzenia – akson, miejsce propagowania potencjału czynnościowego w postaci impulsu nerwowego.
- strefa wyjścia – zakończeniach aksonu i synapsy, miejsce przekazywania i wartościowania impulsów nerwowych.

Pierwsze próby matematycznego sformalizowania opisu działania komórki nerwowej podjęte w pracach (McCulloch, Pitts, 1943; McCulloch, Pitts, 1947), doprowadziły do zdefiniowania modelu sztucznego neuronu przedstawionego na Rysunku 1.2. Przyjmując następujące oznaczenia:

n – liczba wejść neuronu,

$x_0, \ldots, x_n$ – sygnały wejściowe, $\boldsymbol{x} = [x_0, \ldots, x_n]^T$,

$w_0, \ldots, w_n$ – wagi synaptyczne, $\boldsymbol{w} = [w_0, \ldots, w_n]^T$,

net – suma ważona sygnałów wejściowych,

f – funkcja aktywacji,

y – wartość wyjściowa neuronu,

działanie sztucznego neuronu można opisać zależnością:

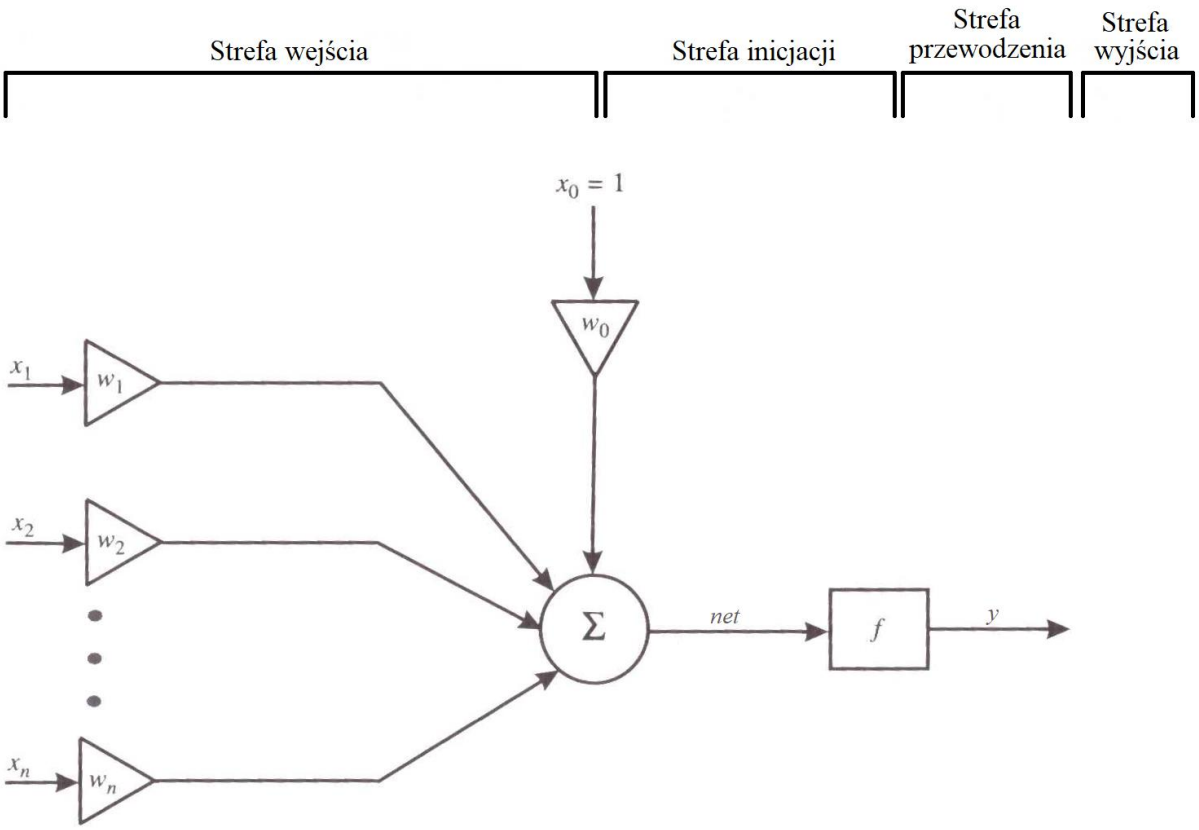
$$y = f(net) \tag{1.1}$$

gdzie

$$net = x^T w = \sum_{i=0}^{n} x_i w_i \tag{1.2}$$

Sygnały wejściowe $\boldsymbol{x} = [x_0, \ldots, x_n]^T$ zostają pomnożone przez odpowiadające im wagi synaptyczne $\boldsymbol{w} = [w_0, \ldots, w_n]^T$, a otrzymane w ten sposób wartości są sumowane. Sygnał *net*

części liniowej neuronu jest następnie poddany działaniu funkcji aktywacji *f*, której wynikiem jest wartość wyjściowa neuronu *y*. Wartość sygnału x_0 jest równa 1, a odpowiadająca mu waga w_0 to próg (bias). Wiedza neuronu jest zapisana w jego wagach synaptycznych. Przy tworzeniu nowego neuronu wagi są dobierane w sposób losowy, następnie neuron poddaje się procesowi uczenia, który prowadzi do korekcji wag i uzyskania neuronu poprawnie interpretującego wprowadzane do niego sygnały wejściowe.



Rysunek 1.2 Model sztucznego neuronu

1.2 MODEL NEURONU SIGMOIDALNEGO

Funkcja aktywacji

Jedną ze znanych specyfikacji modelu sztucznego neuronu, przedstawioną na Rysunku 1.3, jest neuron sigmoidalny (Rutkowski, 2009), którego nazwa pochodzi od jego funkcji aktywacji będącej funkcją sigmoidalną unipolarną:

$$f(x) = \frac{1}{1 + e^{-\beta x}} \tag{1.3}$$

lub bipolarną:

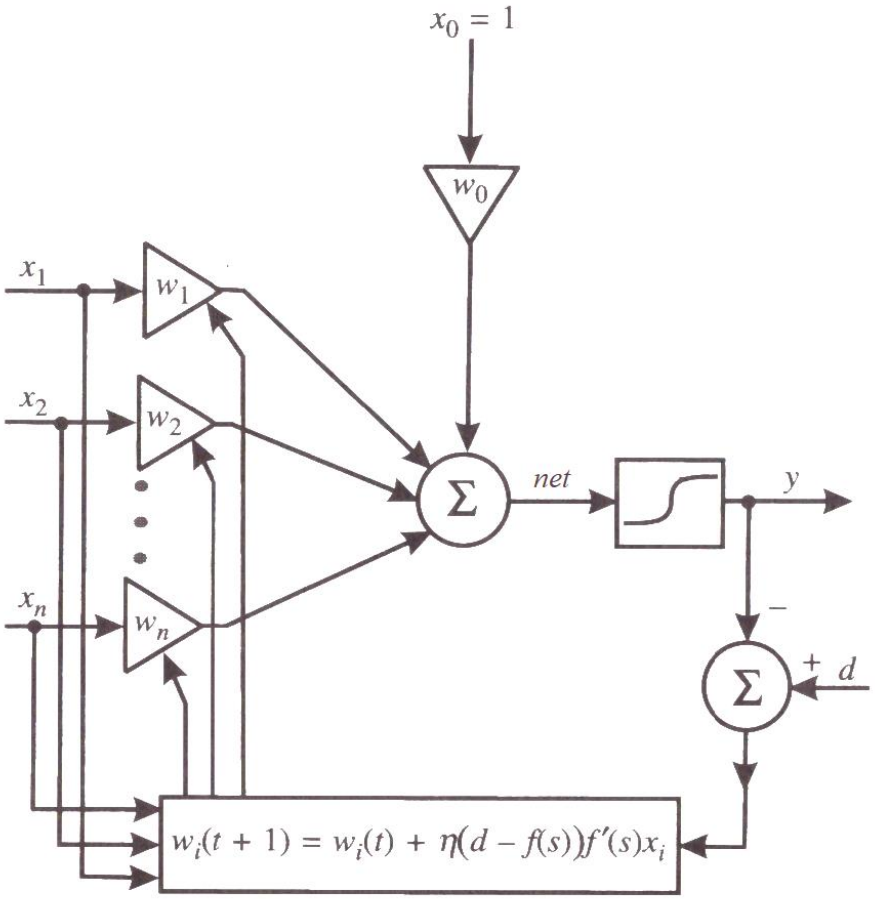
$$f(x) = tanh(\beta x) = \frac{1 - e^{-\beta x}}{1 + e^{-\beta x}} \tag{1.4}$$

Współczynnik β ma wpływ na przebieg funkcji sigmoidalnej: dla małych wartości β funkcja ma kształt łagodny, wraz ze wzrostem współczynnika funkcja staje się coraz bardziej stroma, dla dużych wartości β ma charakter progowy. Wpływ współczynnika β na przebieg funkcji sigmoidalnej został przedstawiony na Rysunku 1.4. Istotną cechą neuronów sigmoidalnych jest różniczkowalność ich funkcji aktywacji, niezbędna w procesie uczenia. Pochodne sigmoidalnych funkcji aktywacji przyjmują następujące postacie, dla funkcji unipolarnej:

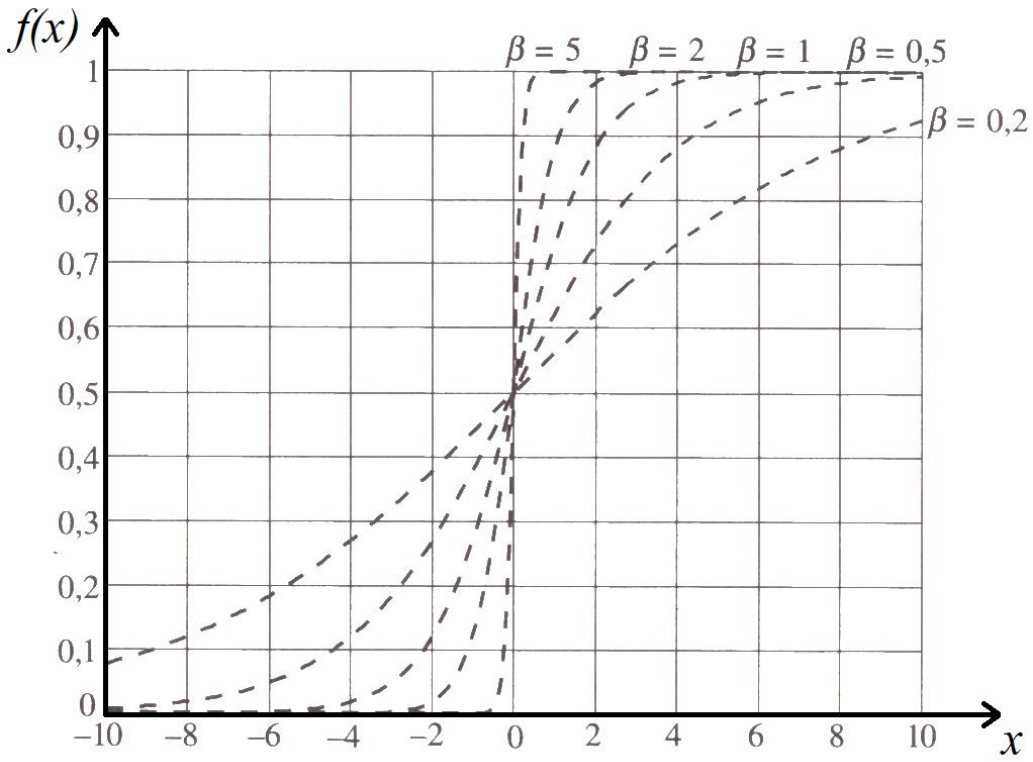
$$f'(x) = \beta f(x)(1 - f(x)) \tag{1.5}$$

oraz dla funkcji bipolarnej:

$$f'(x) = \beta(1 - f^2(x)) \tag{1.6}$$



Rysunek 1.3 Schemat neuronu sigmoidalnego



Rysunek 1.4 Przebiegi unipolarnych funkcji sigmoidalnych f(x) dla różnych wartości parametru β

Korekcja wag synaptycznych

Algorytm korekcji wag synaptycznych neuronu sigmoidalnego, wyprowadzony w Załączniku B, przyjmuje następującą formę:

$$y = f(net) = f\left(\sum_{i=0}^{n} w_i\, x_i\right) \tag{1.7}$$

$$w_i^* = w_i + \eta\big(d - f(net)\big)\, f'(net) x_i \tag{1.8}$$

gdzie η jest współczynnikiem uczenia, d oznacza wartość wzorcową oczekiwaną na wyjściu neuronu w odpowiedzi na wektor sygnałów wejściowych $\boldsymbol{x}$, a w_i^* to skorygowana waga w_i.

Do określenia błędu neuronu dla ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, d_1), \ldots, (\boldsymbol{x}_{|P|}, d_{|P|})\}$ może zostać użyta miara

$$Q = \frac{1}{2}\sum_{p=1}^{|P|}\big(d_p - y_p\big)^2 \tag{1.9}$$

Proces uczenia neuronu sigmoidalnego został przedstawiony na Listingu 1. Podczas tworzenia neuronu wektor wag synaptycznych $\boldsymbol{w}$ przyjmuje losowe wartości. Następnie obliczany jest błąd neuronu Q (1.9) dla ciągu wzorów uczących P i porównywany z maksymalnym dopuszczalnym błędem Q_{max}. Dopóki błąd Q jest większy od błędu Q_{max}, algorytm iteruje przez wszystkie elementy ciągu uczącego P i koryguje wektor wag synaptycznych $\boldsymbol{w}$, korzystając ze wzorów (1.7) oraz (1.8). Gdy błąd Q neuronu stanie się mniejszy lub równy maksymalnemu dopuszczalnemu błędowi Q_{max}, algorytm kończy działanie, neuron sigmoidalny jest nauczony rozpoznawania elementów ciągu P.

Listing 1: Uczenie neuronu sigmoidalnego

```
    Input:  P = {(x_1, d_1), ..., (x_|P|, d_|P|)}   – zestaw uczący
            Q_max – maksymalny dopuszczalny błąd neuronu dla zestawu uczącego P
 1  begin
 2      Stwórz neuron z losowymi wagami synaptycznymi
 3      Q ← błąd dla zestawu uczącego P
 4      while Q > Q_max do
 5          for p in 1, ..., |P| do
 6              x ← x_p
 7                  for i in 0, ..., n do
 8                      w_i ← w_i + η · (d_p – f(net)) · f'(net) · x_i
 9                  end
10          end
11          Q ← błąd dla zestawu uczącego P
12      end
13  end
```

1.3 JEDNOKIERUNKOWE SIECI WIELOWARSTWOWE

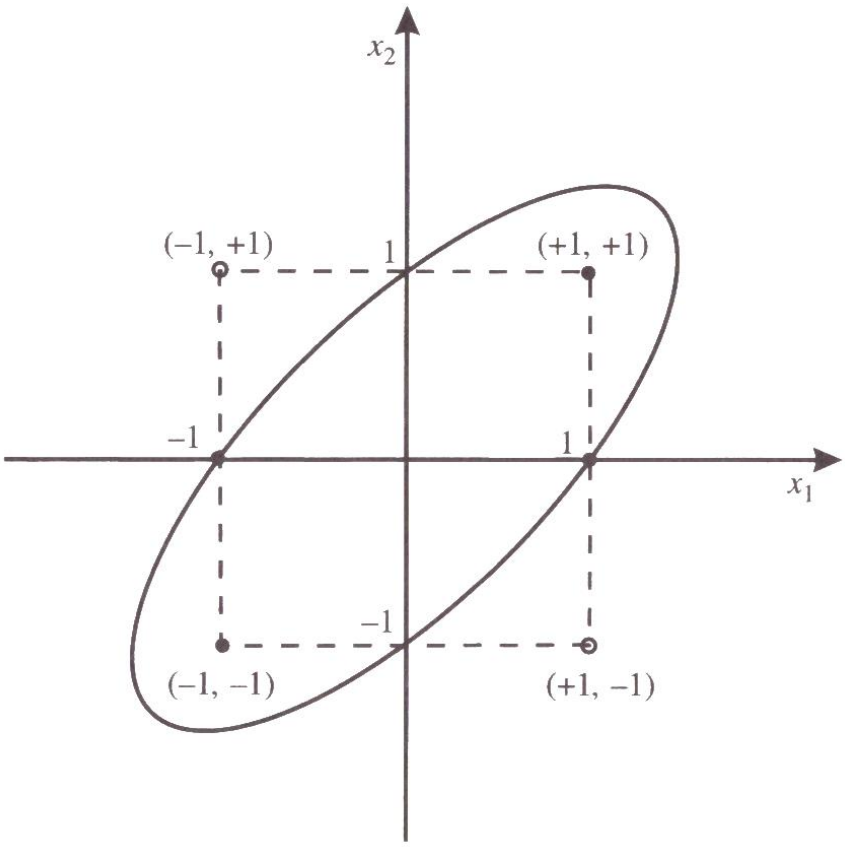
Problem XOR

Przedstawiony w Rozdziale 1.1 oraz 1.2 model sztucznego neuronu jest klasyfikatorem liniowym, który uczy się, modyfikując wartości swoich wag synaptycznych na podstawie wiedzy zawartej w ciągu uczącym. Sztuczny neuron o n wejściach jest w stanie podzielić n-wymiarową przestrzeń na dwie półprzestrzenie przy pomocy n-1-wymiarowej hiperprzestrzeni, reprezentowanej przez funkcję *net* (1.2) po procesie uczenia. Zakres problemów możliwych do rozwiązania przez klasyfikator liniowy jest jednak mocno ograniczony. Klasycznym przykładem problemu, który wykracza poza możliwości sztucznego neuronu, jest problem funkcji logicznej XOR przedstawiony w pracy (Minsky, Papert, 1969). Ciąg uczący dla tego problemu zawarto w Tabeli 1.1. Rysunek 1.5, ilustrujący wartości ciągu uczącego, pokazuje, że problem XOR jest problemem liniowo

nieseparowanym, tzn. nie istnieje prosta rozdzielająca punkty funkcji XOR o wartości -1 od punktów o wartościach 1. W tym przypadku przykładową granicą decyzyjną może być nieosiągalna przez pojedynczy klasyfikator liniowy elipsa.

Tabela 1.1 Ciąg wzorów uczących dla problemu XOR

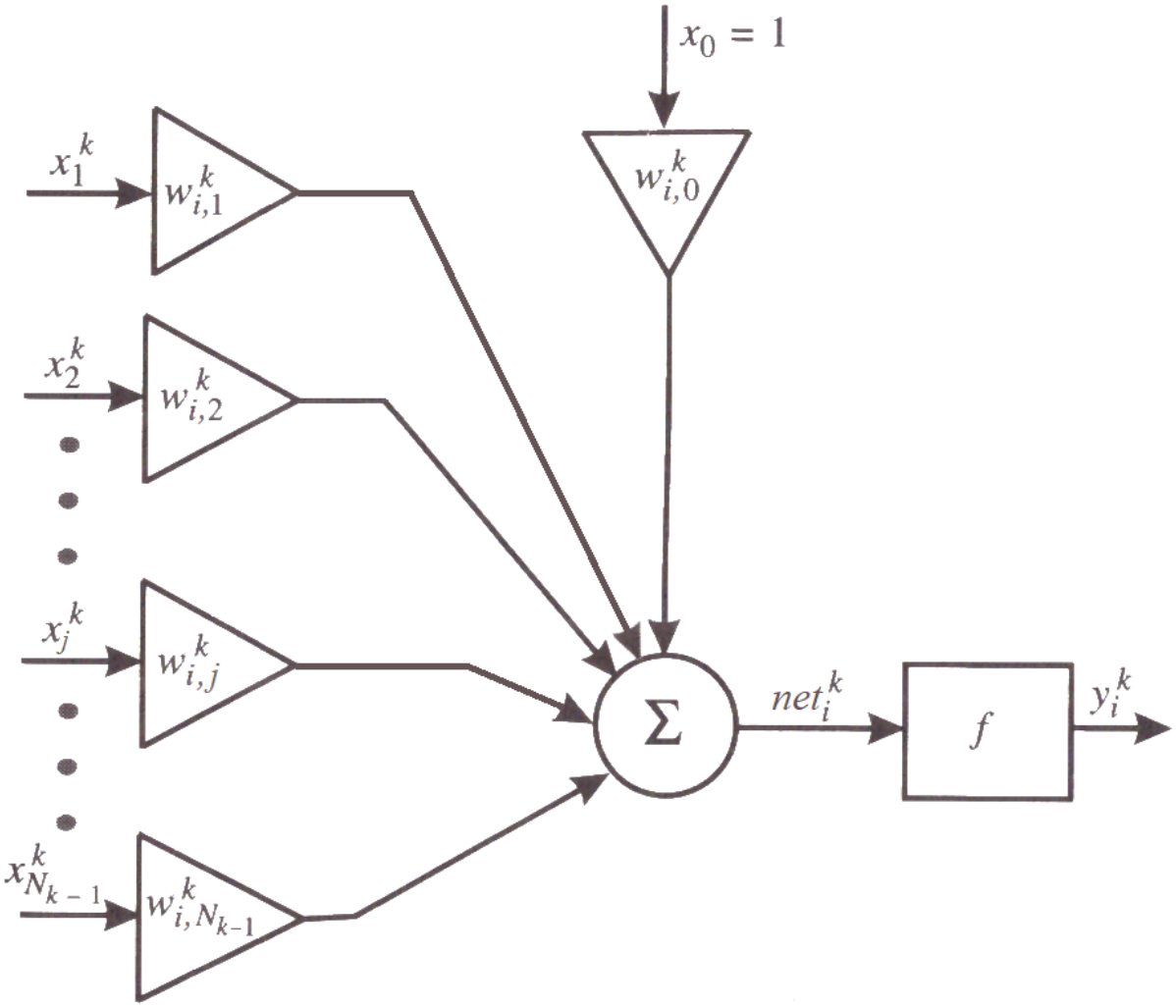
lp.	x_1	x_2	$d = \mathrm{XOR}(x_1, x_2)$
1	1	1	-1
2	1	-1	1
3	-1	1	1
4	-1	-1	-1



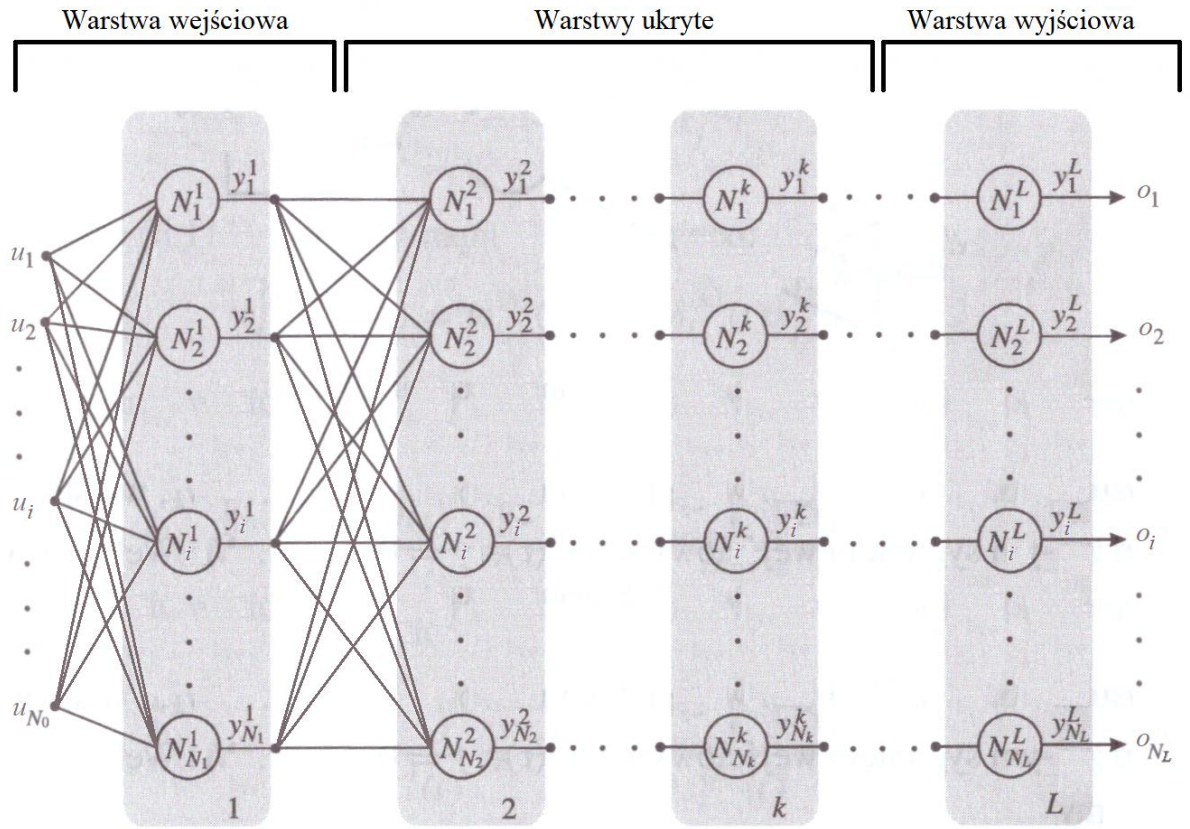
Rysunek 1.5 Ilustracja problemu XOR

Jednokierunkowe sieci wielowarstwowe

W celu rozwiązywania problemów liniowo nieseparowalnych neurony, których schemat przedstawia Rysunek 1.6, łączy się w **sztuczne sieci neuronowe** (ang. *artificial neural networks* – ANN). Jedną z możliwych organizacji takich połączeń jest jednokierunkowa sieć wielowarstwowa (Rutkowski, 2009), przedstawiona na Rysunku 1.7. Sieć zbudowana jest z co najmniej dwóch warstw: wejściowej oraz wyjściowej, między którymi mogą się znajdować warstwy ukryte. Kolejne warstwy sieci są ze sobą **w pełni połączone** (ang. *fully connected* – FC), tj. wyjście każdego neuronu N_i^{k-1} w warstwie $k - 1$ (poprzedniej) jest jednym z wejść każdego neuronu $\boldsymbol{N_i^k}$ w warstwie $k = 2, \ldots, L$ (następnej). Neurony N_i^k w obrębie jednej warstwy k nie mogą się ze sobą łączyć, mogą jedynie przekazywać sygnały między różnymi warstwami k.



