

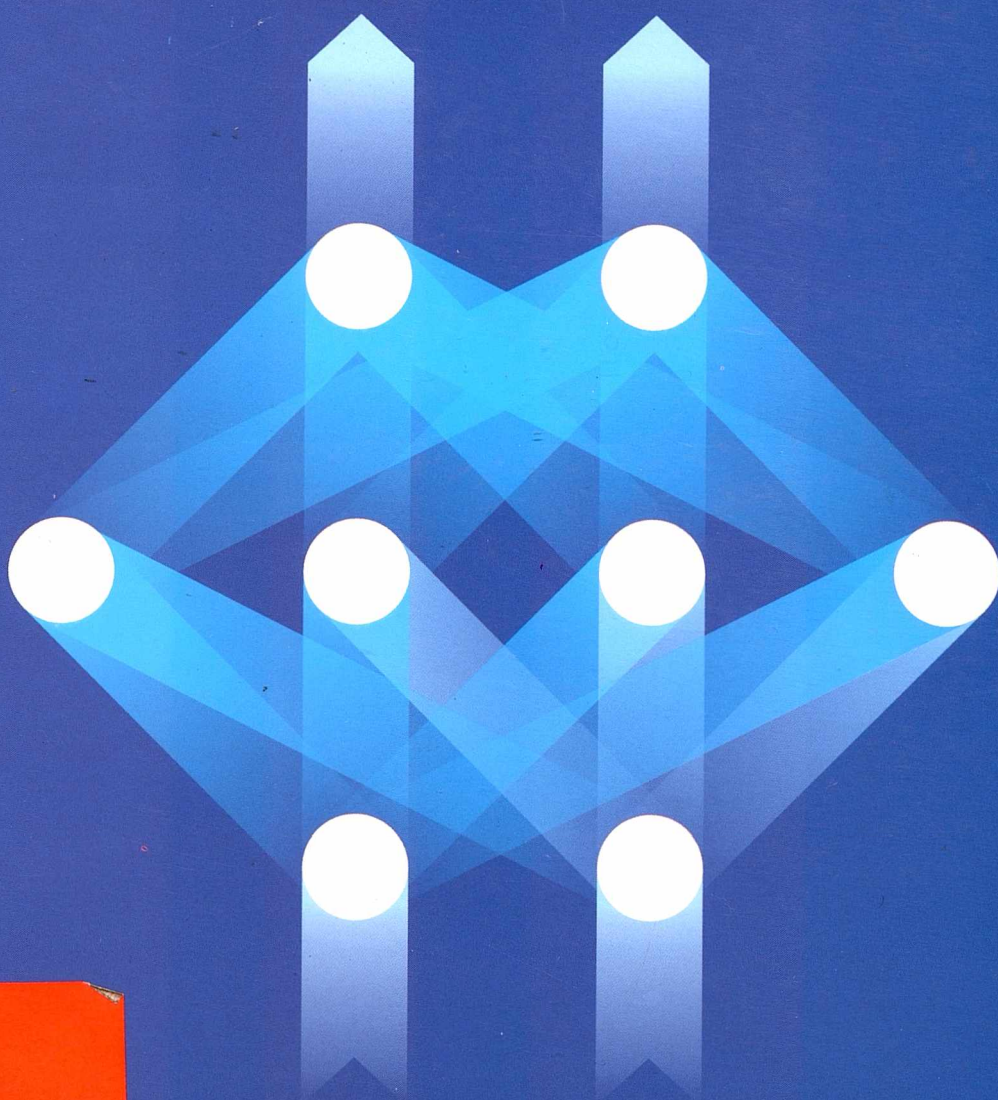
Biblioteka Główna i OINT
Politechniki Wrocławskiej



100100163269

Halina Kwaśnicka

Ewolucyjne projektowanie sieci neuronowych





Halina Kwaśnicka

**EWOLUCYJNE
PROJEKTOWANIE
SIECI NEURONOWYCH**



Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2007

147642,
Praca naukowa finansowana częściowo ze środków Komitetu Badań Naukowych
w latach 2006–2007 jako projekt badawczy Nr 3 T11C 039 30

Recenzenci

PIOTR JĘDRZEJOWICZ
JERZY JÓZEFczyk
PIOTR SZCZEPANIAK

Opracowanie redakcyjne

HANNA JUREK

Korekta

ALINA KACZAK

Projekt okładki

ZOFIA i DARIUSZ GODLEWSCY

Wszelkie prawa zastrzeżone. Żadna część niniejszej książki, zarówno w całości,
jak i we fragmentach, nie może być reprodukowana w sposób elektroniczny,
fotograficzny i inny bez zgody właściciela praw autorskich.