Rysunek 1.6 Schemat sztucznego neuronu N_i^k w jednokierunkowej sieci wielowarstwowej



Rysunek 1.7 Jednokierunkowa sieć wielowarstwowa

Proces propagacji sygnałów w jednokierunkowej sieci wielowarstwowej przestawiono w postaci pseudokodu na Listingu 1. Podczas przetwarzania wektor sygnałów wejściowych $\boldsymbol{u} = [u_1, \ldots, u_{N0}]^T$ jest wprowadzany do warstwy wejściowej ($k = 1$) a następnie przetwarzany i przekazywany przez kolejne warstwy k sieci, aż dotrze na wyjścia neuronów N_i^L w warstwie wyjściowej ($k = L$) przybierając reprezentację wektora sygnałów wyjściowych $\boldsymbol{o} = [o_1, \ldots, o_{NL}]^T$. Taki sposób przetwarzania sygnałów określa się mianem metody propagacji w przód, a realizującą ją sieć – **siecią skierowaną** (ang. *feed-forward* – FF).

Listing 2: Propagacja sygnałów w jednokierunkowej sieci wielowarstwowej

Input: $\mathbf{u} = [u_1, \ldots, u_{N0}]^T$ – wektor sygnałów wejściowych
Result: $\mathbf{o} = [o_1, \ldots, o_{NL}]^T$ – wektor sygnałów wyjściowych

```
1  begin
2      x^1 ← u
3      for k in 1, ..., L do
4          for i in 1, ..., N_k do
5              y_i^k ← f(net_i^k)
6          end
7      end
8      o ← y^L
9  end
```

1.4 STRATEGIE UCZENIA SZTUCZNYCH SIECI NEURONOWYCH

Wiedza sztucznej sieci neuronowej, tak jak wiedza pojedynczego neuronu, jest zapisana w wagach synaptycznych. Przy tworzeniu sieci wagi połączeń synaptycznych między budującymi sieć neuronami przybierają losowe wartości. Analogicznie do procesu uczenia sztucznego neuronu, przedstawionego w Rozdziale 1.2 na przykładzie neuronu sigmoidalnego, opracowano liczne metody uczenia sieci neuronowych. Historycznie pierwszą metodą adaptowania sieci do wypracowania lepszych zachowań jest hebowska reguła uczenia (Hebb, 1949), zwiększająca wagi połączeń neuronów jednocześnie aktywnych. Obecnie rozróżnia się cztery podstawowe strategie uczenia sieci neuronowych (Łęski, 2008):

- **Uczenie z nadzorem** (uczenie z nauczycielem) (ang. *supervised learning*) – w procesie uczenia sieć odpowiada na kolejne zestawy sygnałów wejściowych. Odpowiedzi sieci są następnie porównywane z wartościami wzorcowymi, a wyznaczona między nimi różnica nazywana jest błędem. Na podstawie otrzymanych błędów dokonuje się modyfikacji wag synaptycznych w sieci prowadzącej minimalizacji błędu.
- **Uczenie bez nadzoru** (uczenie bez nauczyciela) (ang. *unsupervised learning*) – sieć otrzymuje kolejne zestawy sygnałów wejściowych, które samodzielnie dzieli na klasy. W procesie uczenia nie otrzymuje wiedzy o pożądanych odpowiedziach, odbierane przez nią bodźce są przyporządkowywane do istniejących klas lub tworzą nową klasę.
- **Uczenie ze współzawodnictwem** (specjalizacja) (ang. *concurrent learning*) – podczas uczenia neurony konkurują ze sobą o uzyskanie stanu aktywacji. Wzmocnieniu ulegają te neurony, których odpowiedź jest najbliższa odpowiedzi pożądanej, np. uzyskują najwyższy potencjał aktywacji.
- **Uczenie ze wzmocnieniem** (uczenie z krytykiem) (ang. *reinforcement learning*) – uczenie systemu (sieci) odbywa się poprzez jej ciągłą interakcję ze środowiskiem (otoczeniem). Na wejście sieci wprowadzane są informacje o stanie środowiska, a jej wyjście stanowią proponowane przez system akcje (działania), prowadzące do osiągnięcia optymalnego stanu środowiska. Podczas uczenia nadzorca (krytyk) ocenia za pomocą wskaźników skalarnych efekty działania sieci. Tendencje sieci do podejmowania działań pozytywnych są wzmacniane, a negatywnych – osłabiane.

1.5 DOBÓR ARCHITEKTURY SIECI

Struktura sztucznej sieci neuronowej opisana w Rozdziale 1.3 formalizuje tylko podstawowe zagadnienia jej organizacji takie jak: grupowanie neuronów w warstwy, budowanie połączeń między warstwami czy sposób przetwarzania sygnałów wejściowych. Implementacja jednokierunkowej sieci wielowarstwowej wymaga jednak dokładnego zdefiniowania szczegółów jej architektury, tj. określenia co najmniej: liczby warstw oraz rodzaju i liczby neuronów w każdej z nich.

Pojedynczy sztuczny neuron dzieli płaszczyznę przestrzeni problemu na dwie półpłaszczyzny, co mocno ogranicza zakres możliwych do rozwiązania przez niego problemów. Aby usunąć to ograniczenie, neurony należy łączyć w warstwy, jak pokazano w Rozdziale 1.3. Dwie warstwy sztucznych neuronów mogą odwzorowywać wypukłe ograniczone hiperpłaszczyznami obszary (simpleksy), a za pomocą trzech warstw można modelować już dowolny obszar. Istnieje niewiele problemów, do rozwiązania których potrzeba więcej niż dwóch warstw ukrytych. Sieci trójwarstwowe i głębsze są w stanie rozwiązywać pełną gamę problemów klasyfikacji i aproksymacji (Rutkowski, 2009).

Twierdzenie Kołmogorowa

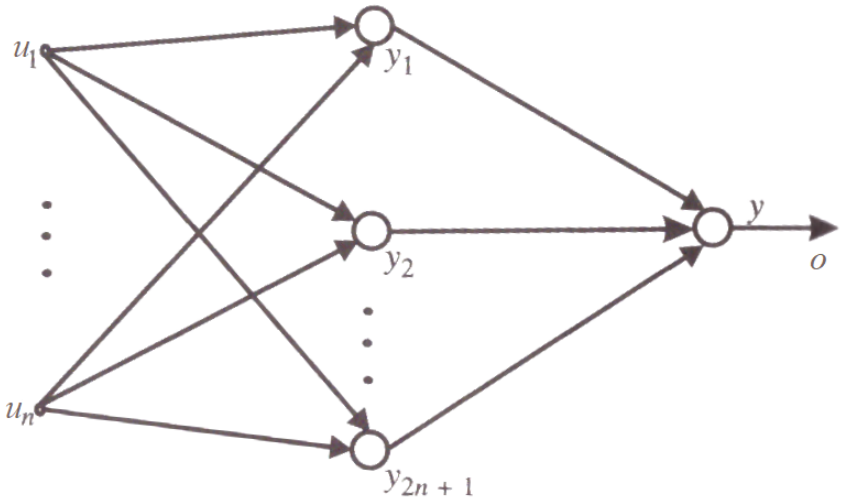
Istotnym punktem w rozważaniach problemu aproksymacji funkcji jest twierdzenie Kołmogorowa o reprezentacji ciągłej funkcji wielu zmiennych (Kolmogorov, 1957). Zgodnie z twierdzeniem Kołmogorowa dowolną ciągłą funkcję rzeczywistą $f(x_1, \ldots, x_n)$ określoną na jednostkowej kostce, tj. przestrzeni $[0, 1]^n$, gdzie $n \geq 2$, można aproksymować za pomocą funkcji F danej wzorem:

$$F(x) = \sum_{j=1}^{2n+1} g_j \left(\sum_{i=1}^{n} \phi_{i,j}(x_i) \right) \tag{1.10}$$

gdzie $\boldsymbol{x} = [x_1, \ldots, x_n]^T$, g_j, $j = 1, \ldots, 2n+1$, są odpowiednio dobranymi ciągłymi funkcjami jednej zmiennej (zewnętrznymi funkcjami aktywacji), natomiast $\varphi_{i,j}$, $i = 1, \ldots, n$, $j = 1, \ldots, 2n+1$, są funkcjami ciągłymi i monotonicznie rosnącymi, niezależnymi od f (wewnętrznymi funkcjami aktywacji).

Zależności (1.10) odpowiada, przedstawiona na Rysunku 1.8, struktura dwuwarstwowej sieci neuronowej o n wejściach, $2n+1$ neuronach w warstwie ukrytej i jednym neuronie o liniowej funkcji aktywacji w warstwie wyjściowej. Twierdzenie Kołmogorowa przeniesione na grunt

teorii sieci neuronowych udowadnia, że sieć neuronowa może aproksymować dowolną funkcję ciągłą *n* zmiennych zdefiniowaną na $[0, 1]^n$. Twierdzenie ma jednak charakter teoretyczny, nie określa rodzaju funkcji nieliniowych oraz metod uczenia sieci. Tym problemem zajęto się w wielu późniejszych pracach, definiując niezbędne algorytmy konstruktywne (Sprecher, 1996; Igielnik, 2003), a nawet dokonując modyfikacji oryginalnego twierdzenia (Michalkiewicz, 2012).



Rysunek 1.8 Struktura sieci neuronowej umożliwiająca aproksymację dowolnej funkcji ciągłej zgodnie z twierdzeniem Kołmogorowa

Problem poszukiwania optymalnej architektury

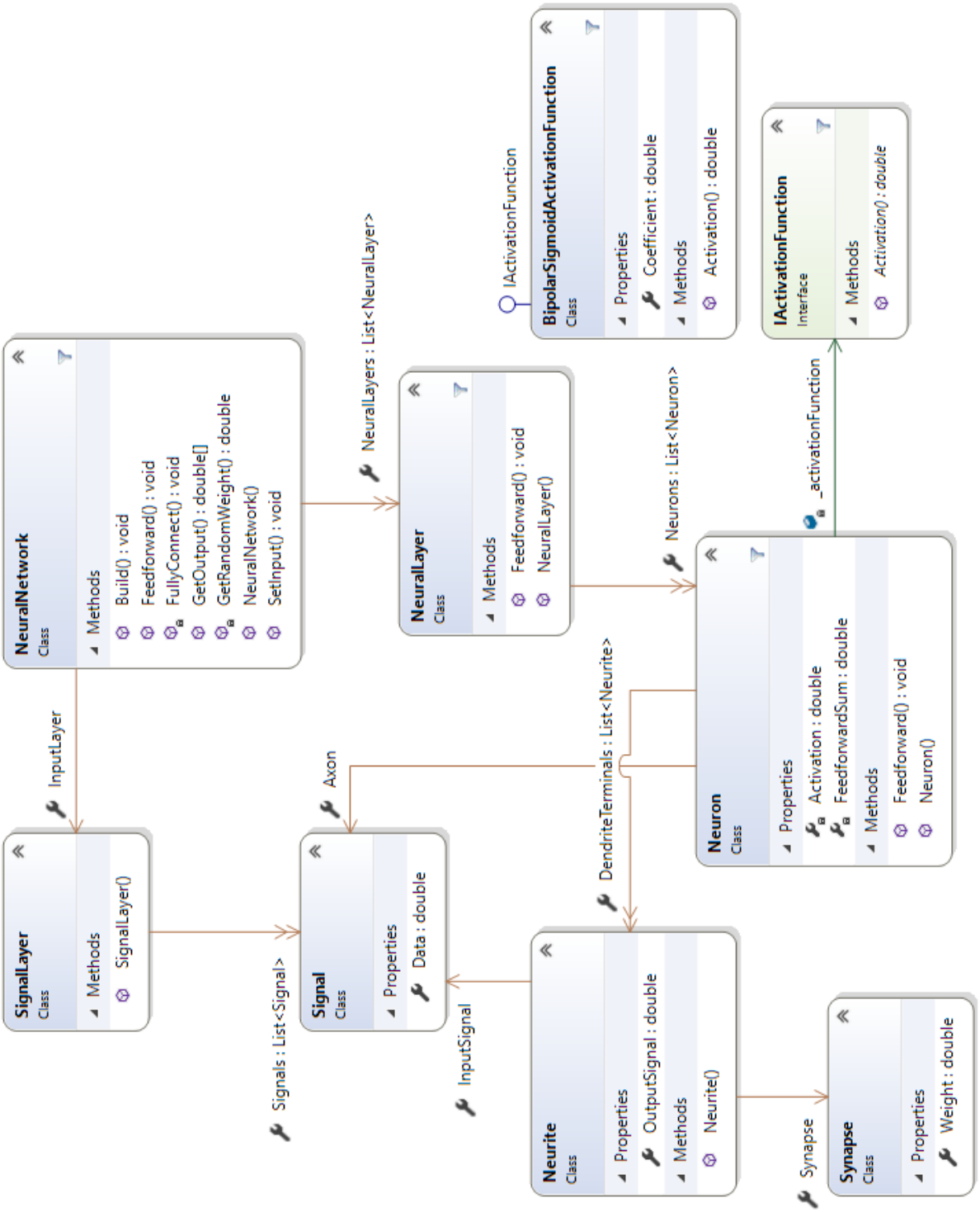
Duży wpływ na działanie sieci ma liczba neuronów w poszczególnych jej warstwach. Wraz ze wzrostem liczby neuronów nie tylko wydłuża się proces uczenia sieci, ale również rośnie koszt uzyskania odpowiedzi sieci na podawane sygnały wejściowe. Ponadto przy niewielkiej, w porównaniu z rozmiarem sieci, liczbie próbek uczących można strukturę sieci „przeuczyć", tj. pozbawić ją możliwości uogólniania wiedzy (aproksymacji). Sieć „przeuczona" będzie doskonale potrafiła odwzorowywać próbki poznane w procesie uczenia, nauczy się ich „na pamięć", mając jednocześnie upośledzone zdolności interpretacji próbek niezawartych w ciągu uczącym, czego przykład można znaleźć w pracy (Rutkowski, 2009).

Osiągnięcie poprawnie nauczonej, zadowalająco aproksymującej sieci nie zawsze oznacza znalezienie sieci o optymalnej architekturze. Często, po skończonym procesie uczenia, strukturę sieci można uprościć, dokonać oczyszczania sieci, np. poprzez usunięcie wag o minimalnym wpływie na jakość jej działania (Le Cun, Denker, Solla, 1990). Algorytmy oczyszczające prowadzą do nowego spojrzenia na problem projektowania sieci,

przygotowując miejsce dla algorytmów konstrukcyjnych trenujących sieci o zmiennej architekturze (Białko, 2005), które prowadzą do uzyskania bardziej skomplikowanych sieci skierowanych niż rozważana w tej pracy jednokierunkowa sieć wielowarstwowa.

1.6 IMPLEMENTACJA JEDNOKIERUNKOWEJ SIECI WIELOWARSTWOWEJ W JĘZYKU C#

Implementacja, opisanej w Rozdziale 1.4, jednokierunkowej sieci wielowarstwowej została przedstawiona w postaci diagramu klas na Rysunku 1.9. Klasa NeuralNetwork reprezentuje sieć skierowaną. Funkcję warstwy sygnałów wejściowych sieci pełni obiekt InputLayer klasy SignalLayer, będącej kontenerem sygnałów – obiektów klasy Signal. Warstwy ukryte oraz wyjściową sieci realizują instancje klasy NeuralLayer, które komponują obiekty klasy Neuron, odwzorowujące sztuczne neurony. Rodzaj neuronu określa przypisana mu implementacja interfejsu IActivationFunction, realizując wzorzec Strategy (Gamma, Helm Johnson, Vlissides, 1993). Klasa BipolarSigmoidActivationFunction implementująca interfejs IActivationFunction, definiuje sposób działania bipolarnego neuronu sigmoidalnego. Połączenia między neuronami są realizowane jako obiekty klasy Neurite, propagujące sygnały Signal z uwzględnieniem przypisanych do nich wag synaptycznych Synapse.



Rysunek 1.9 Diagram klas przedstawiający implementację jednokierunkowej sieci wielowarstwowej w języku C#

Rozdział 2

ALGORYTM WSTECZNEJ PROPAGACJI BŁĘDÓW

Pierwsze prace: (Bryson, Denham, Dreyfus, 1963), (Bryson, Ho, 1969), (Werbos, 1974), opisujące metodę uczenia sztucznej sieci neuronowej z nadzorem opartą na wstecznej propagacji błędów, która umożliwiała praktyczną realizację sieci rozwiązujących problemy liniowo nieseparowalne (Minsky, Papert, 1969), przeszły w dużej mierze niezauważone przez środowisko naukowe . Dopiero publikacja prac (Rummelhart, Hinton, Williams, 1986) oraz (Rummelhart, McClelland, 1986) doprowadziła do rozpowszechnienia **algorytmu wstecznej propagacji błędów** (ang. *Error Backpropagation Training* – EBP) jako metody uczenia wielowarstwowych sieci neuronowych.

Algorytm wstecznej propagacji błędów jako **algorytm uczący rzędu pierwszego** (Załącznik A), koryguje wagi synaptyczne sieci wykorzystując wektor gradientu, tj. wektor pierwszych pochodnych miary błędu sieci po zawartych w niej wagach synaptycznych. W tym celu wprowadza mechanizm wstecznej propagacji błędów, umożliwiający przedstawienie błędów warstwy wyjściowej w postaci funkcji wag synaptycznych neuronów w warstwach ukrytych. Następnie korzystając ze spropagowanych wstecz błędów oblicza elementy wektora gradientu i na ich podstawie koryguje wagi synaptyczne sieci.

W niniejszym rozdziale przedstawiono regułę korekcji wag w algorytmie EBP oraz schemat i sposób jego działania dla jednokierunkowej sieci wielowarstwowej z Rozdziału 1.3. Ponadto opisano praktyczną implementację algorytmu EBP w języku C# do uczenia sieci zrealizowanej w Rozdziale 1.6.

2.1 KOREKCJA WAG W ALGORYTMIE EBP

Algorytm wstecznej propagacji błędów, wyprowadzony w Załączniku C, prowadzący do korekcji wag sztucznej sieci neuronowej zaprezentowanej w Rozdziale 1.3, przyjmuje następującą formę:

$$y_i^k = f\left(net_i^k\right), \qquad net_i^k = \sum_{j=0}^{N_{k-1}} x_j^k \, w_{i,j}^k \tag{2.1}$$

$$\varepsilon_i^k = \begin{cases} d_i - o_i, & k = L \\ \sum_{m=1}^{N_{k+1}} \delta_m^{k+1} \, w_{m,i}^{k+1}, & k = 1, 2, \ldots, L-1 \end{cases} \tag{2.2}$$

$$\delta_i^k = f'\left(net_i^k\right)\varepsilon_i^k \tag{2.3}$$

$$w_{i,j}^{k\,*} = w_{i,j}^k + 2\eta\delta_i^k \, x_j^k \tag{2.4}$$

gdzie η jest współczynnikiem uczenia, d_i oznacza wartość wzorcową oczekiwaną na wyjściu sieci o_i w odpowiedzi na wektor sygnałów wejściowych $\boldsymbol{x}$, a $w_{i,j}^{k*}$ to skorygowana waga $w_{i,j}^{k}$.

Listing 1: Korekcja wag w algorytmie EBP

```
    Input:  p = (x, d)   –  wzór uczący
1   begin
2       u ← x
3       Propaguj sygnały w przód
4       for k in L, …, 1 do
5           for i in 1, …, Nk do
6               if k = L then
7                   εi^k ← di - oi
8               else
9                   εi^k ←∑_{m=1}^{Nk+1} δi^{k+1} · wm,i^{k+1}
10              end
11              δi^k ← f'(neti^k) · εi^k
12          end
13      end
14      for k in 1, …, L do
15          for i in 1, …, Nk do
16              for j in 0, …, Nk-1 do
17                  wi,j^k ← wi,j^k + 2η · δi^k · xj^k
18              end
19          end
20      end
21  end
```

Proces korekcji wag sztucznej sieci neuronowej z wykorzystaniem algorytmu EBP na podstawie wzoru uczącego $p = (\boldsymbol{x}, \boldsymbol{d})$ został przedstawiony na Listingu 1. Wektor sygnałów uczących $\boldsymbol{x}$ zostaje wprowadzony na wejście sieci $\boldsymbol{u}$ i spropagowany w przód na wyjścia sieci jak opisano w Rozdziale 1.3. Następnie błędy $\boldsymbol{\varepsilon}^L$ na wyjściach sieci $\boldsymbol{o}$ są propagowane w tył od warstwy wyjściowej ($k = L$) do warstwy wejściowej ($k = 1$), używając zależności (2.2) oraz (2.3). Spropagowane błędy δ_i^k służą do korekcji wag synaptycznych $w_{i,j}^k$ neuronów N_i^k według reguły (2.4). Po korekcji wag sieć neuronowa jest lepiej przystosowana do rozpoznawania wzoru uczącego p.

2.2 SCHEMAT I SPOSÓB DZIAŁANIA ALGORYTMU EBP

Do określenia błędu sztucznej sieci neuronowej dla ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$ może zostać użyta miara

$$Q = \frac{1}{2}\sum_{p=1}^{|P|}\sum_{i=1}^{N_L} {\varepsilon_{p,i}^L}^2 = \frac{1}{2}\sum_{p=1}^{|P|}\sum_{i=1}^{N_L}\left(d_{p,i} - o_{p,i}\right)^2 \tag{2.5}$$

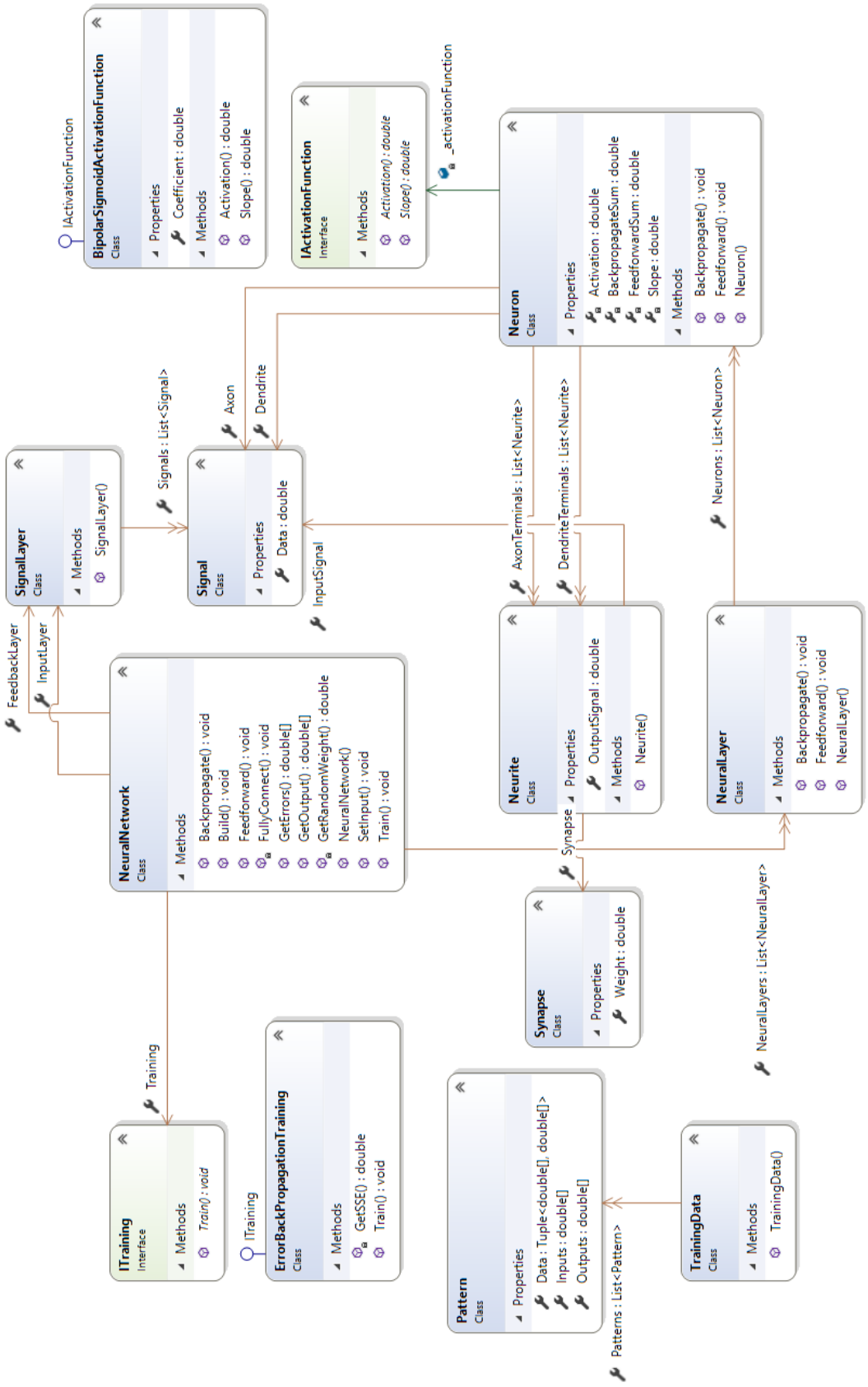
Organizacja procesu uczenia sztucznej sieci neuronowej ciągiem wzorów uczących P z wykorzystaniem algorytmu EBP została przedstawiony na Listingu 2. Po stworzeniu sztucznej sieci neuronowej o losowych wagach synaptycznych obliczany jest błąd Q (2.5) dla ciągu wzorów uczących P i porównywany z maksymalnym dopuszczalnym błędem Q_{max}. Dopóki błąd Q jest większy od błędu Q_{max}, algorytm iteruje przez wszystkie elementy ciągu uczącego P i koryguje wagi synaptyczne, korzystając ze wzorów: (2.1), (2.2), (2.3), (2.4). Gdy błąd Q neuronu stanie się mniejszy lub równy maksymalnemu dopuszczalnemu błędowi Q_{max}, algorytm kończy działanie, sieć neuronowa jest nauczona rozpoznawania elementów ciągu wzorów uczących P. Sposób uczenia sieci neuronowej, w którym korekcję wag synaptycznych przeprowadza się każdorazowo po przetworzeniu wzoru uczącego $(\boldsymbol{x}_p, \boldsymbol{d}_p)$, nosi nazwę **przyrostowego uaktualniania wag** (metody nadążnej) (ang. *on-line*).

Listing 2: Uczenie sieci neuronowej z wykorzystaniem algorytmu EBP

```
    Input:  P = {(x_1, d_1), …, (x_|P|, d_|P|)}   – zestaw uczący
            Q_max – maksymalny dopuszczalny błąd sieci dla zestawu uczącego P
 1  begin
 2      Stwórz sieć z losowymi wagami synaptycznymi
 3      Q ← błąd dla zestawu uczącego P
 4      while Q > Q_max do
 5          for p in 1, …, |P| do
 6              Koryguj wagi według algorytmu EBP wykorzystując wzór uczący (x_p, d_p)
10          end
11          Q ← błąd dla zestawu uczącego P
12      end
13  end
```

2.3 IMPLEMENTACJA ALGORYTMU EBP W JĘZYKU C#

Implementacja, opisanego w Rozdziale 2.1 oraz 2.2 , algorytmu wstecznej propagacji błędów została przedstawiona w postaci diagramu klas na Rysunku 2.1. Rozwiązanie rozszerza realizację sieci z Rozdziału 1.6 o mechanizmy niezbędne do uczenia algorytmem EBP. Klasy NeuralNetwork, NeuralLayer oraz Neuron uzyskały nową metodę Backpropagate umożliwiającą wsteczną propagację błędów, od warstwy wyjściowej do wejściowej sieci. Połączenia, którymi wykonywana jest wsteczna propagacja błędów, zostały zrealizowane z wykorzystaniem istniejących już klas Signal, Neuron oraz Synapse. Błędy sieci są wprowadzane do warstwy wyjściowej poprzez obiekt FeedbackLayer klasy SignalLayer, analogicznie do sposobu realizacji warstwy sygnałów wejściowych InputLayer, opisanego w Rozdziale 1.6. Algorytm uczenia sieci określa przypisana jej implementacja interfejsu ITraining, realizując wzorzec Strategy (Gamma, Helm Johnson, Vlissides, 1993). Klasa ErrorBackPropagationTraining implementująca interfejs ITraining, definiuje sposób działania algorytmu EBP. Wykorzystywany w procesie uczenia ciąg wzorów uczących reprezentuje klasa TrainingData, będąca kontenerem wzorów uczących – obiektów klasy Pattern.



Rysunek 2.1 Diagram klas przedstawiający implementację algorytmu EBP w języku C# dla jednokierunkowej sieci wielowarstwowej

Rozdział 3

ALGORYTM LEVENBERGA-MARQUARDTA

Kolejne modyfikacje algorytmu wstecznej propagacji błędów, m.in. algorytm przyspieszonej wstecznej propagacji (ang. *Quick Propagation Training* – Quickprop) (Fahlman, 1988) czy odporny algorytm wstecznej propagacji błędów (ang. *Resilent Backpropagation Training* – Rprop) (Riedmiller, Braun, 1992), w nieznacznym stopniu przyczyniły się do poprawy tempa jego zbieżności i odporności na występowanie minimów lokalnych miary błędu sieci. Istotnym dokonaniem w rozwoju uczenia nadzorowanego, pozwalającym zerwać z wadami algorytmu EBP, była adaptacja **algorytmu optymalizacji nieliniowej Levenberga-Marquardta** (ang. *Levenberg-Marquardt Training* – LM), opisanego w pracach: (Levenberg, 1944), (Marquardt, 1963), na cel uczenia sztucznych sieci neuronowych, przedstawiona w pracy (Hagan, Menhaj, 1994).

Algorytm Levenberga-Marquardta, jako **algorytm uczący rzędu drugiego** (Załącznik D), koryguje wagi synaptyczne sieci wykorzystując wektor gradientu oraz macierz Hessego, tj. odpowiednio wektor pierwszych pochodnych oraz macierz drugich pochodnych miary błędu sieci po zawartych w niej wagach synaptycznych. Algorytm LM proponuje aproksymację wektora gradientu oraz macierzy Hessego na podstawie macierzy Jacobiego i wektora błędów sieci, z wykorzystaniem której koryguje wagi synaptyczne sieci.

W niniejszym rozdziale przedstawiono regułę korekcji wag w algorytmie LM oraz schemat i sposób jego działania dla jednokierunkowej sieci wielowarstwowej z Rozdziału 1.3. Ponadto opisano praktyczną implementację algorytmu LM w języku C# do uczenia sieci zrealizowanej w Rozdziale 1.6.

3.1 KOREKCJA WAG W ALGORYTMIE LM

Algorytm Levenberga–Marquardta , wyprowadzony w Załączniku E, prowadzący do korekcji wag sztucznej sieci neuronowej zaprezentowanej w Rozdziale 1.3, przyjmuje następującą formę:

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial \varepsilon_{1,1}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{1,1}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{1,1}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \ddots & \vdots & \iddots & \cdots \\ \frac{\partial \varepsilon_{1,N_L}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{1,N_L}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{1,N_L}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_{|P|,1}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{|P|,1}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{|P|,1}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_{|P|,N_L}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{|P|,N_L}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{|P|,N_L}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \end{bmatrix} \tag{3.1}$$

$$\boldsymbol{e} = \left[\varepsilon_{1,1}^{L}, \ldots, \varepsilon_{1,N_L}^{L}, \ldots, \varepsilon_{p,m}^{L}, \ldots, \varepsilon_{|P|,1}^{L}, \ldots, \varepsilon_{|P|,N_L}^{L}\right]^{T} \tag{3.2}$$

$$\boldsymbol{w}^{*} = \boldsymbol{w} - [\boldsymbol{Q} + \mu \boldsymbol{I}]^{-1}\boldsymbol{g} = \boldsymbol{w} - [\boldsymbol{J}^{T}\boldsymbol{J} + \mu \boldsymbol{I}]^{-1}\boldsymbol{J}^{T}\boldsymbol{e} \tag{3.3}$$

gdzie $\mu > 0$ to współczynnik kombinacji, $\boldsymbol{Q}$ to pseudohesjan, $\boldsymbol{g}$ to wektor gradientu, $\boldsymbol{I}$ to macierz jednostkowa, $\boldsymbol{J}$ to macierz Jacobiego, $\boldsymbol{e}$ to wektor błędów sieci, a $\boldsymbol{w}^{*}$ jest skorygowanym wektorem wag synaptycznych $\boldsymbol{w}$. Parametr μ dobiera się w zależności od zmian błędu sieci w wyniku korekcji wag (3.3). Przyjmuje on duże wartości na starcie algorytmu, w miarę zbliżania się do rozwiązania optymalnego jego wartość maleje aż do wartości zerowej.

Proces korekcji wag sztucznej sieci neuronowej z wykorzystaniem algorytmu LM na podstawie ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$ został przedstawiony na Listingu 1. Propagując elementy ciągu P w przód obliczany jest błąd sieci Q z wykorzystaniem, zaproponowanej w Rozdziale 2.2, miary błędu (2.5). Następnie wyznacza się elementy macierzy Jacobiego $\boldsymbol{J}$ (3.1) oraz wektora błędów $\boldsymbol{e}$ (3.2), aby skorzystać z nich przy kolejnych próbach n, $n = 1, \ldots, n_{max}$, optymalnej korekcji wektora wag

synaptycznych **w**. Podczas każdej próby *n*, według reguły (3.3), obliczany jest skorygowany wektor wag $\boldsymbol{w}^*$ i mierzony błąd sieci Q^* wykorzystującej nowy wektor wag. Jeśli błąd Q^* jest mniejszy lub równy błędowi *Q*, próbę *n* uznaje się za udaną, współczynnik μ jest zmniejszany *f* razy, nie jest podejmowana kolejna próba *n* + 1. Jeśli jednak błąd Q^* okazał się większy od błędu *Q*, próbę *n* uznaje się za nieudaną, a współczynnik μ jest zwiększany *f* razy, aby posłużyć do obliczenia skorygowanego wektora wag $\boldsymbol{w}^*$ w kolejnej próbie *n* + 1. Po udanej próbie *n* lub nieudanych n_{max} próbach skorygowany wektor wag $\boldsymbol{w}^*$ staje się nowym wektorem wag synaptycznych ***w***. Po korekcji wag sieć neuronowa jest lepiej przystosowana do rozpoznawania ciągu wzorów uczących *P*.

Listing 1: Korekcja wag w algorytmie LM

```
    Input:   P = {(x1, d1), ..., (x|P|, d|P|)}   – zestaw uczący
 1  begin
 2     Q ← błąd dla zestawu uczącego P
 3     Oblicz macierz Jacobiego J oraz wektor błędów e
 4     for n in 1, ..., nmax do
 5        w* ← w - [J^T J + μI]^-1 · J^T e
 6        Q* ← błąd dla zestawu uczącego P po korekcji wag
 7        if Q* ≤ Q then
 8           μ ← μ / f
 9           break
10        else
11           μ ← f · μ
12     end
13     w ← w*
14  end
```

3.2 OBLICZANIE MACIERZY JACOBIEGO W ALGORYTMIE LM

Do korekcji wag w algorytmie Levenberga-Marquardta według reguły (3.3) konieczne jest wcześniejsze wyznaczenie wartości macierzy Jacobiego ***J*** (3.1) oraz wektora błędów ***e*** (3.2). O ile znalezienie wektora błędów ***e*** po przetworzeniu przez sieć sygnałów wejściowych metodą propagacji w przód, opisaną w Rozdziale 1.3, korzystając z definicji błędu (2.2), jest zadaniem trywialnym, o tyle wyznaczenie składników macierzy Jacobiego ***J***, tj. pochodnych cząstkowych pierwszego rzędu błędów (3.2) tworzących miarę błędu *Q* (2.5), wymaga narzędzia matematycznego pozwalającego na znalezienie zależności między wybranym błędem na wyjściu sieci $\varepsilon_{p,m}{}^{L}$ a daną wagą synaptyczną $w_{i,j}{}^{k}$ neuronu N_i^k.

Wyznaczanie jawnej formy macierzy Jacobiego metodą propagacji w tył

Algorytm wyznaczania jawnej formy macierzy Jacobiego metodą propagacji w tył, wyprowadzony w Załączniku F, przyjmuje następującą formę:

$$y_i^k = f\left(net_i^k\right), \qquad net_i^k = \sum_{j=0}^{N_{k-1}} x_j^k \, w_{i,j}^k \tag{3.4}$$

$$\hat{\varepsilon}_{m,i}^k = \begin{cases} 1, & k = L \wedge i = m \\ 0, & k = L \wedge i \neq m \\ \sum_{j=1}^{N_{k+1}} \hat{\delta}_{m,j}^{k+1} \, w_{j,i}^{k+1}, & k = 1, 2, \ldots, L-1 \end{cases} \tag{3.5}$$

$$\hat{\delta}_{m,i}^k = f'\left(net_i^k\right) \hat{\varepsilon}_{m,i}^k \tag{3.6}$$

$$\frac{\partial \varepsilon_{p,m}^L}{\partial \, w_{i,j}^k} = -\hat{\delta}_{m,i}^k x_j^k \tag{3.7}$$

Proces wyznaczania jawnej formy macierzy Jacobiego metodą propagacji w tył na podstawie ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$ został przedstawiony na Listingu 2. W kolejnych krokach $p = 1, \ldots, |P|$, wektor sygnałów uczących $\boldsymbol{x}_p$ zostaje wprowadzony na wejście sieci **u** i spropagowany w przód na wyjścia sieci jak opisano w Rozdziale 1.3. Następnie dla każdego z błędów sieci $\varepsilon_{p,m}{}^L$, $m = 1, \ldots, N_L$, każdej warstwy k, $k = L, \ldots, 1$ oraz każdego neuronu N_i^k, $i = 1, \ldots, N_k$, obliczane są wartości (3.5) oraz (3.6). Iterując przez kolejne połączenia synaptyczne $w_{i,j}{}^k$, $j = 0, \ldots, N_{k-1}$, neuronu N_i^k, z wykorzystaniem zależności (3.7) otrzymuje się elementy macierzy Jacobiego $\boldsymbol{J}[\partial \varepsilon_{p,m}{}^L / \partial \, w_{i,j}{}^k]$.

Listing 2: Wyznaczanie jawnej formy macierzy Jacobiego metodą propagacji w tył

```
    Input:  P = {(x_1, d_1), ..., (x_|P|, d_|P|)}   – zestaw uczący
    Result: J – macierz Jacobiego
 1  begin
 2      for p in 1, ..., |P| do
 3          u ← x_p
 4          Propaguj sygnały w przód
 6          for m in 1, ..., N_L do
 7              for k in L, ..., 1 do
 8                  for i in 1, ..., N_k do
 9                      if k = L then
10                          if i = m then
11                              ε̂_{m,i}^k ← 1
12                          else
13                              ε̂_{m,i}^k ← 0
14                          end
15                      else
16                          ε̂_{m,i}^k ← Σ_{j=1}^{N_{k+1}} δ̂_{m,j}^{k+1} · w_{j,i}^{k+1}
17                      end
18                      δ̂_{m,i}^k ← f'(net_i^k) · ε̂_{m,i}^k
19                      for j in 0, ..., N_{k-1} do
20                          J[∂ε_{p,m}^L/∂w_{i,j}^k] ← - δ̂_{m,i}^k · x_j^k
21                      end
22                  end
23              end
24          end
25      end
26  end
```

Wyznaczanie niejawnej formy macierzy Jacobiego metodą propagacji w tył

Algorytm wyznaczania niejawnej formy macierzy Jacobiego metodą propagacji w tył, wyprowadzony w Załączniku G, korzystając z wiersza macierzy Jacobiego

$$\boldsymbol{j}_{p,m} = \left[\frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{1,0}^{1}}, \ldots, \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k}}, \ldots, \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{N_L,N_{L-1}}^{L}}\right] \tag{3.8}$$

powstałego z wykorzystaniem zależności (3.7) po przetworzeniu (3.4) przez sieć wzoru uczącego $p = (\boldsymbol{x}_p, \boldsymbol{d}_p)$ i spropagowaniu w tył (3.6) błędów (3.5) dla wyjścia o_m, wyznacza wartość pseudohesjanu $\boldsymbol{Q}$ oraz wektora gradientu $\boldsymbol{g}$ w zależności (3.3) jako

$$\boldsymbol{Q} = \sum_{p=1}^{|P|} \sum_{m=1}^{N_L} q_{p,m} \tag{3.9}$$

$$\boldsymbol{g} = \sum_{p=1}^{P} \sum_{m=1}^{N_L} \eta_{p,m} \tag{3.10}$$

gdzie $\boldsymbol{q}_{p,m}$ to podmacierz pseudohesjanu, a $\boldsymbol{\eta}_{p.m}$ to podwektor wektora gradientu, określone jako

$$\boldsymbol{q}_{p,m} = {\boldsymbol{j}_{p,m}}^T \boldsymbol{j}_{p,m} = \\ = \begin{bmatrix} \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{1,0}^1 \partial w_{1,0}^1} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{1,0}^1 \partial w_{i,j}^k} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{1,0}^1 \partial w_{N_L,N_{L-1}}^L} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{i,j}^k \partial w_{1,0}^1} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{i,j}^k \partial w_{i,j}^k} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{i,j}^k \partial w_{N_L,N_{L-1}}^L} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{N_L,N_{L-1}}^L \partial w_{1,0}^1} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{N_L,N_{L-1}}^L \partial w_{i,j}^k} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{N_L,N_{L-1}}^L \partial w_{N_L,N_{L-1}}^L} \end{bmatrix} \tag{3.11}$$

$$\boldsymbol{\eta}_{p,m} = {\boldsymbol{j}_{p,m}}^T \varepsilon_{p,m}^L = \begin{bmatrix} \frac{\partial \varepsilon_{p,m}^L}{\partial w_{1,0}^1} \varepsilon_{p,m}^L \\ \vdots \\ \frac{\partial \varepsilon_{p,m}^L}{\partial w_{i,j}^k} \varepsilon_{p,m}^L \\ \vdots \\ \frac{\partial \varepsilon_{p,m}^L}{\partial w_{N_L,N_{L-1}}^L} \varepsilon_{p,m}^L \end{bmatrix} \tag{3.12}$$

Proces wyznaczania niejawnej formy macierzy Jacobiego metodą propagacji w tył na podstawie ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$ został przedstawiony na Listingu 3. W kolejnych krokach $p = 1, \ldots, |P|$, wektor sygnałów uczących $\boldsymbol{x}_p$ zostaje wprowadzony na wejście sieci $\boldsymbol{u}$ i spropagowany w przód na wyjścia sieci jak opisano w Rozdziale 1.3. Następnie dla każdego z błędów sieci ${\varepsilon_{p,m}}^L$, $m = 1, \ldots, N_L$, obliczany jest wiersz macierzy Jacobiego $\boldsymbol{j}_{p,m}$, z wykorzystaniem którego wyznacza się kolejne składniki $\boldsymbol{q}_{p,m}$ (3.11) oraz $\boldsymbol{\eta}_{p,m}$ (3.12), dodawane odpowiednio do pseudohesjanu $\boldsymbol{Q}$ (3.9) oraz wektora gradientu $\boldsymbol{g}$ (3.10).

Listing 3: Wyznaczanie niejawnej formy macierzy Jacobiego metodą propagacji w tył

```
Input:  P = {(x1, d1), …, (x|P|, d|P|)}   – zestaw uczący
Result: Q – pseudohesjan
        g – wektor gradientu
1  begin
2    for p in 1, …, |P| do
3      u ← xp
4      Propaguj sygnały w przód
5      for m in 1, …, NL do
6        Oblicz wiersz macierzy Jacobiego jp,m
7        Q ← Q + jp,m^T · jp,m
8        g ← g + jp,m^T · εp,m^L
9      end
10   end
11 end
```

Aproksymacja jawnej formy macierzy Jacobiego metodą propagacji w przód

Algorytm aproksymacji jawnej formy macierzy Jacobiego metodą propagacji w przód, wyprowadzony w Załączniku H, przyjmuje następującą formę:

$$\frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k}} = -\frac{\partial \mathrm{o_{p,m}}}{\partial net_i^k} x_j^k \tag{3.13}$$

$$\frac{\partial \mathrm{o_{p,m}}}{\partial net_i^k} \approx \frac{\mathrm{o_{p,m}}\left(net_i^k + \mathrm{h}\right) - \mathrm{o_{p,m}}\left(net_i^k\right)}{\mathrm{h}} \tag{3.14}$$

gdzie h to wartość przesunięcia funkcji net_i^k użyta do aproksymacji wartości $\partial o_{p,m}/\partial net_i^k$ metodą różnic skończonych.

Proces aproksymacji jawnej formy macierzy Jacobiego metodą propagacji w przód na podstawie ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$ został przedstawiony na Listingu 4. W kolejnych krokach p, $p = 1, \ldots, |P|$, wektor sygnałów uczących $\boldsymbol{x}_p$ zostaje wprowadzony na wejście sieci **u** i spropagowany w przód na wyjścia sieci jak opisano w Rozdziale 1.3. Wektor wyjść sieci $\boldsymbol{o}_p$ jest zapisywany jako wektor $\boldsymbol{o}'$, po czym dla każdej warstwy k, $k = 1, \ldots, L$, oraz każdego neuronu N_i^k, $i = 1, \ldots, N_k$, funkcja net_i^k ulega przesunięciu o wartość h, a jej zmiana jest propagowana w przód od warstwy k na wyjścia sieci $\boldsymbol{o}_p$. Następnie dla każdego z wyjść $o_{p,m}$, $m = 1, \ldots, N_L$, obliczana jest wartość (3.14), która służy do aproksymacji (3.13) kolejnych elementów macierzy Jacobiego $\boldsymbol{J}[\partial \varepsilon_{p,m}{}^{L} / \partial\, w_{i,j}{}^{k}]$.

Po obliczeniu wszystkich (N_{k-1} x N_L) aproksymacji należy przywrócić pierwotną wartość funkcji net_i^k.