© Copyright by Halina Kwaśnicka, Wrocław 2007

OFICyna WYDAWNICZA POLITECHNIKI WROCLAWSKIEJ

Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

<http://www.oficyna.pwr.wroc.pl>

e-mail: oficwyd@pwr.wroc.pl

ISBN 978-83-7493-353-7



332263L/1

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr 752/2007.

Alc 2008/0039/45

*Witkowi,
Jakubowi, Mateuszowi i Agatce*

SPIS TREŚCI

Od autorki.....	7
Wstęp.....	9
1. Mózg a sztuczna inteligencja i sztuczne sieci neuronowe	14
1.1. Budowa mózgu	14
1.2. Mózg a umysł	17
1.3. Ewolucja mózgu	20
1.4. Modelowanie mózgu.....	21
2. Sztuczne sieci neuronowe.....	26
2.1. Sztuczny neuron	27
2.2. Budowa sieci neuronowych	29
2.3. Działanie sieci neuronowych	32
3. Algorytmy ewolucyjne	34
3.1. Teoria ewolucji	35
3.2. Algorytm genetyczny.....	37
3.3. Programowanie genetyczne	40
3.4. Strategie ewolucyjne.....	41
3.5. Programowanie genowe (GEP).....	42
4. Hybrydowe systemy inteligentne	49
4.1. Techniki łączenia systemów inteligentnych.....	49
4.2. Przesłanki genetycznego projektowania sieci neuronowych.....	52
5. Przegląd systemów genetyczno-neuronowych	55
5.1. Spojrzenie na problem projektowania sieci neuronowych	55
5.2. Genetyczne uczenie sieci neuronowych.....	57
5.3. Optymalizacja parametrów algorytmu uczenia i parametrów sieci.....	59
6. Algorytmy ewolucyjne jako metoda selekcji cech	62
6.1. Wprowadzenie w problematykę selekcji cech	62
6.2. Algorytm genetyczny w przygotowaniu danych uczących	65
6.3. Genetyczna selekcja cech dla NN i C4.5 – studium przypadku	68
6.3.1. Opis systemu	69
6.3.2. Przeprowadzone badania	70
6.3.3. Zestawienie i porównanie wyników	71
7. GANN – metoda genetycznego projektowania SSN	75
7.1. Ogólny schemat genetycznego projektowania sieci neuronowych	75
7.2. Kodowanie sieci neuronowych i stosowane operatory	77
7.2.1. Kodowanie bezpośrednie	78
7.2.2. Kodowanie pośrednie.....	81
7.3. Funkcja przystosowania.....	85
7.3.1. Ocena uczenia się sieci.....	85
7.3.2. Ocena rozmiaru generowanych sieci	87
7.3.3. Trzy funkcje przystosowania o różnych preferencjach.....	88
7.4. Stosowane algorytmy uczenia sieci	89
8. Badania eksperymentalne	92
8.1. Charakterystyka zbiorów uczących	92

8.2. Wyniki eksperymentów z problemami bez jawnych zależności czasowych.....	94
8.2.1. Wpływ funkcji przystosowania na efektywność metody.....	96
8.2.2. Wpływ sposobu kodowania na efektywność ewolucji sieci neuronowych.....	100
8.2.3. Ewolucja jednokierunkowych i rekurencyjnych sieci z dyskretnym czasem, dla zadania generowania sekwencji.....	102
8.2.4. Omówienie wyników eksperymentów z programem GANN	106
9. Algorytm genetyczny z efektem pleiotropowym i poligenicznym jako narzędzie projektowania NN ...	108
9.1. K-Model a projektowanie sieci neuronowych	109
9.2. Propozycja rozwiązania	110
9.2.1. Struktury genetyczne.....	110
9.2.2. Kodowanie sieci	112
9.2.3. Obliczanie przystosowania osobnika.....	114
9.3. Algorytm ewolucyjny z modyfikacjami.....	115
9.4. Przeprowadzone eksperymenty.....	120
9.5. Omówienie wyników	124
10. Programowanie genowe (GEP) w projektowaniu SSN	128
10.1. Specjalizowane operatory przeszukujące	129
10.2. Możliwości zaproponowanego podejścia.....	131
11. Podsumowanie.....	133
Literatura	138