Listing 4: Aproksymacja jawnej formy macierzy Jacobiego metodą propagacji w przód

```
    Input:  P = {(x_1, d_1), ..., (x_|P|, d_|P|)}   – zestaw uczący
    Result: J  – macierz Jacobiego
 1  begin
 2    for p in 1, ..., |P| do
 3      u ← x_p
 4      Propaguj sygnały w przód
 5      o' ← o_p
 6      for k in L, ..., 1 do
 7        for i in 1, ..., N_k do
 8          net_i^k ← net_i^k + h
 9          Propaguj sygnały w przód od warstwy k
10          for m in 1, ...., N_L do
11            ∂o_p,m /∂net_i^k ← (o_p,m - o'_p,m)/h
12            for j in 0, ..., N_k-1 do
13              J[∂ε_p,m^L/∂w_i,j^k] ← - ∂o_p,m /∂net_i^k · x'_j^k
14            end
15          end
16          net_i^k ← net_i^k - h
17        end
18      end
19    end
20  end
```

3.3 SCHEMAT I SPOSÓB DZIAŁANIA ALGORYTMU LM

Do określenia błędu sztucznej sieci neuronowej dla ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$ może zostać użyta miara

$$Q = \frac{1}{2}\sum_{p=1}^{|P|}\sum_{i=1}^{N_L} {\varepsilon_{p,i}^L}^2 = \frac{1}{2}\sum_{p=1}^{|P|}\sum_{i=1}^{N_L}\left(d_{p,i} - o_{p,i}\right)^2 \tag{3.15}$$

Organizacja procesu uczenia sztucznej sieci neuronowej ciągiem wzorów uczących P z wykorzystaniem algorytmu LM została przedstawiony na Listingu 5. Po stworzeniu sztucznej sieci neuronowej o losowych wagach synaptycznych obliczany jest błąd Q (3.15) dla ciągu wzorów uczących P i porównywany z maksymalnym dopuszczalnym błędem Q_{max}. Dopóki błąd Q jest większy od błędu Q_{max}, algorytm koryguje wagi synaptyczne, korzystając ze wzorów: (3.1), (3.2), (3.3). Gdy błąd Q neuronu stanie się mniejszy lub równy maksymalnemu dopuszczalnemu błędowi Q_{max}, algorytm kończy działanie, sieć neuronowa

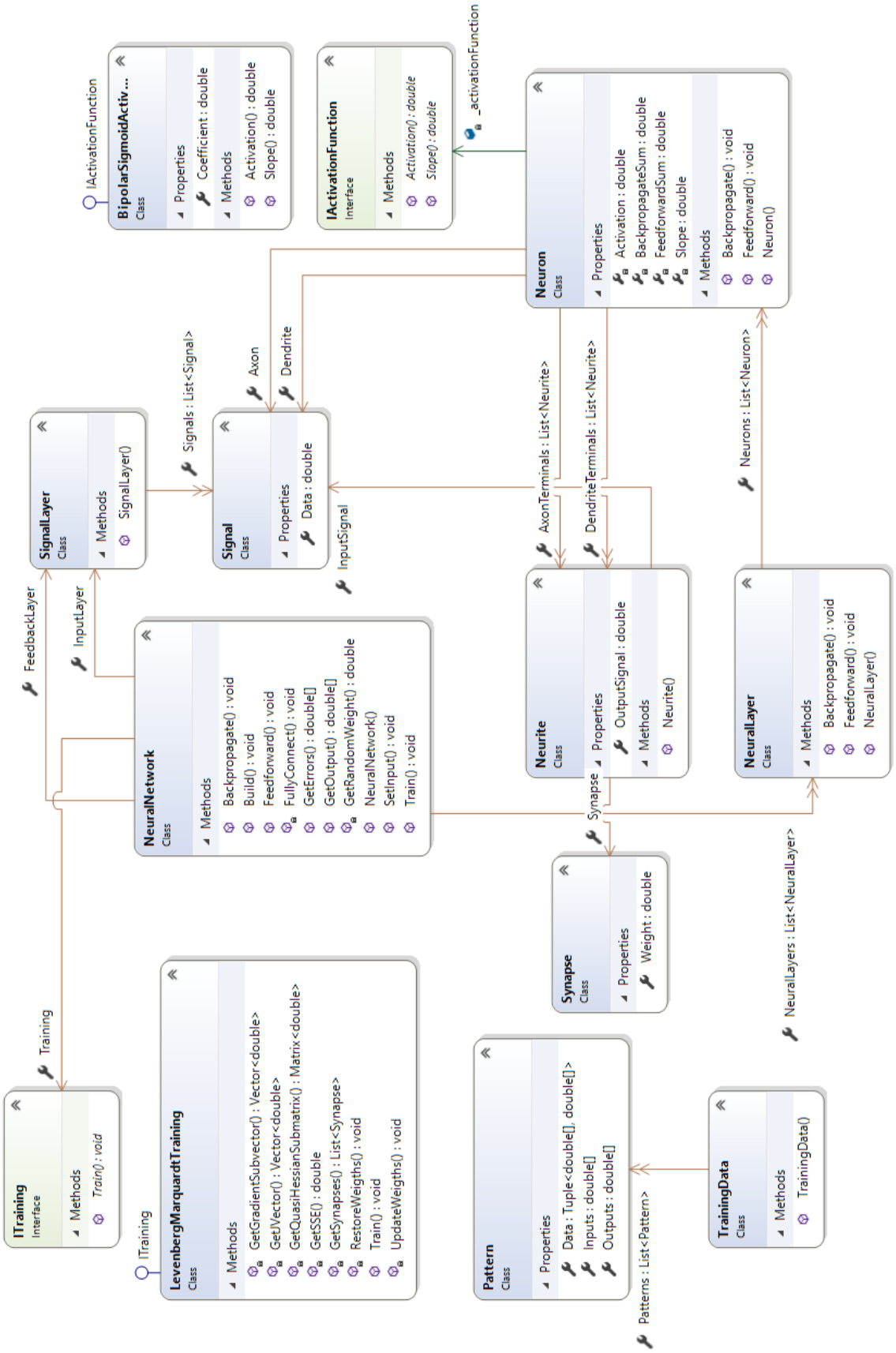
jest nauczona rozpoznawania elementów ciągu wzorów uczących P. Sposób uczenia sieci neuronowej, w którym korekcję wag synaptycznych przeprowadza się po przetworzeniu całego ciągu wzorów uczących *P*, nosi nazwę **kumulacyjnego uaktualniania wag** (metody nienadążnej) (ang. *off-line*).

Listing 5: Uczenie sieci neuronowej z wykorzystaniem algorytmu LM

```
Input:  P = {(x_1, d_1), ..., (x_|P|, d_|P|)}   – zestaw uczący
        Q_max – maksymalny dopuszczalny błąd sieci dla zestawu uczącego P
1   begin
2     Stwórz sieć z losowymi wagami synaptycznymi
3     Q ← błąd dla zestawu uczącego P
4     while Q > Q_max do
5       Koryguj wagi według algorytmu LM wykorzystując zestaw uczący P
11      Q ← błąd dla zestawu uczącego P
12    end
13  end
```

3.4 IMPLEMENTACJA ALGORYTMU LM W JĘZYKU C#

Implementacja, opisanego w Rozdziale 3.1, 3.2 oraz 3.3 , algorytmu Levenberga-Marquardta została przedstawiona w postaci diagramu klas na Rysunku 3.1. Rozwiązanie korzysta z wprowadzonego mechanizmu wstecznej propagacji błędów do uczenia algorytmem EBP opisanego w Rozdziale 2.3. Podczas wyznaczania macierzy Jacobiego, sygnały jednostkowe są wprowadzane do warstwy wyjściowej poprzez obiekt FeedbackLayer klasy SignalLayer, a następnie propagowane wstecz, umożliwiając obliczanie kolejnych elementów macierzy. Klasa LevenbergMarquardtTraining implementująca interfejs ITraining, definiuje sposób działania algorytmu LM, korzystający z wyznaczania niejawnej formy macierzy Jacobiego metodą propagacji w tył, opisanego w Rozdziale 3.2.



Rysunek 3.1 Diagram klas przedstawiający implementację algorytmu LM w języku C# dla jednokierunkowej sieci wielowarstwowej

Rozdział 4

KWANTOWO INSPIROWANE ALGORYTMY GENETYCZNE RZĘDU II

Postulaty wykorzystania pojęć Informatyki Kwantowej w dziedzinie Sztucznej Inteligencji (Menneer, Narayanan, 1995; Narayanan, Moore, 1996) doprowadziły do stworzenia nowej klasy w obszarze metod ewolucyjnych - **kwantowo inspirowanych metod ewolucyjnych** (ang. *Quantum Inspired Evolutionary Algorithms* – QIEA). **Kwantowo inspirowane algorytmy genetyczne** (ang. *Quantum-Inspired Genetic Algorithms* – QIGA) stanowią podklasę metod QIEA, a jednym z pierwszych jej reprezentantów został pierwotny algorytm QIGA1 (Han, Kim, 2000), w którym genotypy osobników kwantowych zamodelowano z użyciem kubitów. Obecnie klasa QIEA skupia również metody korzystające z innych reprezentacji rozwiązań niż ta zaproponowana w QIGA1. Przykładem takich algorytmów jest m.in. iQIEA (da Cruz et al., 2007) implementujący „kodowanie impulsowe".

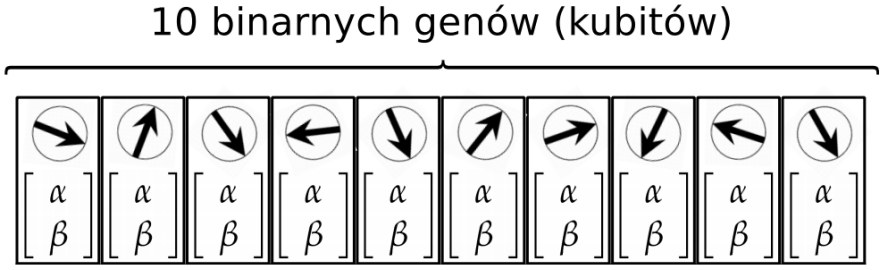
Znaczącym postępem w dziedzinie kwantowo inspirowanych algorytmów genetycznych jest sformułowanie pojęcia *rzędu* i *współczynników kwantowości algorytmu*, prowadzące do definicji nowej klasy **kwantowo inspirowanych algorytmów genetycznych wyższych rzędów** (Nowotniak, Kucharski, 2014). Czerpiąc inspiracje ze świata natury, kwantowo inspirowane algorytmy genetyczne wyższych rzędów przenoszą mechanizm zależności między grupami genów do dziedziny QIGA, ewoluując zaproponowaną w algorytmie QIGA1 reprezentację genotypów osobników kwantowych – z pierwotnej złożonej z niezależnych kubitów w nową w postaci splątanych wewnętrznie **rejestrów kwantowych**. Zmiana reprezentacji osobników pociąga za sobą stworzenie nowych operatorów genetycznych, pozostając jednocześnie przy klasycznym schemacie działania kwantowo inspirowanego algorytmu ewolucyjnego QIEA.

Nową przestrzeń algorytmów QIGA otwiera **kwantowo inspirowany algorytm rzędu II** (ang. *Order-2 Quantum Inspired Genetic Algorithm* – QIGA2) po raz pierwszy przedstawiony w pracy (Nowotniak, Kucharski, 2014). W niniejszym rozdziale opisano reprezentację rozwiązań, kwantowe operatory genetyczne oraz schemat i sposób działania algorytmu QIGA2. Ponadto przedstawiono praktyczną implementację algorytmu QIGA2 w języku C#.

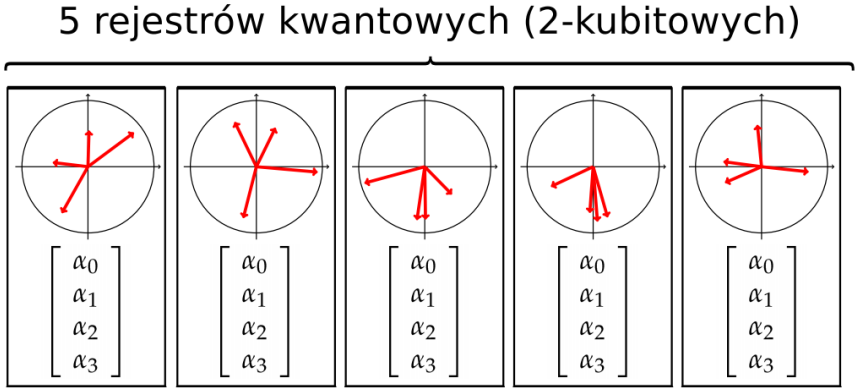
4.1 REPREZENTACJA ROZWIĄZAŃ W ALGORYTMIE QIGA2

Algorytmy QIGA1 oraz QIGA2 znacząco różnią się reprezentacją rozwiązań. Pierwotny algorytm QIGA1 wykorzystuje populację osobników reprezentowanych jako kwantowe genotypy, których poszczególne geny modelowane są za pomocą kubitów, tj. dwupoziomowych układów kwantowych, kodujących dwupunktowe rozkłady prawdopodobieństwa. Geny zakodowane w kubitach mogą przyjmować wartość 0 z prawdopodobieństwem $|\alpha|^2$ lub 1 z prawdopodobieństwem $|\beta|^2$, co ilustruje Rysunek 4.1.

Kwantowo inspirowany algorytm genetyczny rzędu II wykorzystuje nową reprezentację rozwiązań, modelując populację osobników jako kwantowe genotypy złożone z 2-kubitowych rejestrów kwantowych, tj. czteropoziomowych układów kwantowych, kodujących czteropunktowe rozkłady prawdopodobieństwa. W tym ujęciu sąsiadujące ze sobą geny są łączone w pary, a odpowiadające tym parom 2-kubitowe rejestry kwantowe kodują wartości binarne par. Para może przyjąć cztery wartości, tj. 00, 01, 10 i 11, z zapisanymi w rejestrze czterema wartościami prawdopodobieństw odpowiednio $|\alpha_0|^2$, $|\alpha_1|^2$, $|\alpha_2|^2$, $|\alpha_3|^2$. Zostało to zilustrowane na Rysunku 4.2.



Rysunek 4.1 Reprezentacja rozwiązań w algorytmach rzędu I złożona jest z kubitów – niezależnych binarnych genów kwantowych



Rysunek 4.2 Reprezentacja rozwiązań w algorytmach rzędu II złożona jest z 2-kubitowych rejestrów kwantowych – zależnych od siebie par binarnych genów kwantowych

Rysunek 4.1 oraz Rysunek 4.2 ilustruje reprezentację odpowiednio kubitów i rejestrów kwantowych jako wektorów na płaszczyźnie zespolonej. W implementacji algorytmu QIGA2, tak jak przy implementacji algorytmu QIGA1, wykorzystywana jest tylko część rzeczywista amplitud prawdopodobieństw α_0, …, α_3, a ich część urojona jest pomijana. Podczas inicjalizacji populacji bazowej $Q(0)$ cała przestrzeń rozwiązań X jest próbkowana z takim samym prawdopodobieństwem, co oznacza, że każdy gen kwantowy w algorytmie QIGA2 przyjmuje wartość $\boldsymbol{q}_{i,j} = [\ ½, ½, ½, ½\ ]^T$.

4.2 KWANTOWE OPERATORY GENETYCZNE W ALGORYTMIE QIGA2

Algorytm QIGA2 używa również innych operatorów genetycznych niż pierwotny algorytm QIGA1. W przeciwieństwie do algorytmu QIGA1, którego operatory są realizowane przez kwantowe bramki unitarne rozmiaru (2 × 2), operatory genetyczne algorytmu QIGA2 wykorzystują tylko podstawowe operacje arytmetyczne, przy czym obydwa algorytmy ograniczają się do uproszczenia amplitud prawdopodobieństw do wartości rzeczywistych.

Operacja pomiaru stanu 2-kubitowego rejestru kwantowego

$$q_{i,j} = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle = [\alpha_0, \alpha_1, \alpha_2, \alpha_3]^T \tag{5.1}$$

kodującego parę klasycznych genów binarnych została zaprezentowana na Listingu 1. Funkcja zwraca wartości binarne pary zakodowanej przez rejestr: 00, 01, 10, 11, z prawdopodobieństwem odpowiednio: $|\alpha_0|^2$, $|\alpha_1|^2$, $|\alpha_2|^2$, $|\alpha_3|^2$.

Listing 1: Pomiar stanu genów kwantowych w QIGA2

```
    Input:  g_{i,j} = [α0, α1, α2, α3]^T   – 2-kubitowy rejestr kwantowy
    Result: p ∈ {00, 01, 10, 11}           – para klasycznych genów binarnych
1   begin
2       r ← losowa liczba rzeczywista ∈ [0,1]
3       if r < |α0|^2 then
4           p ← 00
5       else if r < |α0|^2 + |α1|^2 then
6           p ← 01
7       else if r < |α0|^2 + |α1|^2 + |α2|^2 then
8           p ← 10
9       else
10          p ← 11
11      end
12  end
```

Listing 2: Aktualizacja stanu genów kwantowych w QIGA2

```
1  begin
2      for i in 0, …, |Q| - 1 do
3          for j in 0, …, N/2 do
4              sum ← 0
5              for amp in {0, 1, 2, 3} do
6                  if amp ≠ b_j then
7                      q_{i,j}[amp] ← μ · q_{i,j}[amp]
8                      sum ← sum + (q_{i,j}[amp])^2
9                  end
10             end
11             q_{i,j}[b_j] ← √(1 − sum)
12         end
13     end
14 end
```

Nowy operator genetyczny w algorytmie QIGA2 – aktualizacja stanu 2-kubitowego rejestru, został zaprezentowany na Listingu 2. Funkcja iteruje w głównej pętli przez wszystkie osobniki w populacji kwantowej q_0, …, $q_{|Q|-1}$ wykorzystując indeks i = 0, 1, …, $|Q|$ - 1. Wewnątrz głównej pętli, wykorzystując indeks j = 0, 1, …, N/2, iteruje przez kolejne pary genów $q_{i,j}$ danego osobnika q_i. Wewnątrz wymienionych pętli funkcja oblicza nowy stan $q_{i,j}$` pary genów kwantowych $q_{i,j}$ w następujący sposób: Dla amplitud o indeksach *amp* = 0, 1, 2, 3 reprezentujących w parze $q_{i,j}$ inną wartość niż para bitów b_j najlepszego znalezionego dotychczas osobnika b, następuje zmniejszenie amplitudy (kontrakcja amplitudy) według wzoru:

$$q'_{i,j}[amp] = \mu \cdot q_{i,j}[amp] \tag{5.2}$$

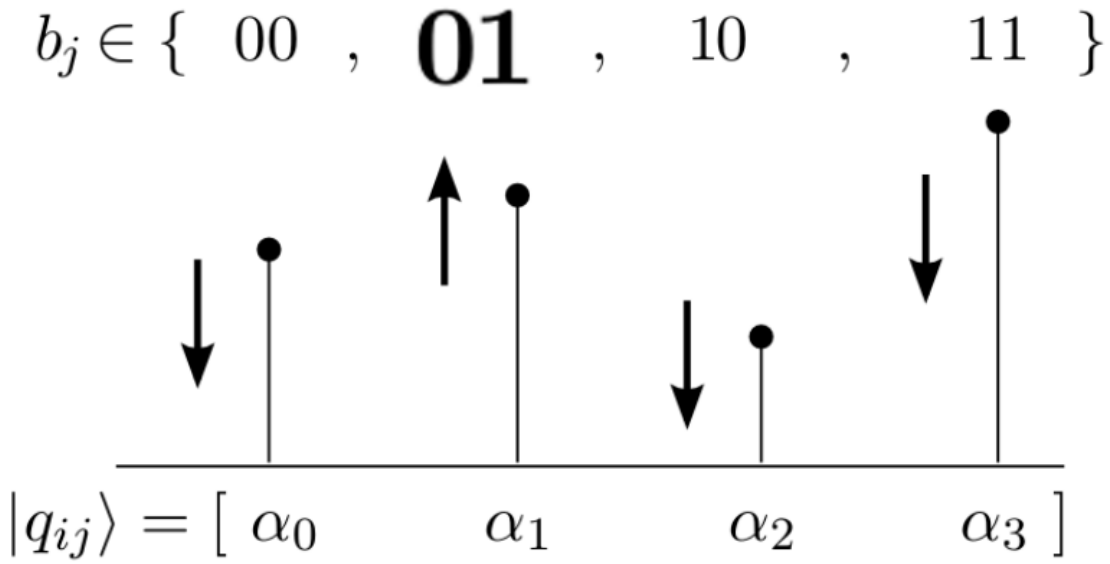
gdzie $\mu \in (0, 1)$ to współczynnik kontrakcji amplitud (parametr algorytmu). Natomiast amplituda odpowiadająca parze j bitów osobnika b jest aktualizowana tak, aby zachować warunek normalizacyjny wektora stanu, który dla rejestru 2-kubitowego przybiera formę:

$$\sum_{amp=0}^{3} \left|\alpha_{amp}\right|^2 = 1 \tag{5.3}$$

Na podstawie badań w pracy (Nowotniak, Kucharski, 2014) ustalono, że za optymalną wartość współczynnika kontrakcji można przyjąć $\mu \approx 0.99$.

Działanie ww. operatora ilustruje Rysunek 4.3. Pionowe słupki reprezentują wartości amplitud prawdopodobieństw $|\alpha_0|^2$, $|\alpha_1|^2$, $|\alpha_2|^2$, $|\alpha_3|^2$. Zwiększana jest tylko ta amplituda rejestru

kwantowego $q_{i,j}$ osobnika q_i, która odpowiada parze bitów b_j najlepszego znalezionego dotychczas osobnika b, pozostałe amplitudy w rejestrze ulegają kontrakcji.



Rysunek 4.3 Działanie operatora aktualizacji 2-kubitowego rejestru kwantowego. Amplituda α_1 w rejestrze $q_{i,j}$, odpowiadająca parze bitów b_j najlepszego znalezionego dotąd osobnika b, ulega zwiększeniu, dla pozostałych amplitud następuje kontrakcja.

4.3 SCHEMAT I SPOSÓB DZIAŁANIA ALGORYTMU QIGA2

Mimo odmiennej reprezentacji rozwiązań oraz nowych operatorów genetycznych kwantowo inspirowany algorytm genetyczny rzędu II pozostaje przy schemacie działania znanym z innych algorytmów ewolucyjnych QIEA. Pseudokod algorytmu QIGA2 został przedstawiony na Listingu 3.

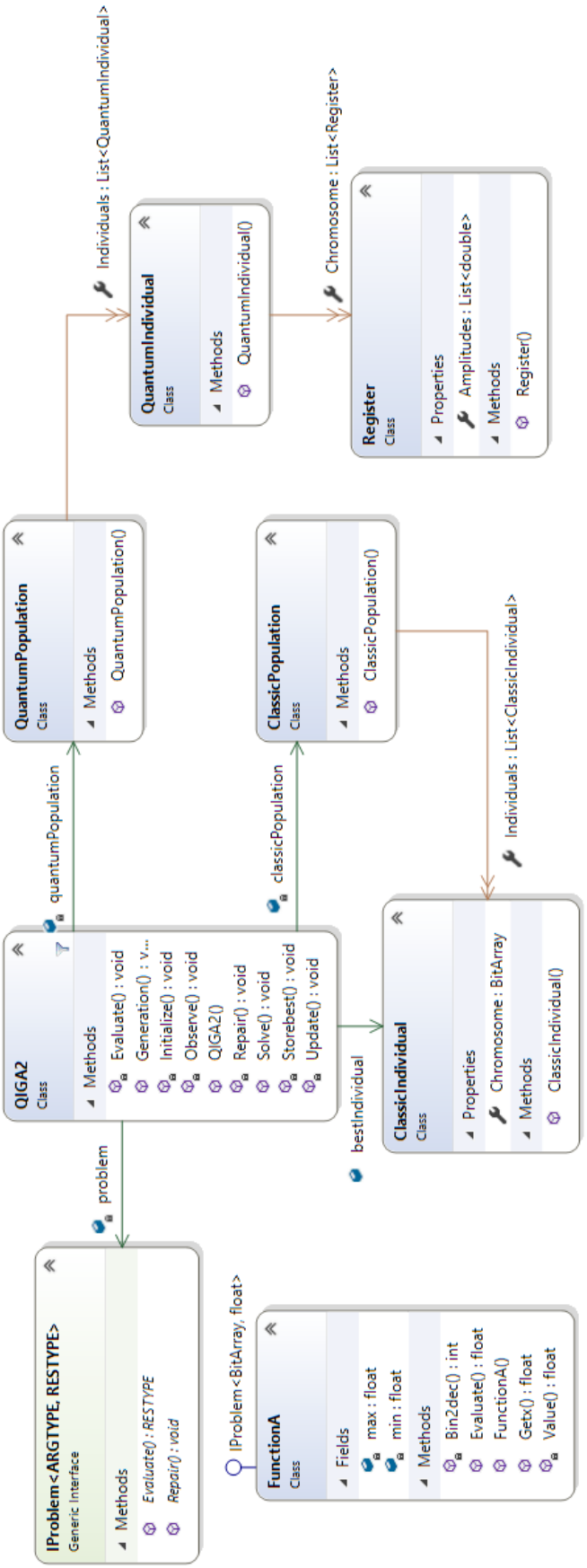
Listing 3: Pseudokod algorytmu QIGA2

```
1  begin
2      t ← 0
3      Inicjalizuj populację kwantową Q
4      while t ≤ t_max do
5          Utwórz populację P poprzez obserwację stanów Q
6          Oceń osobniki w populacji P
7          Wybierz najlepszego osobnika b z populacji P
8          Aktualizuj stan populacji Q
9          t ← t + 1
10     end
11 end
```

W początkowej fazie algorytmu QIGA2 zostaje zainicjalizowana populacja kwantowa *Q*. W kolejnych krokach $t = 0, \ldots, t_{max}$ populacja *Q* jest poddawana obserwacji prowadzącej do generacji pokolenia populacji klasycznej *P*. Osobniki klasyczne podlegają następnie ocenie przystosowania, a najlepszy z nich jest zapisywany jako *b*. Z wykorzystaniem zdefiniowanego w Rozdziale 4.2 operatora aktualizacji oraz zapisanego najlepszego osobnika *b* z bieżącego pokolenia klasycznego, generowane jest nowe pokolenie populacji kwantowej *Q*. Kolejne kroki *t* są wykonywane aż do spełnienia zadanego warunku stopu, np. stworzenia wyznaczonej liczby pokoleń.

4.4 IMPLEMENTACJA ALGORYTMU QIGA2 W JĘZYKU C#

Implementacja, opisanego w Rozdziale 4.1, 4.2 oraz 4.3 , algorytmu QIGA2 została przedstawiona w postaci diagramu klas na Rysunku 4.4. Klasa QIGA2 realizuje kwantowe operatory genetyczne oraz organizuje proces generacji i oceny kolejnych pokoleń. W każdej iteracji algorytmu uaktualnia ona populację kwantową w postaci obiektu klasy QuantumPopulation, komponującą osobników kwantowych – QuantumIndividual, złożonych z rejestrów kwantowych – Register. W wyniku pomiaru populacji kwantowej powstaje populacja klasyczna – ClassicPopulation, złożona z osobników klasycznych – ClassicIndividual, a z niej wybierany jest najlepszy osobnik – bestIndividual. Interfejs IProblem definiuje problem, który ma zostać rozwiązany z użyciem algorytmu QIGA2. Metoda Evaluate pozwala na ocenę przystosowania osobników klasycznych, a Repair – na dostosowanie wygenerowanych rozwiązań do dziedziny problemu, np. przy rozwiązywaniu problemu plecakowego. Klasa FunctionA implementuje interfejs IProblem, określając problem optymalizacji funkcji jednej zmiennej rzeczywistej.



Rysunek 4.4 Diagram klas przedstawiający implementację algorytmu QIGA2 w języku C#

Rozdział 5

EWOLUCJA SZTUCZNYCH SIECI NEURONOWYCH

Postulaty połączenia algorytmów genetycznych i sztucznych sieci neuronowych, przedstawione w pracy (Schaffer, Whitley, Eshelman, 1992), doprowadziły do stworzenia nowego podejścia do problemu projektowania sieci neuronowych – **neuroewolucji** (ang. *neuroevolution*). Jednym z pierwszych przykładów jej realizacji był algorytm ewolucji neuro-genetycznej (ang. *neuro-genetic evolution*) (Ronald, Schoenauer, 1994), pozwalający stworzyć sieć neuronową sterującą procesem lądowania modelu modułu księżycowego z dokładnością nieosiągalną metodami uczenia nadzorowanego.

Neuroewolucja wykorzystuje **metody ewolucyjne do poszukiwania optymalnych parametrów i topologii sieci oraz uczenia jej** bez korzystania z tradycyjnych algorytmów pierwszego i drugiego rzędu. Ponadto umożliwia wykorzystanie sieci neuronowych w problemach, w których uzyskanie informacji dotyczących gradientów jest trudne lub kosztowne. W tym celu adaptuje m.in. algorytmy genetyczne pozwalające na globalne przeszukiwanie przestrzeni wag i unikanie minimów lokalnych. W takim podejściu zaadaptowany algorytm genetyczny, korzystając ze zdefiniowanego wcześniej sposobu kodowania, tworzy kolejne pokolenia osobników reprezentujących sieci neuronowe, a następnie dokonuje oceny ich przystosowania według przyjętych kryteriów, aby przekazać pożądane cechy kolejnym pokoleniom. W ten sposób doprowadza do znalezienia sieci optymalnej, dobrze dostosowanej do postawionego przed nią problemu.

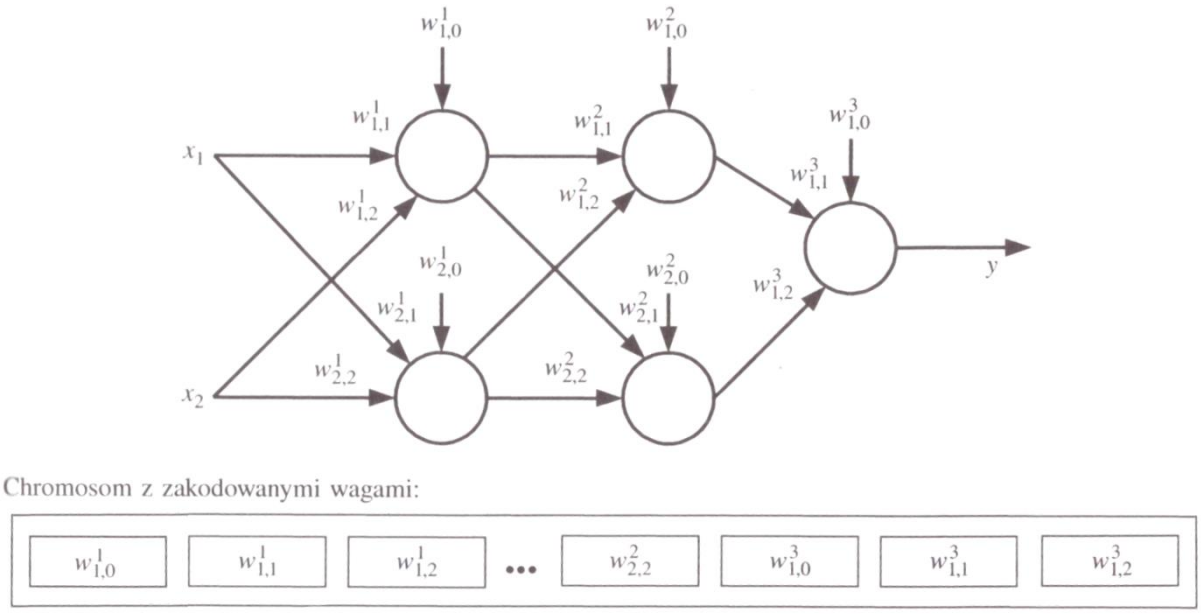
W niniejszym rozdziale przedstawiono sposoby kodowania i oceny przystosowania sztucznej sieci neuronowej oraz schemat i sposób ewolucji sieci z wykorzystaniem algorytmu QIGA2. Ponadto opisano praktyczną implementacją procesu ewolucji w języku C# do projektowania sieci zrealizowanej w Rozdziale 1.6.

5.1 KODOWANIE SIECI NEURONOWEJ

Wykorzystanie algorytmów genetycznych do budowy i uczenia sieci neuronowych wymaga zdefiniowania sposobu kodowania cech sieci w genotypach generowanych osobników. Wybór zestawu cech sieci optymalizowanych algorytmem genetycznym determinuje różne schematy ich kodowania, których przeglądu dokonano w pracy (Rutkowska, Piliński, Rutkowski, 1999).

Kodowanie wag sieci

W problemie uczenia sieci neuronowej algorytmem genetycznym, czyli określania wag połączeń synaptycznych w sieci o zadanej topologii, wagi mogą być kodowane w postaci ciągu binarnego, który będzie interpretowany jako wektor liczb rzeczywistych. Każdy osobnik populacji algorytmu genetycznego stanowi zbiór wszystkich wag sieci, co zostało zilustrowane na Rysunku 5.1. Kolejność wag w chromosomie jest dowolna, jednak nie może być zmieniana w kolejnych etapach algorytmu.

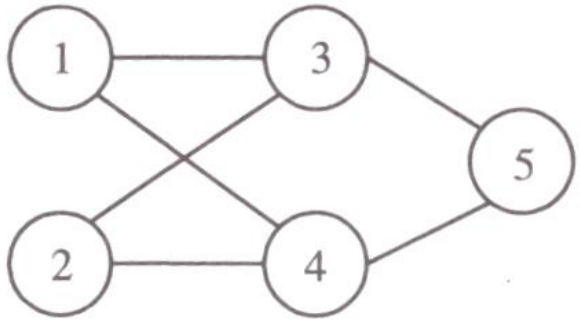


Rysunek 5.1 Kodowanie wag połączeń synaptycznych w sztucznej sieci neuronowej w postaci wektora liczb rzeczywistych

Kodowanie topologii sieci

Rozwiązując problem poszukiwania optymalnej topologii sieci neuronowej z użyciem algorytmu genetycznego, połączenia między neuronami mogą zostać zakodowane z użyciem macierzy połączeń, przestawionej na Rysunku 5.2. Macierz $\boldsymbol{C}$ o wymiarze (n x n), $\boldsymbol{C} = [c_{i,j}]_{n \times n}$

reprezentuje połączenia w sieci zbudowanej z n neuronów. Każdy element $c_{i,j}$ macierzy $\boldsymbol{C}$ opisuje połączenie z neuronu i do neuronu j, gdzie $c_{i,j} = 1$ oznacza występowanie połączenia, a $c_{i,j} = 0$ – jego brak. W chromosomach osobników generowanych przez algorytm genetyczny zaproponowana macierz połączeń jest reprezentowana przez ciąg binarny o długości n^2, stanowiący złożenie wierszy lub kolumn macierzy $\boldsymbol{C}$. Wadą takiego rozwiązania jest gwałtowny wzrost potrzebnej pamięci wraz ze zwiększaniem liczby neuronów w sieci, jednakże, nakładając pewne ograniczenia, adekwatne do rozpatrywanego problemu, można ten zapis skracać (Miller, Todd, Hadge, 1989).



Macierz połączeń:

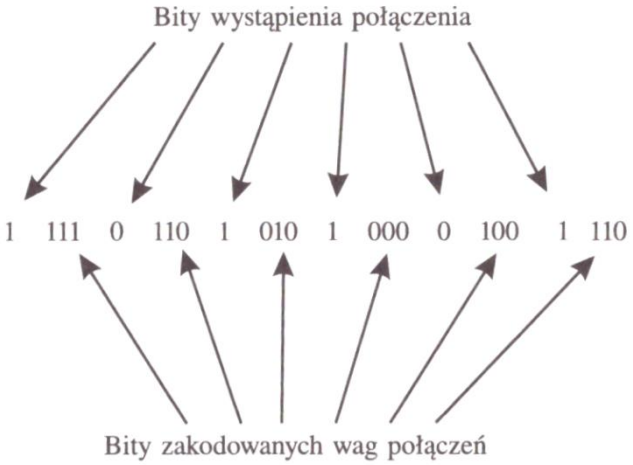
$$C = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Chromosom:
[0011000110000010000100000]

Rysunek 5.2 Kodowanie topologii sztucznej sieci neuronowej z użyciem macierzy połączeń

Kodowanie wag i topologii sieci

Jednoczesne kodowanie wag i topologii sieci, przedstawione na Rysunku 5.3, pozwala znaleźć sieć o optymalnej architekturze bez konieczności uczenia jej. Połączenia między neuronami, podobnie jak w reprezentacji macierzowej, są określone w chromosomach osobników za pomocą pojedynczych bitów. Oddzielone tymi bitami ciągi binarne kodują wagi kolejnych połączeń synaptycznych.



Rysunek 5.3 Jednoczesne kodowanie topologii i wag połączeń synaptycznych sztucznej sieci neuronowej

5.2 OCENA PRZYSTOSOWANIA SIECI NEURONOWEJ

Wybór najlepszego osobnika, którego cechy zostaną wzmocnione w kolejnym pokoleniu populacji algorytmu genetycznego, wymaga zdefiniowania sposobu oceny przystosowania dla problemu ewolucji sztucznych sieci neuronowych. Przebieg oceny osobników zależy od kodowanych przez nich cech sieci neuronowej.

Ocena przystosowania osobników kodujących wagi lub wagi i topologię sieci

Proces oceny przystosowania osobników kodujących wagi lub wagi i topologię sieci został przedstawiony na Listingu 1. Sieć neuronową, stworzoną poprzez dekodowanie osobnika *p*, ocenia się według przyjętego kryterium, którym może być m.in. miara błędu *Q* (3.15) dla ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$.

Listing 1: Ocena przystosowania osobników kodujących wagi lub wagi i topologię sieci

```
1 begin
2     Stwórz sieć neuronową dekodując osobnika p
3     Oceń przystosowanie stworzonej sieci według przyjętego kryterium
4 end
```

Ocena przystosowania osobników kodujących topologię sieci

Proces oceny przystosowania osobników kodujących topologię sieci został przedstawiony na Listingu 2. Sieć neuronową, stworzoną poprzez dekodowanie osobnika p, należy nauczyć rozpoznawania ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$ wybraną wcześniej

metodą uczenia. Następnie ocenia się ją według przyjętego kryterium, którym może być m.in. czas uczenia czy zdolność generalizacji.

Listing 2: Ocena przystosowania osobników kodujących wagi lub wagi i topologię sieci

```
1 begin
2     Stwórz sieć neuronową dekodując osobnika p
3     Naucz sieć rozpoznawania ciągu wzorów uczących P
4     Oceń przystosowanie stworzonej sieci według przyjętego kryterium
5 end
```

5.3 SCHEMAT I SPOSÓB EWOLUCJI SIECI Z WYKORZYSTANIEM ALGORYTMU QIGA2

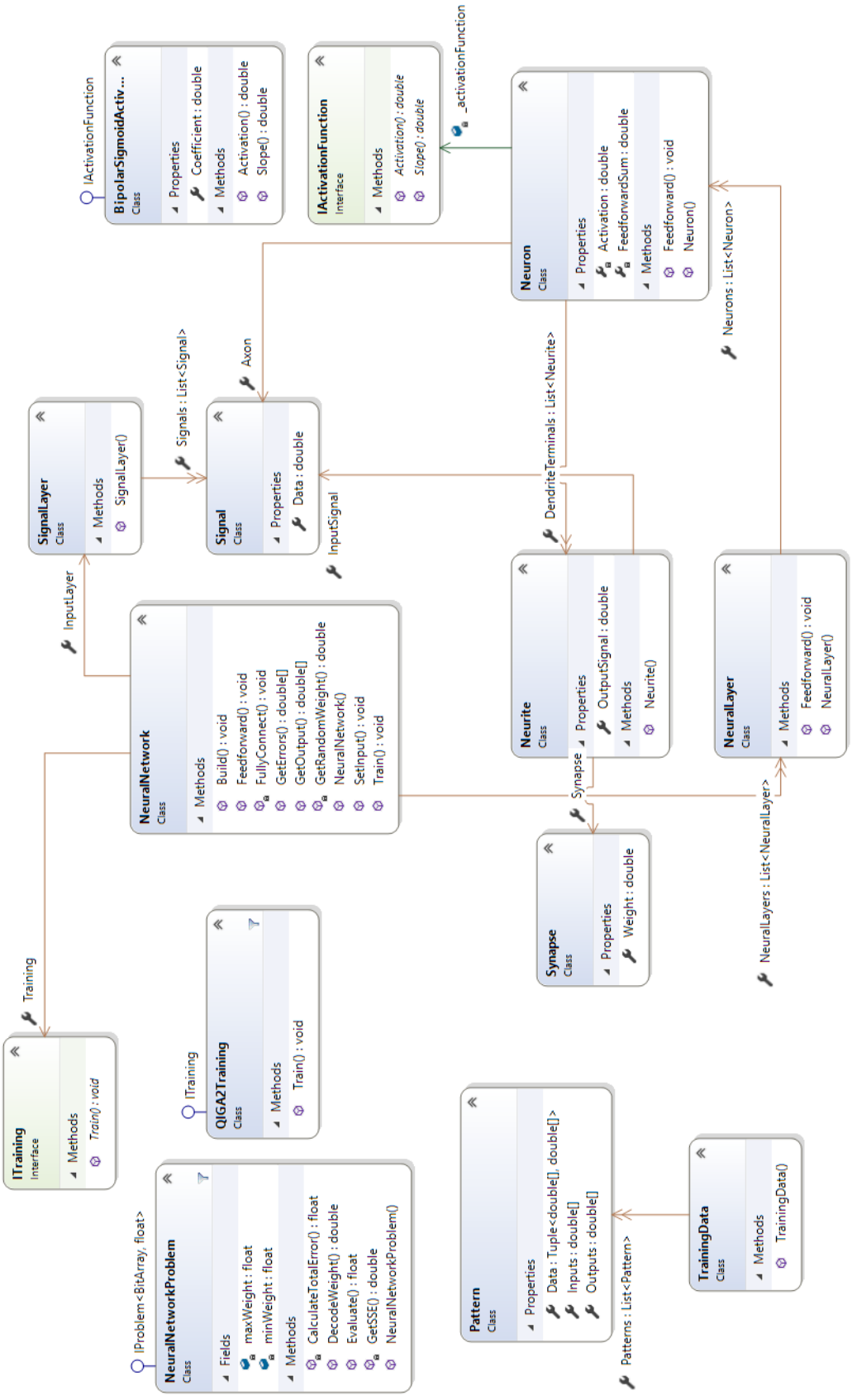
Organizacja procesu projektowania i uczenia sieci neuronowej z wykorzystaniem algorytmu QIGA2 została przedstawiona na Listingu 3. Pozostając przy schemacie działania algorytmu QIGA2, opisanym w Rozdziale 4.3, wykorzystuje kodowanie i ocenę przystosowania zaproponowaną w Rozdziałach 5.1 oraz 5.2. W początkowej fazie ewolucji zostaje zainicjalizowana populacja kwantowa Q. W kolejnych krokach $t = 0, \ldots, t_{max}$ populacja Q jest poddawana obserwacji prowadzącej do generacji pokolenia populacji klasycznej P. Następnie ze zdekodowanych osobników klasycznych tworzy się kolejne sieci neuronowe i poddaje się je ocenie przystosowania według wybranego kryterium, np. miary błędu Q (3.15) dla ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$. Osobnik reprezentujący sieć najlepiej przystosowaną jest zapisywany jako b. Z wykorzystaniem zdefiniowanego w Rozdziale 4.2 operatora aktualizacji oraz zapisanego najlepszego osobnika z bieżącego pokolenia klasycznego b, generowane jest nowe pokolenie populacji kwantowej Q. Kolejne kroki t są wykonywane aż do spełnienia zadanego warunku stopu, np. stworzenia wyznaczonej liczby pokoleń.

Listing 3: Ewolucja sieci z wykorzystaniem algorytmu QIGA2

```
1  begin
2      t ← 0
3      Inicjalizuj populację kwantową Q
4      while t ≤ t_max do
5          Utwórz populację P poprzez obserwację stanów Q
6          Stwórz sieci neuronowe dekodując osobniki w populacji P
7          Oceń przystosowanie stworzonych sieci według wybranego kryterium
8          Wybierz najlepszego osobnika b z populacji P
9          Aktualizuj stan populacji Q
10         t ← t + 1
11     end
12 end
```

5.4 IMPLEMENTACJA EWOLUCJI SIECI Z WYKORZYSTANIEM ALGORYTMU QIGA2 W JĘZYKU C#

Implementacja, opisanej w Rozdziale 5.3, ewolucji sztucznej sieci neuronowej z wykorzystaniem algorytmu QIGA2 została przedstawiona w postaci diagramu klas na Rysunku 5.4. Rozwiązanie rozszerza diagram klas z Rozdziału 1.6 o nową implementację interfejsu ITraining w postaci klasy QIGA2Training, organizującej proces ewolucji jednokierunkowej sieci wielowarstwowej z wykorzystaniem algorytmu QIGA2. Do realizacji tego procesu niezbędne było zdefiniowanie nowej specyfikacji interfejsu IProblem, opisanego w Rozdziale 4.4, zawartej w klasie NeuralNetworkProblem. Ta implementacja interfejsu dekoduje osobniki w populacji, tworząc kolejne sieci neuronowe, i ocenia ich przystosowanie, według sposobu opisanego w Rozdziale 5.1 oraz 5.2.



Rysunek 5.4 Diagram klas przedstawiający implementację ewolucji jednokierunkowej sieci wielowarstwowej neuronowej z wykorzystanie, algorytmu QIGA2 w języku C#

Rozdział 6

BADANIA DOŚWIADCZALNE

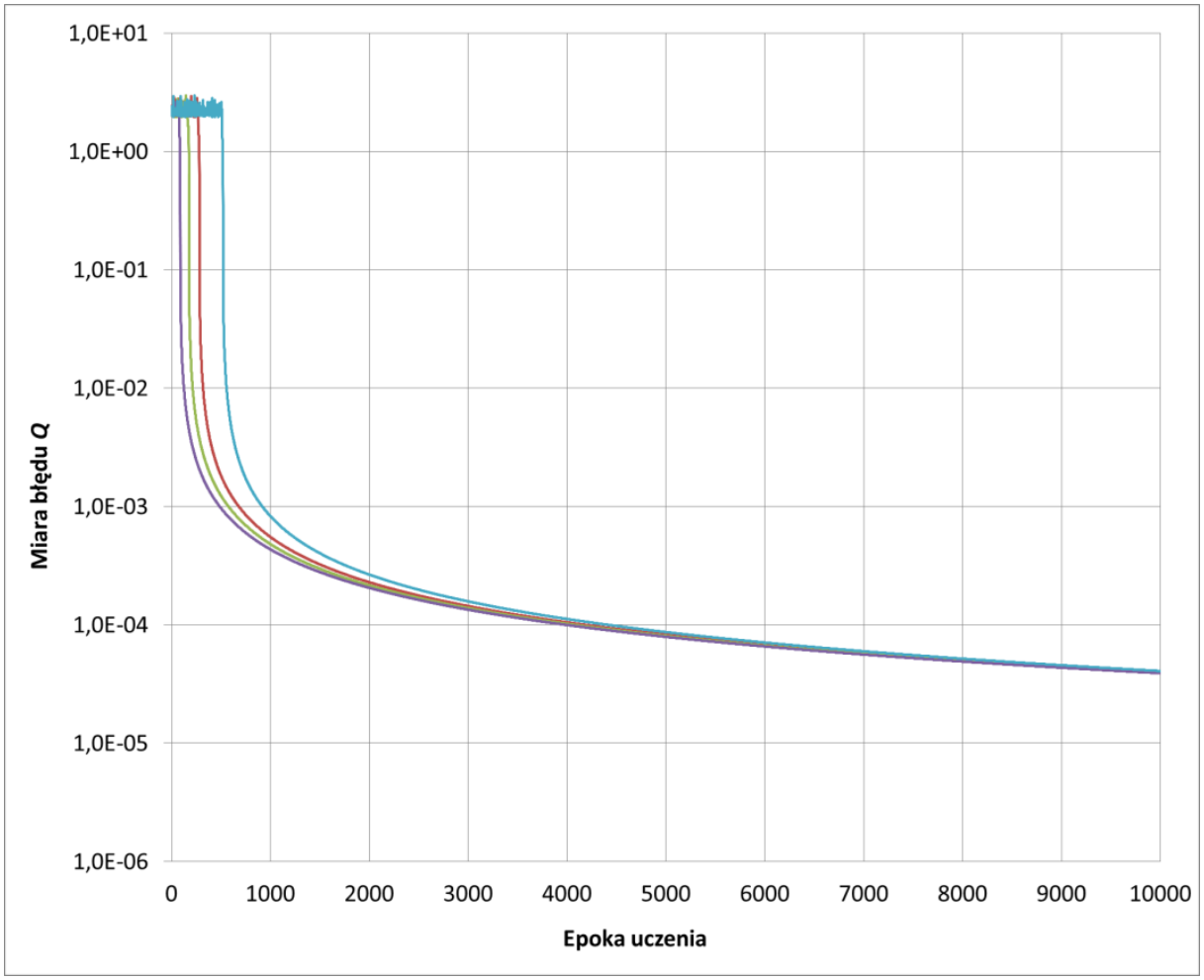
Dla zbadania efektywności projektowania sztucznych sieci neuronowych z wykorzystaniem kwantowo inspirowanych algorytmów genetycznych, przedstawionego w Rozdziale 5, przeprowadzono badania doświadczalne, pozwalające porównać przebieg procesu neuroewolucji z efektami działania algorytmów uczących pierwszego i drugiego rzędu.