OD AUTORKI

Algorytmy ewolucyjne, sieci neuronowe, systemy mrówkowe, sztuczne systemy immunologiczne to przykłady efektywnego naśladowania natury. Wszystkie mają swoje mocne i słabe strony, swoich zwolenników i przeciwników. Osobiście nie umiem powiedzieć, który z tych przykładów jest najbardziej ciekawy, ani najbardziej obiecujący. Podczas kolokwium habilitacyjnego dostałam pytanie: którą z metaheurez – algorytmy ewolucyjne, sieci neuronowe, systemy mrówkowe czy tabu search wybrać dla danego zadania? Odpowiedź nie jest prosta. Część z tych metaheurez wydaje się bardziej predysponowana do pewnej klasy zadań, np. sztuczne sieci neuronowe do zadania klasyfikacji, ale ogólnie? Nie wiemy. Nie mamy pewności nawet tego, że sieci neuronowe są lepszymi klasyfikatorami niż np. zbiór reguł otrzymany za pomocą algorytmów genetycznych albo systemu mrówkowego. Jaka odpowiedź na to pytanie jest poprawna? Ja odpowiedziałam wtedy, i jestem skłonna podtrzymać tę odpowiedź dzisiaj, że każda z tych metaheurez wymaga wycucia wynikającego z doświadczenia i intuicji. Powinniśmy zatem stosować tę metaheurę, którą najbardziej znamy, potrafimy najlepiej dobrać jej parametry i/lub operatory. Wybór nawet bardziej odpowiedniej metaheurezy, ale bez umiejętnie zdefiniowanych takich elementów, jak sformułowanie problemu w sposób odpowiedni dla wybranej metody, dobór operatorów (krzyżowanie, mutacja, określenie sąsiedztwa, sposób parowania feromonu, rodzaj sieci neuronowej itp.) oraz ich parametrów, najczęściej nie da nam satysfakcjonującego rozwiązania.

Niniejsza praca dotyczy dwóch metod inspirowanych naturą – uproszczonych modeli ewolucji biologicznej i mózgu, czyli algorytmów ewolucyjnych i sztucznych sieci neuronowych. Staram się pokazać, jak można wykorzystać algorytmy ewolucyjne dla wsparcia projektowania sieci neuronowej dla określonego zadania. Zawarte tu przemyślenia, wyniki badań i wnioski są efektem wielu lat pracy, w której, w różnym zakresie, brali udział studenci kierunku informatyki, zwykle moi magistranci. Ponieważ problem jest bardzo złożony i trudny, uznałam, że warto podzielić się swoim doświadczeniem.

W pracy skupiono się głównie na projektowaniu jednokierunkowych sieci neuronowych, choć część eksperymentów dotyczyło sieci ze sprzężeniem zwrotnym. Doświadczenie wskazuje, że ogólne wnioski wynikające z badań nad genetycznym projektowaniem jednokierunkowych sieci neuronowych są prawdziwe również w odniesieniu do genetycznego projektowania innych rodzajów sieci.

Pragnę podziękować tym, którzy współpracowali ze mną. Specjalne podziękowania należą się recenzentom tej książki, prof. *Piotrowi Jędrzejowiczowi*, prof. *Jerzemu Józefczykowi* i prof. *Piotrowi Szczepaniakowi*, których rzetelna praca przyczyniła się

do znaczącej poprawy tej książki, a ich życzliwość spowodowała, że końcowe prace nad książką były łatwiejsze. Dziękuję Im również za okazaną życzliwość i szybkie przygotowanie recenzji, miało to dla mnie duże znaczenie.

Czytelnikom zaś życzę wielu inspiracji do dalszych poszukiwań *uniwersalnej* metody projektowania sieci neuronowych.

Wrocław, maj 2007

Halina Kwaśnicka

WSTĘP

Obserwacja otoczenia ma duży wpływ na nasze codzienne życie, nawet, jeśli sobie tego nie uświadamiamy. Rozwój człowieka, podobnie jak każdego innego organizmu żywego, zależny jest od środowiska, w którym się wychowuje. Człowiek, jako organizm dobrze zaadaptowany, nie tylko dostosował się do różnorodnego środowiska, ale w dużej mierze podporządkował sobie środowisko. Człowiek, próbując poznać mechanizm funkcjonowania ewolucji biologicznej dąży do „zrozumienia zagadki życia”¹. Darwinowski model powstawania gatunków wskazuje na dobór naturalny, jako na sprawcę pojawiania się nowych gatunków. Przypadkowe zmiany mutacyjne wspomagają ten dobór. Można się zgodzić ze stwierdzeniem, że zdolność do adaptacji powstaje wskutek zależności przyczynowej pomiędzy tym, jak sprawnie coś działa a sposobem jego powstawania². Naszkicowana przez Darwina teoria ewolucji uległa znacznym modyfikacjom, nadal jednak trwają spory nie tylko o sam mechanizm ewolucji, ale i o definiowanie niektórych pojęć. Przez ewolucję rozumie się dzisiaj zmianę właściwości populacji organizmów, która wykracza poza życie pojedynczego organizmu. Natomiast przez adaptację rozumie się zmianę struktury bądź funkcji organizmu, która zwiększa szanse jego przeżycia i wydania potomstwa. Wydanie potomstwa jest równoznaczne z rozpowszechnianiem swojego zestawu genów (genotypu) w populacji.

Możemy postawić pytanie: Czy istnieje sukces ewolucji? Weźmy, np. wrotka. Jego kolonie zamieszkują wszystkie strefy klimatyczne zarówno w tropikach, jak i wodach Antarktyki, w dorzeczu Amazonki i górskich jeziorach. Jeśli potraktujemy sukces ewolucyjny jako zdolność gatunku do przetrwania, to z pewnością przypadek wrotka jest sukcesem ewolucyjnym (reprodukcyjnym). Przez sukces możemy rozumieć tu organizm żyjący od dawna, we wszystkich środowiskach ziemi. Wrotek nie ma życia rodzinnego, rozmnaża się przez podział. W okresie wyjątkowo niesprzyjającym wrotki przyjmują formę przetrwalnikową, odporną zarówno na bardzo wysokie, jak i niskie temperatury. Ponadto forma przetrwalnikowa wędruje z wiatrem, jak napotka

¹ J. Smolińska: *Dwie ewolucje*, http://www.racjonalista.pl/kkDwie_ewolucje.html.

² N. Etcoff: *Przetrwają najpiękniejsi*, Warszawa. WAB, 2002. Za Smolińską (wyżej).

sprzyjające warunki, rozpoczyna kolonię³. Można także uznać, że człowiek również jest przykładem sukcesu ewolucyjnego, zamieszkuje różne obszary, różne strefy klimatyczne. Jednakże człowiek nie jest tak „pokornym” organizmem, jak wrotek. Człowiek chlubi się czymś, co nazywane jest *inteligencją*. Dzięki niej, człowiek potrafi się zachować nieprzewidywalnie, zaskakująco. Ze tę *inteligencję* człowiek płaci odpowiednio wysoką cenę: duży mózg i długi czas opieki, aby ów mózg osiągnął pełną funkcjonalność, dlatego człowiek rodzi się niesamodzielny, nie ma szans przeżycia bez wieloletniej opieki. Zarówno sukces, jak i styl życia wrotka można łatwo wyjaśnić, natomiast to, co robi i jak się zachowuje człowiek, nie jest łatwe do wyjaśnienia. Fakt istnienia ludzkiej inteligencji jest nadal zagadką, mimo wieloletnich, intensywnych badań. Wiadomo bowiem, że adaptacyjna potrzeba nie wymusza pozytywnych zmian; dobór działa na zasadzie selekcji negatywnej – niekorzystne zmiany są eliminowane, ale nie ma stymulacji zmian pozytywnych. Nie ma zgody wśród uczonych, że złożony mózg jest niezbędny do przetrwania⁴. Część naukowców (tzw. funkcjonałści) uważa, że natura nie tworzy żadnych bytów nadaremnie, dlatego ludzki mózg nie jest przypadkowym, niepotrzebnym obciążeniem człowieka.