6.1 PROCEDURA BADAWCZA

Badania przeprowadzone zostały według następującego schematu:

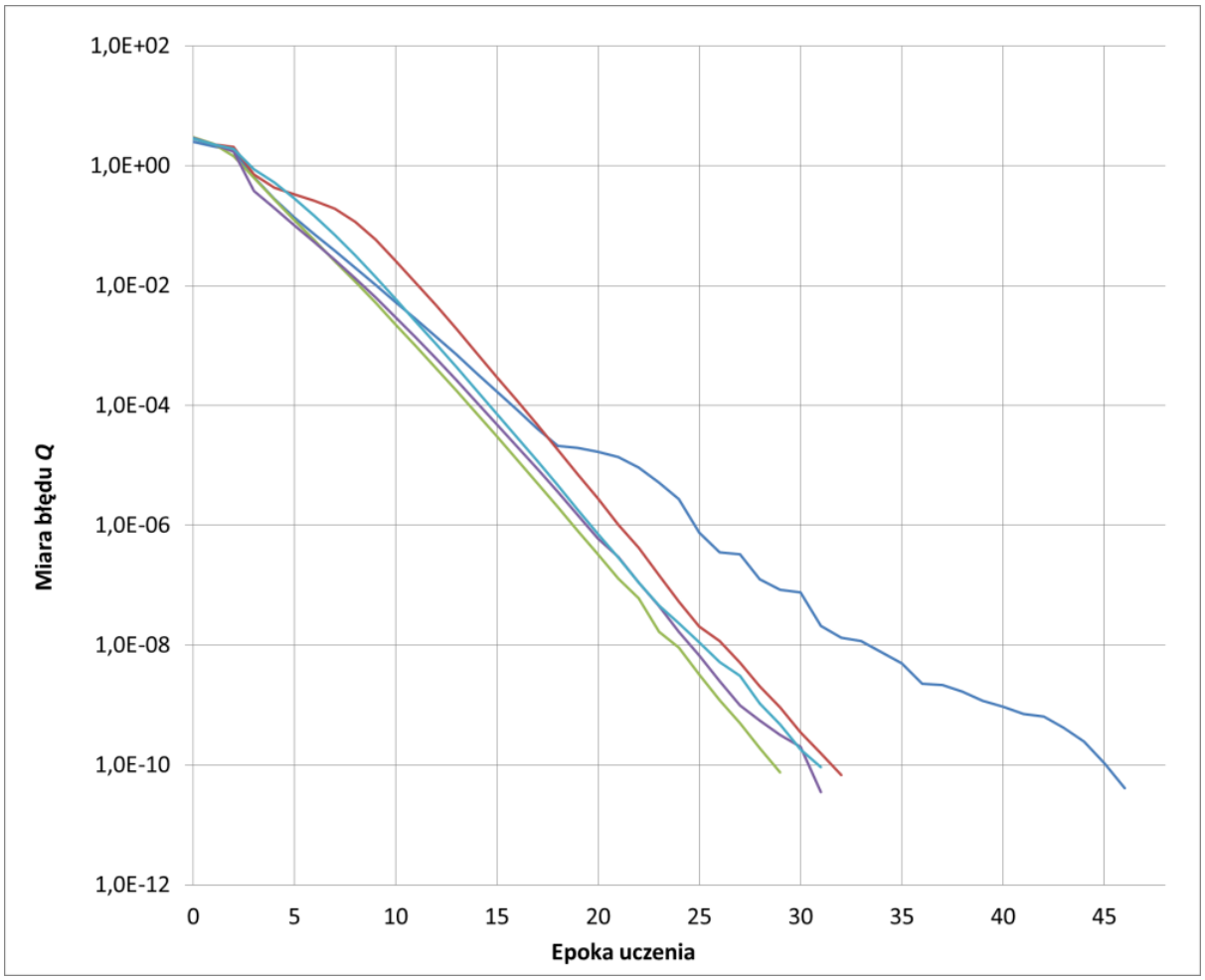
1) Przygotowano ciąg wzorów uczących (Tabela 1.1) dla problemu XOR, opisanego w Rozdziale 1.3.
2) Dla jednokierunkowej sieci wielowarstwowej o minimalnej architekturze pozwalającej na rozwiązanie problemu XOR (Rutkowski, 2009), tj. dwa neurony sigmoidalne bipolarne (β = 1) w warstwie wejściowej i jeden neuron sigmoidalny bipolarny (β = 1) w warstwie wyjściowej, z losowymi wagami połączeń synaptycznych z przedziału [-1, 1], przeprowadzono proces uczenia przygotowanego ciągu wzorów uczących z wykorzystaniem:
   a) algorytmu wstecznej propagacji błędów (η = 0,5) (Rysunek 6.1).
   b) algorytmu Levenberga-Marquardta (μ = 10, f = 10) (Rysunek 6.2).
   c) neuroewolucji z wykorzystaniem algorytmu QIGA2 ($|Q|$ = 20, μ = 0,99) oraz kodowania wag sieci opisanego w Rozdziale 5.1 (Rysunek 6.3).
3) Dla jednokierunkowej sieci wielowarstwowej o nadmiarowej architekturze pozwalającej na rozwiązanie problemu XOR, tj. pięć neuronów sigmoidalnych bipolarnych (β = 1) w warstwie wejściowej i jeden neuron sigmoidalny bipolarny (β = 1) w warstwie wyjściowej, przeprowadzono proces neuroewolucji z wykorzystaniem algorytmu QIGA2 ($|Q|$ = 100, μ = 0,99) oraz kodowania wag i topologii sieci, opisanego w Rozdziale 5.1, w celu uzyskania sieci rozpoznającej przygotowany ciąg wzorów uczących (Rysunek 6.4).
4) Proces uczenia lub neuroewolucji był realizowany do momentu znalezienia sieci, której miara błędu Q była mniejsza lub równa maksymalnemu dopuszczalnemu błędowi sieci $Q_{max} = 10^{-10}$.

6.2 WYNIKI BADAŃ DOŚWIADCZALNYCH



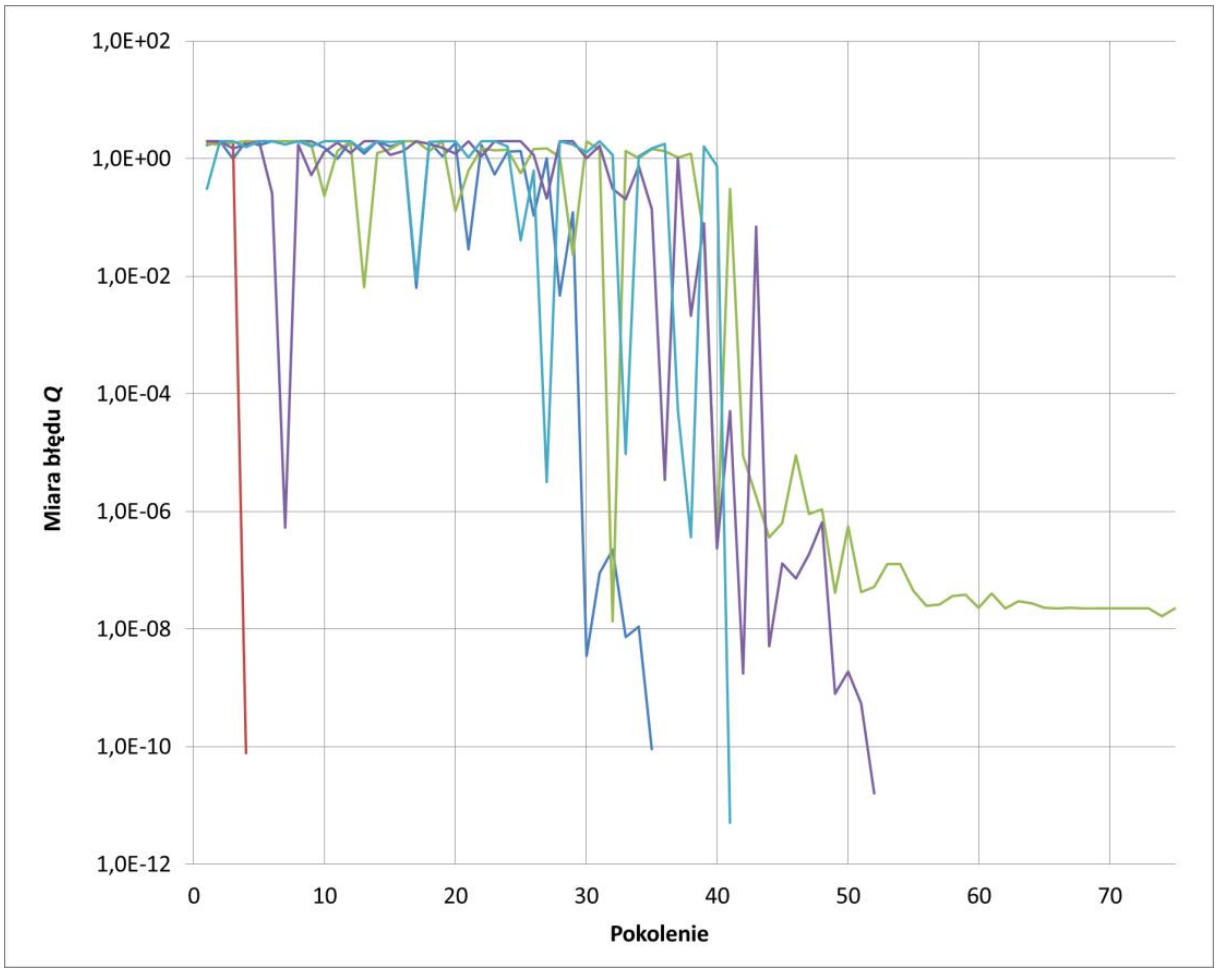
Rysunek 6.1 Efektywność algorytmu wstecznej propagacji błędów (η = 0,5) w poszukiwaniu rozwiązania problemu XOR. Kolory reprezentują niezależne przebiegi procesu uczenia.

Na Rysunku 6.1 przedstawiony został wykres, prezentujący efektywność algorytmu wstecznej propagacji błędów w poszukiwaniu rozwiązania problemu XOR dla jednokierunkowej sieci wielowarstwowej o minimalnej architekturze pozwalającej na jego rozwiązanie. Oś odciętych na wykresie przedstawia liczbę wykonanych epok uczenia sztucznej sieci neuronowej, rozumianych jako kompletne przetworzenie przez algorytm ciągu wzorów uczących. Na osi rzędnych odłożona została minimalizowana miara błędu Q sieci, osiągnięta na wybranym etapie uczenia. Kolory na Rysunku 6.1 reprezentują niezależne przebiegi procesu uczenia. Każdy z nich został przerwany po 10 000 epok uczenia ze względu na wolną zbieżność algorytmu EBP. W wyniku uczenia nie znaleziono sieci, której miara błędu Q byłaby mniejsza lub równa założonemu maksymalnemu dopuszczalnemu błędowi sieci $Q_{max} = 10^{-10}$.



Rysunek 6.2 Efektywność algorytmu Levenberga-Marquardta (μ = 10, f = 10) w poszukiwaniu rozwiązania problemu XOR. Kolory reprezentują niezależne przebiegi procesu uczenia.

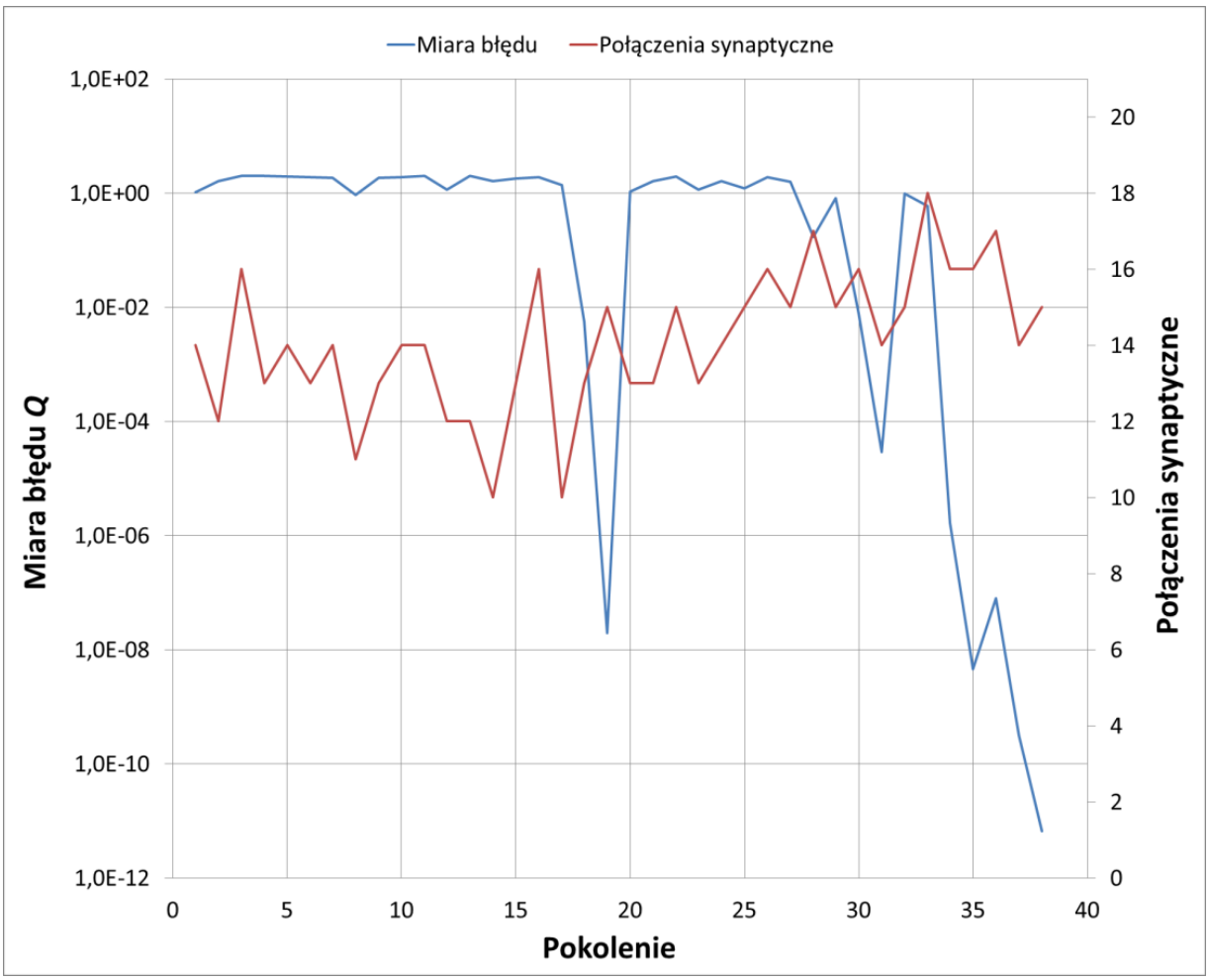
Na Rysunku 6.2 przedstawiony został wykres, prezentujący efektywność algorytmu Levenberga-Marquardta w poszukiwaniu rozwiązania problemu XOR dla jednokierunkowej sieci wielowarstwowej o minimalnej architekturze pozwalającej na jego rozwiązanie. Oś odciętych na wykresie przedstawia liczbę wykonanych epok uczenia sztucznej sieci neuronowej, rozumianych jako kompletne przetworzenie przez algorytm ciągu wzorów uczących. Na osi rzędnych odłożona została minimalizowana miara błędu Q sieci, osiągnięta na wybranym etapie uczenia. Kolory na Rysunku 6.2 reprezentują niezależne przebiegi procesu uczenia. Algorytm LM, średnio po 34 epokach uczenia, znajduje sieci, których miara błędu Q jest mniejsza lub równa założonemu maksymalnemu dopuszczalnemu błędowi sieci $Q_{max} = 10^{-10}$.



Rysunek 6.3 Efektywność neuroewolucji z wykorzystaniem algorytmu QIGA2 ($|Q| = 20$, $\mu = 0{,}99$) oraz kodowania wag sieci w poszukiwaniu rozwiązania problemu XOR. Kolory reprezentują niezależne przebiegi procesu uczenia.

Na Rysunku 6.3 przedstawiony został wykres, prezentujący efektywność neuroewolucji z wykorzystaniem algorytmu QIGA2 oraz kodowania wag sieci w poszukiwaniu rozwiązania problemu XOR dla jednokierunkowej sieci wielowarstwowej o minimalnej architekturze pozwalającej na jego rozwiązanie. Oś odciętych na wykresie przedstawia liczbę wygenerowanych przez algorytm genetyczny pokoleń, rozumianych jako kolejne powtórzenia procesu obserwacji populacji kwantowej, oceny zaobserwowanych osobników klasycznych oraz wykorzystania najlepszego z nich do aktualizacji stanu ich reprezentacji kwantowych. Na osi rzędnych odłożona została minimalizowana miara błędu Q sieci, osiągnięta na wybranym etapie neuroewolucji. Kolory na Rysunku 6.3 reprezentują niezależne przebiegi procesu neuroewolucji. Neuroewolucja z wykorzystaniem algorytmu QIGA2 oraz kodowania wag sieci, średnio po 33 pokoleniach, znajduje sieci, których miara błędu Q jest mniejsza lub równa założonemu maksymalnemu dopuszczalnemu błędowi sieci $Q_{max} = 10^{-10}$. Nie zawsze

pozostaje jednak odporna na występowanie minimów lokalnych optymalizowanej miary błędu Q, co pokazuje przebieg algorytmu oznaczony na Rysunku 6.3 kolorem zielonym.



Rysunek 6.4 Efektywność neuroewolucji z wykorzystaniem algorytmu QIGA2 ($|Q| = 100$, $\mu = 0{,}99$) oraz kodowania wag i topologii sieci w poszukiwaniu rozwiązania problemu XOR. Kolorem niebieskim wykreślona została miara błędu Q, kolorem czerwonym – liczba wykorzystanych połączeń synaptycznych.

Na Rysunku 6.4 przedstawiony został wykres, prezentujący efektywność neuroewolucji z wykorzystaniem algorytmu QIGA2 oraz kodowania wag i topologii sieci w poszukiwaniu rozwiązania problemu XOR dla jednokierunkowej sieci wielowarstwowej o nadmiarowej architekturze, złożonej z 6 neuronów połączonych przy pomocy 21 połączeń synaptycznych. Oś odciętych na wykresie przedstawia liczbę wygenerowanych przez algorytm genetyczny pokoleń, rozumianych jako kolejne powtórzenia procesu obserwacji populacji kwantowej, oceny zaobserwowanych osobników klasycznych oraz wykorzystania najlepszego z nich do aktualizacji stanu ich reprezentacji kwantowych. Na głównej osi rzędnych, po lewej stronie wykresu, odłożona została minimalizowana miara błędu Q sieci, osiągnięta na wybranym

etapie neuroewolucji, wykreślona kolorem niebieskim. Natomiast pomocnicza oś rzędnych, po prawej stronie wykresu, przedstawia liczbę wykorzystanych w danym rozwiązaniu połączeń synaptycznych, wykreśloną kolorem czerwonym. Neuroewolucja z wykorzystaniem algorytmu QIGA2 oraz kodowania wag i topologii sieci, po 38 pokoleniach, znalazła sieć, której miara błędu Q jest mniejsza lub równa założonemu maksymalnemu dopuszczalnemu błędowi sieci $Q_{max} = 10^{-10}$, wykorzystując do jej realizacji 15 połączeń synaptycznych. Proces neuroewolucji pozwolił zredukować nadmiarową architekturę jednokierunkowej sieci wielowarstwowej, stanowiącą ok. 233% architektury minimalnej, o ok. 30%, uzyskując topologię wykorzystującą ok. 166% architektury minimalnej.

Rozdział 7

PODSUMOWANIE I WNIOSKI

Sztuczne neurony są klasyfikatorami liniowymi, które połączone w sztuczną sieć neuronową mogą służyć do rozwiązywania problemów liniowo nieseparowalnych, np. klasyfikacji obiektów czy aproksymacji funkcji. Wiedza neuronów i sieci neuronowych jest zapisana w ich wagach synaptycznych, które odpowiednio koryguje się w procesie uczenia. Problem doboru architektury sieci, poruszany m.in. w pracach (Rutkowski, 2009) , (Białko, 2005), pozostaje otwarty, a praktyczne realizacje sieci neuronowych są projektowane doświadczalnie.

Algorytm wstecznej propagacji błędów, ze względu na swoją prostotę, stał się podstawową i najczęściej stosowaną metodą nadzorowanego uczenia sztucznych sieci neuronowych, przyczyniając się do szerokiego rozpowszechnienia samych sieci jako narzędzia do rozwiązywania problemów klasyfikacji czy aproksymacji. Dla małych wartości współczynnika uczenia η, algorytm EBP pozostaje stabilny kosztem wolnej zbieżności, co ilustruje Rysunek 6.1. Ponadto, jako algorytm uczący pierwszego rzędu wykorzystujący regułę największego spadku, nie jest on odporny na występowanie minimów lokalnych miary błędu. Złożoność pamięciowa algorytmu EBP, zorganizowanego jako przyrostowe uaktualnianie wag, jest zależna jedynie od liczby wag synaptycznych sieci, co sprawia, że jego implementacja może być z powodzeniem wykorzystana do nauki długich ciągów wzorów uczących P.

Algorytm Levenberga-Marquardta, łącząc w sobie stabilność metody największego spadku (Debye, 1909) oraz efektywność algorytmu Gaussa-Newtona (Gauss, 1809), stanowi jedną z najskuteczniejszych metod uczenia nadzorowanego (Hagan, Menhaj, 1994)., co ilustruje Rysunek 6.2. Jednocześnie charakteryzują go duża złożoność pamięciowa i obliczeniowa, związane odpowiednio z przechowywaniem i odwracaniem macierzy Jacobiego, co ogranicza wykorzystanie przy uczeniu dużych sieci neuronowych. Złożoność pamięciowa algorytmu LM, zorganizowanego jako kumulacyjne uaktualnianie wag, może jednak zostać uniezależniona od długości ciągów wzorów uczących P poprzez wyznaczanie niejawnej formy macierzy Jacobiego metodą propagacji w tył, opisane w Rozdziale 3.2, co znacznie rozszerza możliwości aplikacyjne algorytmu.

Algorytmy kwantowo inspirowane wyższych rzędów są lepiej przystosowane do rozwiązywania zadań zwodniczych dzięki modelowaniu zależności między zbiorami genów. Algorytm QIGA2 przewyższa prostotą i skutecznością działania zarówno pierwotny algorytm QIGA1 jak i wiele jego wariacji, mających prowadzić do podniesienia efektywności algorytmu pierwotnego. Dzięki uproszczeniu operatorów i zerwaniu tym samym z wykorzystaniem tablic LookupTable algorytm QIGA2 jest też ok. 15-30% szybszy od algorytmu QIGA1, co pokazują wyniki badań doświadczalnych przeprowadzonych w pracy (Nowotniak, Kucharski, 2014).

Ewolucja jednokierunkowej sieci wielowarstwowej z wykorzystaniem algorytmu QIGA2 rozwiązuje problem poszukiwania optymalnej topologii sieci i metody jej uczenia. Wykorzystując kodowanie wag synaptycznych pozwala tworzyć sieci o dużej dokładności, z miarą błędu rzędu 10^{-10} dla problemu XOR (Rysunek 6.3), trudnej lub czasami niemożliwej do osiągnięcia klasycznymi algorytmami uczenia pierwszego i drugiego rzędu (Rysunek 6.1). Nie wymaga przy tym realizacji mechanizmu wstecznej propagacji błędów, używając jedynie przetwarzania w przód w celu oceny przystosowania sieci. Jednoczesne kodowanie wag synaptycznych i topologii sieci, pozwala znaleźć sieci o optymalnej (zredukowanej) architekturze już nauczone (Rysunek 6.4). Zaproponowane podejście, zapewnia kompleksowe rozwiązanie problemu projektowania sieci neuronowych, niewymagające doświadczalnego poszukiwania korzystnej topologii czy skutecznych metod uczenia sieci.

ZAŁĄCZNIKI

Załącznik A ALGORYTMY UCZĄCE RZĘDU PIERWSZEGO

Proces uczenia neuronów i sieci neuronowych polega na takiej korekcie wektora ich wag synaptycznych $\boldsymbol{w} = [w_1, w_2, w_3, \ldots, w_N]^T$, która prowadziłaby do minimalizacji miary błędu $Q(\boldsymbol{w})$, kumulującej różnice między wartościami wzorcowymi a otrzymanymi sygnałami wyjściowym. Rozwijając funkcję $Q(\boldsymbol{w})$ w szereg Taylora w najbliższym sąsiedztwie znanego aktualnego rozwiązania w wzdłuż kierunku $\boldsymbol{p}$, otrzymuje się

$$Q(\boldsymbol{w}+\boldsymbol{p}) = Q(\boldsymbol{w}) + [\boldsymbol{g}(\boldsymbol{w})]^T\boldsymbol{p} + 0{,}5\,\boldsymbol{p}^T\boldsymbol{H}(\boldsymbol{w})\boldsymbol{p} + \cdots \tag{A.1}$$

gdzie $\boldsymbol{g}(\boldsymbol{w})$ oznacza wektor gradientu, tzn. wektor pierwszych pochodnych

$$\boldsymbol{g}(\boldsymbol{w}) = \left[\frac{\partial Q}{\partial w_1}, \frac{\partial Q}{\partial w_2}, \frac{\partial Q}{\partial w_3}, \ldots, \frac{\partial Q}{\partial w_N}\right]^T \tag{A.2}$$

a $\mathbf{H}(\mathbf{w})$ jest hesjanem (macierzą Hessego), tzn. macierzą drugich pochodnych

$$\boldsymbol{H}(\boldsymbol{w}) = \begin{bmatrix} \dfrac{\partial^2 Q}{\partial w_1 \partial w_1} & \cdots & \dfrac{\partial^2 Q}{\partial w_1 \partial w_N} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial^2 Q}{\partial w_N \partial w_1} & \cdots & \dfrac{\partial^2 Q}{\partial w_N \partial w_N} \end{bmatrix} \tag{A.3}$$

Modyfikację wag

$$\boldsymbol{w}^* = \boldsymbol{w} + \eta\boldsymbol{p} \tag{A.4}$$

gdzie $\eta > 0$ jest współczynnikiem uczenia, należy przeprowadzać w taki sposób, aby w kolejnych krokach uczenia błąd na wyjściu sieci malał, tj. aby spełniona była nierówność

$$Q(\boldsymbol{w}^*) < Q(\boldsymbol{w}) \tag{A.5}$$

Po ograniczeniu szeregu Taylora aproksymującego miarę błędu Q (A.1) do rozwinięcia liniowego, tzn.

$$Q(\boldsymbol{w}+\boldsymbol{p}) = Q(\boldsymbol{w}) + [\boldsymbol{g}(\boldsymbol{w})]^T\boldsymbol{p} \tag{A.6}$$

aby spełniona była nierówność (A.5), należy tak dobrać wektor $\boldsymbol{p}$, żeby

$$[\boldsymbol{g}(\boldsymbol{w})]^T\boldsymbol{p} < 0 \tag{A.7}$$

Warunek (A.7) jest spełniony, gdy

$$\boldsymbol{p} = -\,\boldsymbol{g}(\boldsymbol{w}) \tag{A.8}$$

Podstawiając zależność (A.8) do wzoru (A.4), otrzymuje się następujący sposób korekcji wag synaptycznych $\boldsymbol{w}$

$$\boldsymbol{w}^* = \boldsymbol{w} - \eta \boldsymbol{g}(\boldsymbol{w}) \tag{A.9}$$

Formuła (A.9) jest ogólną postacią **reguły największego spadku** (ang. *method of steepest descent*) (Debye, 1909). Algorytmy uczące wykorzystujące zależność (A.9), ograniczają aproksymację miary błędu *Q* szeregiem Taylora (A.1) do rozwinięcia liniowego (A.6) przez co nazywane są **algorytmami uczącymi rzędu pierwszego** (ang. *first-order training algorithms*).

Załącznik B KOREKCJA WAG NEURONU SIGMOIDALNEGO

Korzystając ze wzorów (1.1) oraz (1.2), sygnał wyjściowy *y* neuronu sigmoidalnego, przedstawionego w Rozdziale 1.2, można przedstawić jako

$$y = f(net) = f\left(\sum_{i=0}^{n} w_i\, x_i\right) \tag{B.1}$$

Miarę błędu *Q* definiuje się jako kwadrat różnicy wartości wzorcowej *d* i wartości sygnału wyjściowego *y* (B.1), tzn.

$$Q = \frac{1}{2}\left[d - f\left(\sum_{i=0}^{n} w_i\, x_i\right)\right]^2 \tag{B.2}$$

Algorytm korekcji wag neuronu sigmoidalnego jest **algorytmem uczącym rzędu pierwszego**, tj. do korekcji wag synaptycznych neuronu stosuje się regułę największego spadku (A.9), wyprowadzoną w Załączniku A, w postaci

$$w_i^* = w_i - \eta \frac{\partial Q}{\partial w_i} \tag{B.3}$$

Pochodną miary błędu (B.2) względem wagi w_i można przedstawić jako

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial net}\,\frac{\partial net}{\partial w_i} \tag{B.4}$$

Zauważając, że

$$\frac{\partial net}{\partial w_i} = x_i \tag{B.5}$$

oraz

$$\frac{\partial Q}{\partial net} = -\big(d - f(net)\big) f'(net) \tag{B.6}$$

i wykorzystując zależności (B.4), reguła korekcji wag (B.3) przyjmuje postać

$$w_i^* = w_i + \eta\big(d - f(net)\big)\, f'(net) x_i \tag{B.6}$$

Powyższe wyprowadzenia prowadzą do następującej formy algorytmu korekcji wag neuronu sigmoidalnego:

$$y = f(net) = f\left(\sum_{i=0}^{n} w_i\, x_i\right) \tag{B.7}$$

$$w_i^* = w_i + \eta\big(d - f(net)\big)\, f'(net) x_i \tag{B.8}$$

Załącznik C KOREKCJA WAG W ALGORYTMIE EBP

Sygnał wyjściowy neuronu N_i^k w jednokierunkowej sieci wielowarstwowej, przedstawionej w Rozdziale 1.3, jest określony wzorem:

$$y_i^k = f\big(net_i^k\big) = \sum_{j=0}^{N_{k-1}} x_j^k\, w_{i,j}^k \tag{C.1}$$

Sygnały wyjściowe sieci:

$$o_1 = y_1^L, o_2 = y_2^L, \ldots, o_{N_L} = y_{N_L}^L \tag{C.2}$$

porównywane są z sygnałami wzorcowymi zawartymi we wzorze uczącym $p = (\boldsymbol{x}, \boldsymbol{d})$:

$$d_1, d_2, \ldots, d_{N_L} \tag{C.3}$$

w wyniku czego uzyskuje się błędy w warstwie wyjściowej ($k = L$):

$$\varepsilon_i^L = d_i - o_i, \qquad i = 1, 2, \ldots, N_L \tag{C.4}$$

Miara błędu sieci *Q*, będąca rezultatem porównania sygnałów (C.2) oraz (C.3), jest zdefiniowana jako suma kwadratów różnic (C.4):

$$Q = \sum_{i=1}^{N_L} {\varepsilon_i^L}^2 = \sum_{i=1}^{N_L} (d_i - o_i)^2 \tag{C.5}$$

Wzory (C.1), (C.2) oraz (C.4) pokazują, że miara błędu *Q* jest funkcją wag sieci. Algorytm EBP jest **algorytmem uczącym rzędu pierwszego**, tj. do korekcji wag synaptycznych neuronu N_i^k stosuje regułę największego spadku (A.9), wyprowadzoną w Załączniku A, w postaci

$$w_{i,j}^{k\,*} = w_{i,j}^k - \eta \frac{\partial Q}{\partial w_{i,j}^k} \tag{C.6}$$

Przekształcając

$$\frac{\partial Q}{\partial w_{i,j}^k} = \frac{\partial Q}{\partial net_i^k} \frac{\partial net_i^k}{\partial w_{i,j}^k} = \frac{\partial Q}{\partial net_i^k}\, x_j^k \tag{C.6}$$

i oznaczając

$$\delta_i^k = -\frac{1}{2} \frac{\partial Q}{\partial net_i^k} \tag{C.7}$$

otrzymujemy równość

$$\frac{\partial Q}{\partial w_{i,j}^k} = -2\delta_i^k\, x_j^k \tag{C.8}$$

Zatem reguła korekcji wag (C.6) przyjmuje postać

$$w_{i,j}^{k\,*} = w_{i,j}^k + 2\eta\delta_i^k\, x_j^k \tag{C.9}$$

Sposób obliczania wartości δ_i^k określonej wzorem (C.7) dla neuronu N_i^k zależy od warstwy *k*. Dla warstwy ostatniej ($k = L$)

$$\begin{aligned} \delta_i^L &= -\frac{1}{2}\frac{\partial Q}{\partial net_i^L} = -\frac{1}{2}\frac{\partial {\varepsilon_i^L}^2}{\partial net_i^L} = -\frac{1}{2}\frac{\partial (d_i - o_i)^2}{\partial net_i^L} = \\ &= \varepsilon_i^L \frac{\partial y_i^L}{\partial net_i^L} = \varepsilon_i^L\, f'(net_i^L) \end{aligned} \tag{C.10}$$

Dla pozostałych warstw ($k \neq L$)

$$\delta_i^k = -\frac{1}{2}\frac{\partial Q}{\partial net_i^k} = -\frac{1}{2}\sum_{m=1}^{N_{k+1}} \frac{\partial Q}{\partial net_m^{k+1}} \frac{\partial net_m^{k+1}}{\partial net_i^k} = \\ = \sum_{m=1}^{N_{k+1}} \delta_m^{k+1} w_{m,i}^{k+1} f'\left(net_i^k\right) = f'\left(net_i^k\right) \sum_{m=1}^{N_{k+1}} \delta_m^{k+1} w_{m,i}^{k+1} \tag{C.11}$$

Definiując błąd w warstwie k (z wyjątkiem warstwy ostatniej, $k \neq L$) dla neuronu N_i^k jako

$$\varepsilon_i^k = \sum_{m=1}^{N_{k+1}} \delta_m^{k+1} w_{m,i}^{k+1}, \qquad k = 1, 2, \dots, L-1 \tag{C.12}$$

równanie (C.11) można zapisać jako

$$\delta_i^k = f'\left(net_i^k\right)\varepsilon_i^k \tag{C.13}$$

Powyższe wyprowadzenia prowadzą do następującej formy algorytmu wstecznej propagacji błędów:

$$y_i^k = f\left(net_i^k\right), \qquad net_i^k = \sum_{j=0}^{N_{k-1}} x_j^k \, w_{i,j}^k \tag{C.14}$$

$$\varepsilon_i^k = \begin{cases} d_i - o_i, & k = L \\ \sum_{m=1}^{N_{k+1}} \delta_m^{k+1} w_{m,i}^{k+1}, & k = 1, 2, \dots, L-1 \end{cases} \tag{C.15}$$

$$\delta_i^k = f'\left(net_i^k\right)\varepsilon_i^k \tag{C.16}$$

$$w_{i,j}^{k\,*} = w_{i,j}^k + 2\eta\delta_i^k \, x_j^k \tag{C.17}$$

Załącznik D ALGORYTMY UCZĄCE RZĘDU DRUGIEGO

Przedstawione w Załączniku A algorytmy uczące rzędu pierwszego ograniczają aproksymację miary błędu $Q(\boldsymbol{w})$ szeregiem Taylora (A.1) do rozwinięcia liniowego (A.6). **Algorytmy uczące rzędu drugiego** (ang. *second–order training algorithms*) korzystają z kwadratowego rozwinięcia aproksymacji miary błędu $Q(\boldsymbol{w})$ szeregiem Taylora (A.1), tj.

$$Q(\boldsymbol{w} + \boldsymbol{p}) = Q(\boldsymbol{w}) + [\boldsymbol{g}(\boldsymbol{w})]^T \boldsymbol{p} + 0{,}5\, \boldsymbol{p}^T \boldsymbol{H}(\boldsymbol{w}) \boldsymbol{p} \tag{D.1}$$

Sprowadzając problem korekcji wag sieci neuronowej do zadania minimalizacji miary błędu $Q(\boldsymbol{w})$, w celu znalezienia minimum kryterium $Q(\boldsymbol{w})$ względem wektora $\boldsymbol{p}$, należy spełnić następujące warunki wystarczające:

$$\frac{\partial Q(\boldsymbol{w}+\boldsymbol{p})}{\partial \boldsymbol{p}} = [\boldsymbol{g}(\boldsymbol{w})]^T + \boldsymbol{H}(\boldsymbol{w})\boldsymbol{p} = 0 \tag{D.2}$$

oraz

$$\frac{\partial^2 Q(\boldsymbol{w}+\boldsymbol{p})}{\partial^2 \boldsymbol{p}} = \boldsymbol{H}(\boldsymbol{w}) > 0 \tag{D.3}$$

Z warunku (D.2) wynika, że

$$\boldsymbol{p} = -[\boldsymbol{H}(\boldsymbol{w})]^{-1}\boldsymbol{g}(\boldsymbol{w}) \tag{D.4}$$

Podstawiając zależność (D.4) do wzoru modyfikacji wag (A.4), definiuje się następujący sposób korekcji wektora wag synaptycznych $\boldsymbol{w}$:

$$\boldsymbol{w}^* = \mathbf{w} - \eta[\mathbf{H}(\mathbf{w})]^{-1}\mathbf{g}(\mathbf{w}) \tag{D.5}$$

Formuła (D.5) jest znana jako **metoda Newtona** (ang. *Newton's method*), opisana w pracy (Fletcher, 2013). Ponadto, aby miara błędu $Q(\boldsymbol{w} + \boldsymbol{p})$ osiągnęła minimum w danym punkcie, hesjan (A.3) powinien być w tym punkcie dodatnio określony, zgodnie z warunkiem (D.3). Ten warunek jest jednak trudny do spełnienia, dlatego algorytmy uczące rzędu drugiego proponują różne przybliżenia wartości hesjanu oraz wektora gradientu, które dodatkowo pozwalają ominąć problem, często kosztownego, liczenia pochodnych drugiego rzędu.

Załącznik E KOREKCJA WAG W ALGORYTMIE LM

Miara błędu Q sieci neuronowej przedstawionej w Rozdziale 1.3, będąca rezultatem porównania sygnałów (C.2) oraz (C.3) dla ciągu wzorów uczących $P = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), \ldots, (\boldsymbol{x}_{|P|}, \boldsymbol{d}_{|P|})\}$, jest zdefiniowana jako suma kwadratów różnic (C.4) dla wszystkich wzorów uczących w ciągu P:

$$Q = \frac{1}{2}\sum_{p=1}^{|P|}\sum_{m=1}^{N_L} {\varepsilon_{p,m}^L}^2 = \frac{1}{2}\sum_{p=1}^{|P|}\sum_{m=1}^{N_L}\left(d_{p,m} - o_{p,m}\right)^2 \tag{E.1}$$

Algorytm Levenberga–Marquardta jest **algorytmem uczącym rzędu drugiego**, tj. do korekcji wektora wag synaptycznych $\boldsymbol{w}$ stosuje metodę Newtona (D.5), wyprowadzoną

w Załączniku D. Z użyciem definicji miary błędu (E.1), elementy macierzy Hessego (A.3) mogą zostać wyznaczone jako

$$\frac{\partial^2 \mathrm{Q}}{\partial w_{i,j}^k \, \partial w_{r,s}^t} = \sum_{\mathrm{p}=1}^{|\mathrm{P}|} \sum_{\mathrm{m}=1}^{N_L} \left(\frac{\partial \varepsilon_{p,m}^L}{\partial w_{i,j}^k} \frac{\partial \varepsilon_{p,m}^L}{\partial w_{r,s}^t} + \frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{i,j}^k \, \partial w_{r,s}^t} \varepsilon_{p,m}^L \right) \tag{E.2}$$

gdzie element

$$\sum_{p=1}^{|P|} \sum_{m=1}^{N_L} \left(\frac{\partial^2 \varepsilon_{p,m}^L}{\partial w_{i,j}^k \partial w_{r,s}^t} \varepsilon_{p,m}^L \right) \tag{E.3}$$

zgodnie z założeniami metody Newtona (D.5), jest bliski zeru (Hagan, Menhaj, 1994), wobec czego formułę (E.2) można aproksymować jako

$$\frac{\partial^2 Q}{\partial w_{i,j}^k \partial \, \partial w_{r,s}^t} \approx \sum_{p=1}^{|P|} \sum_{m=1}^{N_L} \left(\frac{\partial \varepsilon_{p,m}^L}{\partial w_{i,j}^k} \frac{\partial \varepsilon_{p,m}^L}{\partial w_{r,s}^t} \right) \tag{E.4}$$

Algorytm Levenberga-Marquardta, korzystając z aproksymacji elementów hesjanu (E.4) oraz macierzy Jacobiego, tzn. macierzy pochodnych cząstkowych pierwszego rzędu błędów (C.4) tworzących miarę błędu Q (E.1), postaci

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial \varepsilon_{1,1}^L}{\partial w_{1,0}^1} & \cdots & \frac{\partial \varepsilon_{1,1}^L}{\partial w_{i,j}^k} & \cdots & \frac{\partial \varepsilon_{1,1}^L}{\partial w_{N_L,N_{L-1}}^L} \\ \vdots & \ddots & \vdots & \iddots & \cdots \\ \frac{\partial \varepsilon_{1,N_L}^L}{\partial w_{1,0}^1} & \cdots & \frac{\partial \varepsilon_{1,N_L}^L}{\partial w_{i,j}^k} & \cdots & \frac{\partial \varepsilon_{1,N_L}^L}{\partial w_{N_L,N_{L-1}}^L} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ \frac{\partial \varepsilon_{p,m}^L}{\partial w_{1,0}^1} & \cdots & \frac{\partial \varepsilon_{p,m}^L}{\partial w_{i,j}^k} & \cdots & \frac{\partial \varepsilon_{p,m}^L}{\partial w_{N_L,N_{L-1}}^L} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_{|P|,1}^L}{\partial w_{1,0}^1} & \cdots & \frac{\partial \varepsilon_{|P|,1}^L}{\partial w_{i,j}^k} & \cdots & \frac{\partial \varepsilon_{|P|,1}^L}{\partial w_{N_L,N_{L-1}}^L} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_{|P|,N_L}^L}{\partial w_{1,0}^1} & \cdots & \frac{\partial \varepsilon_{|P|,N_L}^L}{\partial w_{i,j}^k} & \cdots & \frac{\partial \varepsilon_{|P|,N_L}^L}{\partial w_{N_L,N_{L-1}}^L} \end{bmatrix} \tag{E.5}$$

wprowadza następującą aproksymację macierzy Hessego (A.3)

$$\boldsymbol{H} \approx \boldsymbol{Q} + \mu \boldsymbol{I} = \boldsymbol{J}^T \boldsymbol{J} + \mu \boldsymbol{I} \tag{E.6}$$

gdzie $\boldsymbol{Q}$ to pseudohesjan, którego elementy są określone zależnością (E.4), $\boldsymbol{I}$ to macierz jednostkowa, a $\boldsymbol{\mu} > 0$ to współczynnik kombinacji. Przybliżenie hesjanu z zależności (A.3) jako (E.6) sprawia, że elementy na diagonali macierzy będą zawsze większe od zera, co gwarantuje, że przybliżona macierz Hessego będzie odwracalna.

Z użyciem definicji miary błędu (E.1), elementy wektora gradientu (A.2) mogą zostać wyznaczone jako

$$\frac{\partial Q}{\partial w_{i,j}^{k}} = \sum_{p=1}^{P} \sum_{m=1}^{N_L} \left(\frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k}} \varepsilon_{p,m}^{L} \right) \tag{E.7}$$

Korzystając z macierzy Jacobiego (E.5) oraz wektora błędów e o następującej postaci

$$\boldsymbol{e} = \left[\varepsilon_{1,1}^{L}, \dots, \varepsilon_{1,N_L}^{L}, \dots, \varepsilon_{p,m}^{L}, \dots, \varepsilon_{|P|,1}^{L}, \dots, \varepsilon_{|P|,N_L}^{L} \right]^{T} \tag{E.8}$$

wektor gradientu o elementach (E.7) można przedstawić jako

$$\boldsymbol{g} = \boldsymbol{J}^{T} \boldsymbol{e} \tag{E.9}$$

Połączenie zależności (D.5) z (E.6) i (E.9) prowadzi do sformułowania następującej reguły korekcji wektora wag $\boldsymbol{w}$:

$$\boldsymbol{w}^{*} = \boldsymbol{w} - [\boldsymbol{Q} + \mu \boldsymbol{I}]^{-1} \boldsymbol{g} = \boldsymbol{w} - [\boldsymbol{J}^{T} \boldsymbol{J} + \mu \boldsymbol{I}]^{-1} \boldsymbol{J}^{T} \boldsymbol{e} \tag{E.10}$$

Powyższe wyprowadzenia prowadzą do następującej formy algorytmu Levenberga-Marquardta:

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial \varepsilon_{1,1}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{1,1}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{1,1}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \ddots & \vdots & \iddots & \cdots \\ \frac{\partial \varepsilon_{1,N_L}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{1,N_L}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{1,N_L}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_{|P|,1}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{|P|,1}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{|P|,1}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \frac{\partial \varepsilon_{|P|,N_L}^{L}}{\partial w_{1,0}^{1}} & \cdots & \frac{\partial \varepsilon_{|P|,N_L}^{L}}{\partial w_{i,j}^{k}} & \cdots & \frac{\partial \varepsilon_{|P|,N_L}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \end{bmatrix} \tag{E.11}$$

$$\boldsymbol{e} = \left[\varepsilon_{1,1}^{L}, \dots, \varepsilon_{1,N_L}^{L}, \dots, \varepsilon_{p,m}^{L}, \dots, \varepsilon_{|P|,1}^{L}, \dots, \varepsilon_{|P|,N_L}^{L}\right]^T \tag{E.12}$$

$$\boldsymbol{w}^* = \boldsymbol{w} - [\boldsymbol{Q} + \mu \boldsymbol{I}]^{-1}\boldsymbol{g} = \boldsymbol{w} - [\boldsymbol{J}^T\boldsymbol{J} + \mu \boldsymbol{I}]^{-1}\boldsymbol{J}^T\boldsymbol{e} \tag{E.13}$$

Nazwa współczynnika kombinacji μ oddaje naturę algorytmu Levenberga-Marquardta, który stanowi połączenie reguły największego spadku (A.9) oraz algorytmu Gaussa-Newtona (Gauss, 1809), przybliżającego metodę Newtona (D.5) jako

$$\boldsymbol{H} \approx \boldsymbol{Q} = \boldsymbol{J}^T\boldsymbol{J}, \qquad \boldsymbol{g} = \boldsymbol{J}^T\boldsymbol{e} \tag{E.14}$$

$$\boldsymbol{w}^* = \boldsymbol{w} - [\boldsymbol{J}^T\boldsymbol{J}]^{-1}\boldsymbol{J}^T\boldsymbol{e} \tag{E.15}$$

Dla małych wartości współczynnika μ (bliskich zero), reguła korekcji wag w algorytmie Levenberga-Marquardta (E.13) zbliża się do reprezentacji reguły w algorytmie Gaussa-Newtona (E.15). Kiedy współczynnik kombinacji μ jest bardzo duży, reguła korekcji wag (E.13) przybliża regułę największego spadku (A.9).