Warto tu zwrócić uwagę na opinię takich zoologów, jak Richard Alexander czy Nicholas Humphrey. Twierdzą oni, że „Jedynie istnienie innych ludzi mogło stać się przyczyną naszej własnej ewolucji”⁵. Oznacza to, że człowiek przede wszystkim rywalizuje z innymi ludźmi⁶. Dlaczego na ten element zwracam uwagę? Otóż ten właśnie element jest podstawą działania algorytmów ewolucyjnych – metody szukającej satysfakcjonujących rozwiązań, wykorzystującej paradygmat ewolucji biologicznej. Algorytmy ewolucyjne są szeroko stosowaną metodą rozwiązywania problemów, które są trudne do rozwiązania metodami algorytmicznymi, dającymi w akceptowalnym czasie dokładne rozwiązanie, lub tam, gdzie nie znamy metody rozwiązania zadania. Dają one w akceptowalnym czasie zadowalające rozwiązanie⁷.

Powszechnie znane i stosowane sieci neuronowe są znane jako paradygmat biologicznego mózgu. Są wprawdzie bardzo uproszczonym modelem mózgu, ale znajdują

³ K. Szymborski: Ewolucja inteligencji, www.racionalista.pl/kkEwolucja_inteligencji.html.

⁴ S. J. Gould uważa, że mając skomplikowany, produkujący różne myśli mózg, powodujący, że jesteśmy inteligentni, znajdujemy dla niego różnorodne zastosowania, np. tworzenie poezji czy wymyślanie teorii względności. Obie z tych czynności nie mają znaczenia adaptacyjnego, jest to efekt „dryfu genetycznego”.

⁵ Za K. Szymborski: *Ewolucja inteligencji* http://www.racionalista.pl/kkEwolucja_inteligencji.html.

⁶ Są przynajmniej dwie sprzeczne teorie. Jedna, znana jest jako hipoteza makiawelicznej inteligencji. W uproszczeniu głosi ona, że miarą inteligencji jest to, na ile człowiek jest w stanie oszukać innych i wykryć oszustwo innych. Jest to teoria A. Humphrey’ego. Inna teoria, której zwolennikiem jest G. Miller, głosi, że ten stosunkowo duży mózg u człowieka powstał jako efekt doboru płciowego, jest przydatny w grze miłosnej pomiędzy mężczyzną a kobietą. Hipoteza ta jest o tyle ciekawa, że zgodnie z nią istnieje powiązanie pomiędzy sukcesem reprodukcyjnym a wielkością mózgu.

⁷ Zachęcam do zapoznania się z pozycjami: Michalewicz Z., Fogel D.B.: *Jak to rozwiązać, czyli nowoczesna heurystyka*, WNT, Warszawa 2006 oraz Michalewicz Z., Schmidt M., Michalewicz M., Chiriac C.: *Adaptive Business Intelligence*, Springer, Berlin 2006.

wielorakie zastosowania⁸. Opracowano wiele różnych architektur sieci neuronowych, które potrafią się nauczyć rozwiązywać zadane zadanie. Opracowano różne algorytmy uczenia, dostosowane do poszczególnych architektur sieci neuronowych. Opracowano też specjalizowane sieci neuronowe do modelowania mózgu, czy też pewnych specyficznych funkcji mózgu⁹. Jednakże najpopularniejsze są jednokierunkowe sieci neuronowe. Przyczyny tego mogą być różne:

- uczenie tych sieci jest stosunkowo łatwe,
- ta architektura sieci neuronowych jest najbardziej znana,
- najłatwiej jest korzystać z doświadczenia innych użytkowników i badaczy sieci neuronowych podczas projektowania sieci dla własnego zadania.

Ostatni z czynników jest bardzo ważny. Chcąc zaprojektować sieć neuronową dla swojego zadania badacz musi proponować architekturę sieci: tzn. liczbę neuronów i sposób ich powiązania. Odpowiedni dobór sieci neuronowej ma decydujący wpływ na jej możliwości i jakość działania. Skąd badacz ma wiedzieć, jaką sieć neuronową dobrać do danego zadania? Musi tu polegać na wiedzy i doświadczeniu swoim i innych badaczy.