Załącznik F WYZNACZANIE JAWNEJ FORMY MACIERZY JACOBIEGO METODĄ PROPAGACJI W TYŁ

W strukturze jednokierunkowej sieci wielowarstwowej, przedstawionej w Rozdziale 1.3, między wyjściem y_i^k neuronu ukrytego N_i^k a wyjściem sieci neuronowej o_m istnieje złożona relacja nieliniowa

$$o_m = F_{m,i}^k\left(y_i^k\right) \tag{F.1}$$

Elementy macierzy Jacobiego (E.5) mogą zostać wyrażone jako

$$\begin{aligned}\frac{\partial \varepsilon_{p,m}^L}{\partial w_{i,j}^k} &= \frac{\partial\left(d_{p,m} - o_{p,m}\right)}{\partial w_{i,j}^k} = -\frac{\partial o_{p,m}}{\partial w_{i,j}^k} = \\ &= -\frac{\partial o_{p,m}}{\partial y_i^k}\frac{\partial y_i^k}{\partial net_i^k}\frac{\partial net_i^k}{\partial w_{i,j}^k} = -F'^k_{m,i}\, f'(net_i^k)\, x_j^k\end{aligned} \tag{F.2}$$

Korzystając z zależności (F.2) proces obliczania macierzy Jacobiego może zostać zorganizowany podobnie do procesu propagacji w tył w algorytmach uczących pierwszego rzędu, np. w wyprowadzonym w Załączniku C algorytmie EBP, z uwzględnieniem dwóch istotnych różnic (Yu, Wilamowski, 2011). W algorytmie EBP dla każdego wzoru uczącego, wystarczyło dokonać jednego procesu wstecznej propagacji błędów, błędy (C.15) wszystkich wyjść o_1, …, o_{N_L} były propagowane jednocześnie. Podczas obliczania macierzy Jacobiego

w algorytmie Levenberga-Marquardta proces wstecznej propagacji błędów musi zostać wykonany oddzielnie dla każdego wyjścia sieci, aby otrzymać kolejne wiersze macierzy (E.5). Ponadto zmianie musi ulec koncepcja propagowanego wstecz parametru δ_i^k. W algorytmie EBP, błędy wyjść sieci są częścią parametru δ_i^k (C.16)

$$\delta_i^k = f'(net_i^k) \sum_{m=1}^{N_L} F'^k_{m,i}\, \varepsilon_m^L \tag{F.3}$$

W przypadku algorytmu Levenberga-Marquardta, parametr $\delta_{m,i}{}^k$ jest liczony dla każdego neuronu N_i^k i każdego wyjścia o_m oddzielnie, zatem błąd we wzorze (F.3) musi zostać zastąpiony wartością jednostkową, tj.

$$\delta_{m,i}^k = f'(net_i^k)\, F'^k_{m,i} \tag{F.4}$$

Łącząc zależności (F.2) oraz (F.4), elementy macierzy Jacobiego (E.5) przyjmują następującą postać

$$\frac{\partial \varepsilon_{p,m}^L}{\partial w_{i,j}^k} = -\delta_{m,i}^k x_j^k \tag{F.5}$$

Obliczenie elementu macierzy Jacobiego, odpowiadającego zależności między błędem wyjścia o_m w odpowiedzi na wzór uczący $p = (\boldsymbol{x}_p, \boldsymbol{d}_p)$ a wagą synaptyczna $w_{i,j}{}^k$ neuronu N_i^k, wykorzystując zależność (F.5), sprowadza się do znalezienia wartości współczynnika $\delta_{m,i}{}^k$ oraz sygnału wejściowego x_j^k. Sygnał x_j^k jest wyznaczany podczas propagacji wzoru uczącego w przód, z warstwy wejściowej do warstwy wyjściowej sieci, zgodnie metodą opisaną w Rozdziale 1.3, a wartość współczynnika $\delta_{m,i}{}^k$ można uzyskać propagując błędy w tył z warstwy wyjściowej do warstwy wejściowej sieci. Powyższe wyprowadzenia prowadzą do następującego sposobu obliczania elementów macierzy Jacobiego:

$$y_i^k = f\left(net_i^k\right), \qquad net_i^k = \sum_{j=0}^{N_{k-1}} x_j^k\, w_{i,j}^k \tag{F.6}$$

$$\hat{\varepsilon}_{m,i}^k = \begin{cases} 1, & k = L \wedge j = m \\ 0, & k = L \wedge j \neq m \\ \sum_{j=1}^{N_{k+1}} \hat{\delta}_{m,j}^{k+1}\, w_{j,i}^{k+1}, & k = 1, 2, \ldots, L-1 \end{cases} \tag{F.7}$$

$$\hat{\delta}_{m,i}^k = f'\left(net_i^k\right) \hat{\varepsilon}_{m,i}^k \tag{F.8}$$

$$\frac{\partial \varepsilon_{p,m}^L}{\partial w_{i,j}^k} = -\hat{\delta}_{m,i}^k x_j^k \tag{F.9}$$

Załącznik G WYZNACZANIE NIEJAWNEJ FORMY MACIERZY JACOBIEGO METODĄ PROPAGACJI W TYŁ

Podczas wyznaczania jawnej formy macierzy Jacobiego metodą propagacji w tył, wyprowadzonej w Załączniku F, wszystkie elementy macierzy są obliczane i przechowywane, by następnie posłużyć do korekcji wag sieci (E.12). Macierz Jacobiego (E.5) w algorytmie Levenberga-Marquardta po przetworzeniu $|P|$ wzorów uczących przez sieć neuronową o N_L wyjściach i N wagach synaptycznych, przybiera rozmiary ($|P|$ x N_L) x N. Ta cecha może znacząco utrudnić implementację algorytmu dla dużej liczby wzorców uczących i sieci o skomplikowanej strukturze. Problem ten można jednak rozwiązać poprzez zmianę organizacji procesu wyznaczania macierzy Jacobiego zaproponowaną w pracy (Wilamowski, Yu, 2010).

W przypadku wyznaczania jawnej formy macierzy Jacobiego metodą propagacji w tył, przedstawionego w Załączniku F, przetworzenie przez sieć jednego wzoru uczącego $p = (\boldsymbol{x}_\mathrm{p}, \boldsymbol{d}_\mathrm{p})$ i spropagowanie w tył błędu wyjścia o_m prowadzi do wyznaczenia jednego wiersza macierzy Jacobiego lub jednej kolumny jej transpozycji, stanowiącego wektor

$$\boldsymbol{j}_{p,m} = \left[\frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{1,0}^{1}}, \dots, \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k}}, \dots, \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{N_L,N_{L-1}}^{L}}\right] \tag{G.1}$$

Korzystając z zależności (E.4) pseudohesjan z aproksymacji (E.6) można przedstawić jako

$$\boldsymbol{Q} = \sum_{p=1}^{|P|} \sum_{m=1}^{N_L} \boldsymbol{q}_{\mathrm{p,m}} \tag{G.2}$$

gdzie $\boldsymbol{q}_{\mathrm{p,m}}$ to podmacierz pseudohesjanu postaci

$$\boldsymbol{q}_{p,m} = \boldsymbol{j}_{p,m}{}^{T} \boldsymbol{j}_{p,m} = \begin{bmatrix} \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{1,0}^{1} \partial w_{1,0}^{1}} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{1,0}^{1} \partial w_{i,j}^{k}} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{1,0}^{1} \partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k} \partial w_{1,0}^{1}} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k} \partial w_{i,j}^{k}} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k} \partial w_{N_L,N_{L-1}}^{L}} \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{N_L,N_{L-1}}^{L} \partial w_{1,0}^{1}} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{N_L,N_{L-1}}^{L} \partial w_{i,j}^{k}} & \cdots & \frac{\partial^2 \varepsilon_{p,m}^{L}}{\partial w_{N_L,N_{L-1}}^{L} \partial w_{N_L,N_{L-1}}^{L}} \end{bmatrix} \tag{G.3}$$

Analogicznie, uwzględniając zależność (E.7), wektor gradientu (E.9) może przybrać formę

$$\boldsymbol{g} = \sum_{p=1}^{P} \sum_{m=1}^{N_L} \boldsymbol{\eta}_{p,m} \tag{G.4}$$

gdzie $\boldsymbol{\eta}_{p,m}$ to podwektor wektora gradientu postaci

$$\boldsymbol{\eta}_{p,m} = \boldsymbol{j}_{p,m}{}^{T} \varepsilon_{p,m}^{L} = \begin{bmatrix} \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{1,0}^{1}} \varepsilon_{p,m}^{L} \\ \vdots \\ \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{i,j}^{k}} \varepsilon_{p,m}^{L} \\ \vdots \\ \frac{\partial \varepsilon_{p,m}^{L}}{\partial w_{N_L,N_{L-1}}^{L}} \varepsilon_{p,m}^{L} \end{bmatrix} \tag{G.5}$$

Wyznaczanie niejawnej formy macierzy Jacobiego jako pseudohesjanu $\boldsymbol{Q}$, przybierającego rozmiary N x N, pozwala uniezależnić złożoność pamięciową problemu od liczby wzorów uczących i zawartych w sieci neuronów, nie wpływając jednocześnie na złożoność obliczeniową, tzn. ilość operacji mnożenia i dodawania potrzebna do wyznaczenia macierzy Jacobiego w formie jawnej (Załącznik F) i niejawnej jest dokładnie taka sama.

Załącznik H APROKSYMACJA JAWNEJ FORMY MACIERZY JACOBIEGO METODĄ PROPAGACJI W PRZÓD

W podejściu zaprezentowanym w Załącznikach F oraz G wyznaczanie macierzy Jacobiego na potrzeby algorytmu Levenberga-Marquardta (Załącznik E) wymaga zarówno wykonywania przez sieć obliczeń w przód (ang. *forward calulations*) jak i w tył (ang. *backward calculations*). Najpierw sieć przetwarza sygnały wejściowe w przód, aby określić wartości wektora błędów $\boldsymbol{e}$ (E.8), a następnie propagując błędy w tył, otrzymuje się elementy macierzy Jacobiego (F.5). Wielokrotna propagacja błędów w tył jest w dużej mierze odpowiedzialna za złożoność obliczeniową algorytmu Levenberga-Marquardta, wobec tego w pracy (Wilamowski, Cotton, Hewlett, Kaynak, 2007) zaproponowano **aproksymację macierzy Jacobiego wykorzystującą jedynie metodę propagacji w przód** (ang. *feed-forward jacobian calculation*).

Aproksymacja macierzy Jacobiego metodą propagacji w przód wymaga, aby elementy macierzy były wyrażone dla każdego neuronu N_i^k w sieci jako funkcje jego wejść. Korzystając z zależności (F.2) elementy macierzy Jacobiego można przedstawić jako

$$\frac{\partial \varepsilon_{p,m}^L}{\partial w_{i,j}^k} = -\frac{\partial o_{p,m}}{\partial w_{i,j}^k} = -\frac{\partial o_{p,m}}{\partial net_i^k}\frac{\partial net_i^k}{\partial w_{i,j}^k} = -\frac{\partial o_{p,m}}{\partial net_i^k}x_j^k \tag{H.1}$$

Obliczenie elementu macierzy Jacobiego odpowiadającego zależności między błędem $\varepsilon_{p,m}{}^L$ wyjścia o_m w odpowiedzi na wzór uczący $p = (\boldsymbol{x}_p, \boldsymbol{d}_p)$ a wagą synaptyczną $w_{i,j}{}^k$ na wejściu x_j^k neuronu N_i^k, wykorzystując zależność (H.1), sprowadza się do znalezienia wartości $\partial o_{p,m}/\partial net_i^k$ oraz sygnału wejściowego x_j^k. Oba te elementy mogą zostać obliczone podczas propagacji wzoru uczącego p w przód, z warstwy wejściowej do warstwy wyjściowej sieci, zgodnie z metodą opisaną w Rozdziale 1.3. O ile wyznaczenie wartości sygnału wejściowego x_j^k jest zadaniem trywialnym, o tyle poznanie wartości $\partial o_{p,m}/\partial net_i^k$ wymaga nowego podejścia do problemu. Wartość $\partial o_{p,m}/\partial net_i^k$ odpowiada zależności między sygnałem wyjściowym o_m w odpowiedzi na wzór uczący p a sumą sygnałów wejściowych neuronu N_i^k, która może być aproksymowana numerycznie z wykorzystaniem **metody różnic skończonych** (ang. *finite-difference methods* – FDM) (Smith, 1985) w następujący sposób

$$\frac{\partial o_{p,m}}{\partial net_i^k} \approx \frac{o_{p,m}\left(net_i^k + h\right) - o_{p,m}\left(net_i^k\right)}{h} \tag{H.2}$$

W celu aproksymowania wartości $\partial o_{p,m}/\partial net_i^k$ należy dokonać niewielkiego przesunięcia wartości net_i^k o wartość h, a następnie, po spropagowaniu zmiany w przód na wyjście o_m, określić zmianę wyjścia $o_{p,m}$ i podzielić ją przez wartość przesunięcia h wartości net_i^k. Wyznaczanie wartości sygnału wyjściowego x_j^k oraz zależności $\partial o_{p,m}/\partial net_i^k$ wymaga jedynie wielokrotnego wykonywania przetwarzania w przód, nie używając kosztownego mechanizmu propagacji błędów w tył. Wadą takiego podejścia jest ograniczona dokładność przybliżenia elementów macierzy Jacobiego (E.5), uzależniona w pełni od wielkości przesunięcia h wartości net_i^k.

BIBLIOGRAFIA

1) Allauddin, R., Boehmer, S., Behrman, E., Gaddam, K., Steck, J., Quantum Simulataneous Recurrent Networks for Content Addressable Memory. *Quantum Inspired Intelligent Systems*, pp. 57-76, 2008.

2) Bear, M. F., Connors, B. W., Paradiso, M. A., *Neuroscience: exploring the brain*. Lippincott Williams Wilkins, Philadelphia, 2007. ISBN 0-7817-6003-8.

3) Białko, M., *Sztuczna inteligencja i elementy hybrydowych systemów ekspertowych*. Wydawnictwo Uczelnianie Politechniki Koszalińskiej, Koszalin, 2005. ISBN 83-7365-088-1.

4) Bryson, A. E., Denham, W., Dreyfus, S., Optimal Programming Problem with Inequalit Constraints. Part I: Necessary Conditions for Extremal Solutions. *AIAA Jour.*, 1, pp. 2544-2550, 1963.

5) Bryson, A. E., Ho, Y.-C., *Applied optimal control*. Blaisdell, New York, 1969.

6) Buras, A., Ellis, J., Gaillard, M., Nanopoulos, D., Aspects of the grand unification of strong, weak and electromagnetic interactions. *Nuclear Physics B*, 135(1):66–92, 1978.

7) Coelho, L., Nedjah, N., Mourelle, L., Gaussian quantum-behaved particle swarm optimization applied to fuzzy pid controller design. *Quantum Inspired Intelligent Systems*, pp. 1-15, 2008.

8) Compton, A., A quantum theory of the scattering of X-rays by light elements. *Physical Review*, 21(5):483, 1923.

9) da Cruz, A. A., Vellasco, M. M. B. R., Pacheco, M. A. C., Quantum-inspired evolutionary algorithm for numerical optimization. *Hybrid evolutionary algorithms*, Springer, Berlin, Heidelberg, pp. 19-37, 2007.

10) Debye, P., The diffraction theory of aberrations. *Ann. Phys.(Leipzig), 30*, pp. 59-62, 1909.

11) Eberhart, R., Shi, Y., Kennedy, J., *Swarm intelligence*. Morgan Kaufmann, 2001.

12) Einstein, A., *Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt*. JA Barth, 1905.

13) Fahlman, S. E., *An empirical study of learning speed in back-propagation networks*, Carnegie Mellon University, Computer Science Department, pp. 88-162, 1988.

14) Feynman, R., Simulating physics with computers. *International journal of theoretical physics*, 21(6):467-488, 1982.

15) Fletcher, R. *Practical methods of optimization*. John Wiley & Sons, 2013.

16) Gamma, E., Helm, R., Johnson, R., & Vlissides, J., Design patterns: Abstraction and reuse of object-oriented design. *European Conference on Object-Oriented Programming*, Springer, Berlin, Heidelberg, pp. 406-431, 1993.

17) Gauss, C. F., *Theoria motus corporum coelestium in sectionibus conicis solem ambientium* (Vol. 7). Perthes et Besser, 1809.

18) Hagan, M. T.,Menhaj, M. B., Training feedforward networks with the Marquardt algorithm. *IEEE transactions on Neural Networks*, *5*(6), pp. 989-993, 1994.

19) Han, K., Kim, J., Genetic quantum algorithm and its application to combinatorial optimization problem. *Proceedings of the 2000 Congress on Evolutionary computation*, volume 2, Citeseer, pp. 1354–1360, 2000

20) Hawking, S., Public Lectures: Does God Play Dice?, 2007.

21) Hebb, D. O., *The organization of behaviour*. Wiley, New York, 1949.

22) Higgs, P., Broken symmetries and the masses of gauge bosons. *Physical Review Letters*, 13(16):508, 1964.

23) Holland, J., Adaptation in natural and artificial system: an introduction with application to biology, control and artificial intelligence. *Ann Arbor, University of Michigan Press*, 1975.

24) Igielnik, B., Kolmogorovs Spline Network. *IEEE Transactions on Neural Networks*, 14, No 4, July 2003.

25) Kolmogorov, A. N., On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Dokl. Akad. Nauk SSSR*, 114:5, pp. 953-956, 1957

26) Krzyśko, M., Wołyński, W., Górecki, T., Skorzybut, M., *Systemy uczące się, rozpoznawanie wzorców, analiza skupień i redukcja wymiarowości.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2008. ISBN 978-83-204-3459-0.

27) Kurzweil, R., *How to Create a Mind: The Secret of Human Thought Revealed*. Viking Adult, 2012.

28) LeCun, Y., Denker, J. S., Solla, S. A., Optimal brain damage. *Advances in neural information processing systems*, pp. 598-605, 1990.

29) Levenberg, K., A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, *2*(2), pp. 164-168, 1944.

30) Luitel, B., Venayagamoorthy, G., Quantum inspired PSO for the optimization of simultaneous recurrent neural networks as MIMO learning systems. *Neural Networks*, 2010.

31) Łęski, J., *Systemy neuronowo-rozmyte.* Wydawnictwa Naukowo-Techniczne, Warszawa, 2008. ISBN 978-83-204-3229-9.

32) Manju, A., Nigam, M., Applications of quantum inspired computational intelligence: a survey. *Artificial Intelligence Review*, pp. 1-78, 2012. ISSN 0269-2821. 10.1007/s10462- 012-9330-6.

33) Marquardt, D. W., An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics, 11*(2), pp. 431-441, 1963.

34) McCulloch, W. S., Pitts, W., A logical callculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, pp. 115-133, 1943.

35) McCulloch, W. S., Pitts, W., How we know universals the perception of auditory and visual forms. *Bulletin of Mathematical Biophysics*, 9, pp. 127-147, 1947.

36) Menneer, T., Narayanan, A., Quantum-inspired neural networks. *Department of Computer Science, University of Exeter, Exeter, United Kingdom, Technical Report R*, 329:1995, 1995.

37) Michalkiewicz, J., Modified Kolmogorov's theorem. *Mathematica Applicanda, 37*(51/10), 2012.

38) Miller, G. F., Todd, P. M., Hagde, S. U., Designing neural networks using genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, Schaffer, J. D., (Ed.), Morgan Kaufmann, San Mateo, CA, pp. 379-384, 1989.

39) Minsky, M., Papert, S., *Perceptrons: an introduction to computational geometry*. The MIT Press, Cambridge, 1969.

40) Narayanan, A., Moore, M., Quantum-inspired genetic algorithms. *Proceedings of IEEE international conference on evolutionary computation*, IEEE, pp. 61-66, 1996.

41) Nielsen, M. A., Chuang, I. L., *Quantum Computation and Quantum Information*. Cambridge University Press, October 2000. ISBN 521635039.

42) Novaes, S., Standard model: An introduction. *arXiv preprint hep-ph/0001283*, 2000.

43) Nowotniak, R., Kucharski, J., Higher-Order Quantum-Inspired Genetic Algorithms. *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*, Annals of Computer Science and Information Systems, pp. 465–470. IEEE, 2014. ISBN 978-83- 60810-58-3.

44) Planck, M., On the law of distribution of energy in the normal spectrum. *Annalen der Physik*, 4(553):1, 1901.

45) Riedmiller, M., Braun, H., Rprop: A fast adaptive learning algorithm. *Proc. of the Int. Symposium on Computer and Information Science VII*, 1992.

46) Ronald, E., Schoenauer, M., Genetic lander: An experiment in accurate neuro-genetic control. *International Conference on Parallel Problem Solving from Nature*, Springer, Berlin, Heidelberg, pp. 452-461, 1994.

47) Rosenblueth, A., Wiener, N., Bigelow, J., Behavior, Purpose and Teleolgy. *Philosophy of Science*, pp. 18-24, 1943.

48) Rummelhart, D. E., Hinton, G. E., Williams, R. J., Learning representations by back-propagating errors. *Nature*, 323(9), pp. 553-536, 1986.

49) Rummelhart, D. E., McClelland, J. L., *Parallel distributed processing: Exploration in the microstructure of cognition. Volume I: Foundations.* MIT Press, Cambridge, 1986.

50) Russell, S., Norvig, P., Davis, E., Russell, S., Russell, S., *Artificial intelligence: a modern approach.* Prentice hall Upper Saddle River, NJ, 2010.

51) Rutkowska, D., Piliński, M., Rutkowski, L., *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte*. Wydawnictwo Naukowe PWN, Warszawa, 1999. ISBN 83-01-12304-4

52) Rutkowski, L., *Metody i techniki sztucznej inteligencji*. Wydawnictwo Naukowe PWN, Warszawa, 2009. ISBN 978-83-01-15731-9.

53) Smith, G. D., *Numerical Solution of Partial Differential Equations: Finite Difference Methods, 3rd ed.*, Oxford University Press, 1985.

54) Sprecher, D.A., A numerical implementation of Kolmogorov's superpositions. *Neural Networks*, 9, pp. 765-772, 1996.

55) Werbos, P. J., *Beyond regression: New tools for prediction and analysis in the behavioural sciences.* Harvard University, 1974.

56) Wiener, N., *Cybernetics or control and communication in the animal and the machine*. The MIT Press, Cambridge, 1948.

57) Wilamowski, B. M., Cotton, N., Hewlett, J., Kaynak, O., Neural network trainer with second order learning algorithms. *2007 11th International Conference on Intelligent Engineering Systems*, IEEE, pp. 127-132, 2007.

58) Wilamowski, B. M., Yu, H., Improved computation for Levenberg–Marquardt training. *IEEE transactions on neural networks*, *21*(6), pp. 930-937. 2010.

59) Yu, H., Wilamowski, B. M., Levenberg-marquardt training. *Industrial electronics handbook*, *5*(12), 1, 2011.

60) Zhang, G., Quantum-inspired evolutionary algorithms: a survey and empirical study. *Journal of Heuristics*, pp. 1–49, 2010. ISSN 1381-1231. 10.1007/s10732-010-9136-0.

SPIS TABEL

SPIS RYSUNKÓW

SKOROWIDZ