Powiedzieliśmy wcześniej, że źródło inteligencji, czyli mózg człowieka powstał jako efekt ewolucji biologicznej. Człowiek podgląda naturę w różny sposób. Opracowano paradygmat maszynowego uczenia przez obserwację¹⁰. Obserwacja eksperta w miejscu jego pracy jest zalecana do pozyskiwania wiedzy dla systemów ekspertowych. Inżynier wiedzy obserwuje pracę eksperta dziedzinowego na stanowisku jego pracy, w ten sposób, podglądając go, zapoznaje się z metodami stosowanymi przez eksperta w jego pracy¹¹. Naśladowane są i to z sukcesem, takie procesy naturalne, jak zachowania kolonii owadów, np. mrówek do opracowywania metod zespołowego rozwiązywania zadań¹² czy działanie naturalnego systemu immunologicznego organizmów żywych¹³. Widać, że czerpanie z rozwiązań wynalezionych przez naturę nie jest czymś niespotykanym. Powróćmy do sieci neuronowych. Skoro nie istnieją dobre,

⁸ Różnorodność zastosowań sieci neuronowych pokazana jest m.in. w Markowska-Kaczmar U., Kwaśnicka H. (red.): *Sieci neuronowe w zastosowaniach*, Oficyna Wydawnicza PWr. Wrocław 2005.

⁹ Na przykład M. Huk: *Modelowanie zjawiska kierunkowania uwagi przy pomocy sztucznych sieci neuronowych w zadaniu klasyfikacji*, Praca doktorska zrealizowana pod kierunkiem H. Kwaśnickiej. Instytut Informatyki Stosowanej, Politechnika Wrocławska 2007.

¹⁰ Patrz P. Cichosz: *Systemy uczące się*. WNT, Warszawa 2000.

¹¹ Jest to tzw. śledzenie procesu: stosując tę technikę inżynier wiedzy powinien pozyskać informację stosowaną przez eksperta przy podejmowaniu decyzji, oraz poznać sposób, w jaki ekspert przetwarza tę informację (wiedzę). Więcej w Kwaśnicka H.: *Sztuczna Inteligencja. Rozwój, Perspektywy, Systemy ekspertowe*. Wyższa Szkoła Zarządzania i Finansów, Wrocław 2005.

¹² Na przykład Dorigo M., Maniezzo V., Colomni A.: *The Ant System: Optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man and Cyb., Vol. 26, No.1, s. 1–13, 1996; Boryczka U., Boryczka M.: *Multi-cast ant colony system for the bus routing problem*. W: *Metaheuristics: computer decision-making*. Kluwer Academic Publishers, s. 9–125, 2004.

¹³ Sztuczne systemy immunologiczne, np. Wierzchoń S.: *Sztuczne Systemy Immunologiczne. Teoria i Zastosowania*. Akademicka Oficyna Wydawnicza Exit, Warszawa 2001.

powszechnie akceptowane metody projektowania architektury sieci neuronowych, to może warto pozwolić sztucznej ewolucji dobrać odpowiednią dla danego zadania sieć neuronową. Tematyka ta jest przedmiotem licznych prac na świecie. Jest też głównym tematem niniejszej pracy.

Po dyskusjach w czasie spotkań przy okazji krajowych i międzynarodowych konferencji zdecydowałam się na umieszczenie wprowadzenia (na poziomie ogólnym) do funkcjonowania, ewolucji i modelowania mózgu (rozdział 1.). Rozdział ten ma na celu uświadomić „...czytelnikom, że sztuczne sieci neuronowe są bardzo uproszczonymi modelami pewnych funkcji mózgu oraz to, że mimo pewnego postępu, mózg pozostaje w dużym stopniu tajemnicą dla człowieka”¹⁴. Kolejne trzy rozdziały dotyczą krótkiego wprowadzenia do sieci neuronowych, algorytmów ewolucyjnych i systemów łączących w sobie różne techniki – systemów hybrydowych. Kolejny, piąty rozdział to przegląd hybrydowych systemów genetyczno-neuronowych.

Algorytmy genetyczne mogą być wykorzystywane do przygotowywania danych do uczenia sieci neuronowych, jako metoda uczenia sieci neuronowych, jako metoda doboru struktury sieci neuronowych, doboru topologii sieci neuronowej, rozumianej jako architektura i zestaw wag połączeń. Po przemyśleniach elementy te pokazane są w pięciu rozdziałach. Rozdział szósty pokazuje działanie algorytmu genetycznego jako selektora istotnych cech opisujących obiekty, które mają być klasyfikowane przez projektowaną sieć neuronową. Sieć neuronowa dobrana jest metodą tradycyjną, tzn. przyjęto najbardziej popularną jednokierunkową, warstwową sieć neuronową; liczbę neuronów dobrano eksperymentalnie. Najbardziej wszechstronne badania przedstawiono w postaci dwóch kolejnych rozdziałów: rozdział siódmy przedstawia zaprojektowany i wykorzystany system do badania skuteczności projektowania sieci neuronowych w zależności od różnych sposobów kodowania sieci w osobniku (pośrednie i bezpośrednie), przyjętego sposobu uczenia sieci (genetycznego i metodą wstecznej propagacji błędu) i różnego ciśnienia selekcyjnego w czasie ewolucji, wynikającego z różnego zdefiniowania funkcji przystosowania. Wzięto też pod uwagę dwa typy zadań: typowe dla jednokierunkowej sieci neuronowej oraz odpowiednie dla rekurencyjnej sieci – zadanie z jawnymi zależnościami czasowymi. Wnioski wynikające z wykonanych obszernych badań zamieszczone są w rozdziale ósmym. Z uwagi na czytelność i objętość książki, szczegółowe wyniki badań nie są cytowane. Skupiono się nie na konkretnych wynikach, lecz na wnioskach z nich wynikających. W zamierzeniu autorki, tak syntetyczne przedstawienie wyników pozwoli czytelnikom na skorzystanie z tych doświadczeń w przypadku samodzielnej próby stosowania algorytmów ewolucyjnych, nie tylko do projektowania sieci neuronowych. Rozdział ten jest podzieleniem się z czytelnikami wiedzą wynikającą z doświadczenia, potrzebną do efektywnego stosowania algorytmów ewolucyjnych.

Jak już wspomniano, algorytmy ewolucyjne są bardzo uproszczonym modelem ewolucji biologicznej. Istnieje wiele podejść, stosujących różne sposoby kodowania

¹⁴ Fragment zdania z recenzji prof. P. Szczepaniaka.

i różne mechanizmy różnicujące osobniki w populacji. Jednym z takich podejść jest zaproponowany i opisany we wcześniejszej pracy autorki K-model¹⁵. Nie byłoby ciągłości pracy, gdyby nie podjęto badań nad użytecznością tego podejścia do projektowania sieci neuronowych. Tematyce tej poświęcony jest rozdział dziewiąty. W dziewiątym rozdziale przedstawiono próbę wykorzystania stosunkowo nowego podejścia do kodowania osobników w programowaniu genetycznym – Gene Expression Programming (GEP)¹⁶ do projektowania sieci neuronowych. Rozdział ten, w odróżnieniu od czterech poprzednich, jest w całości omówieniem pracy autorki podejścia zwanego GEP do tego zadania, C. Ferreiry¹⁷. Z uwagi na „młody wiek” tego podejścia zdecydowano się na jego pokazanie w tej pracy.

Pracę kończy rozdział podsumowujący. Zawiera on ogólne konkluzje wynikające z przeprowadzonych badań. Sformułowano pewne sugestie, czy też zalecenia, dla potencjalnych użytkowników ewolucyjnego projektowania sieci neuronowych. Dotyczą one takich problemów, jak wpływ sposobu kodowania potencjalnych rozwiązań i wyboru funkcji przystosowania (oceny osobników) na efektywność i skuteczność procesu ewolucji. Wskazano również kierunki dalszych badań nad automatyczną metodą generowania sztucznych sieci neuronowych.

¹⁵ H. Kwaśnicka: *Obliczenia ewolucyjne w sztucznej inteligencji*. Oficyna Wydawnicza PWr., Wrocław 1999.

¹⁶ Warto zwrócić uwagę, że w literaturze polskojęzycznej nie ma jeszcze ustalonego polskiego odpowiednika GEP, w niniejszej pracy, po konsultacji z recenzentem, prof. Piotrem Szczepaniakiem, będziemy używać nazwy *Programowanie Genowe*.

¹⁷ Doświadczenia autorki niniejszej książki z GEP to niezbyt udana próba zastosowania GEP do generowania reguł klasyfikacyjnych (w odróżnieniu od zastosowania algorytmów genetycznych) i dość udana próba zastosowania GEP do ewolucji ruchu postaci w 3D: Kwaśnicka H., Woźniak P.: *EMOT – Evolutionary Approach to 3D Computer Animation*. TASK Quarterly Journal on Advances in Artificial Intelligence, Scientific Bulletin of the Academic Computer Center in Gdansk, Poland. Vol. 11, Issue 1–2, 2007, s. 71–86.

1. MÓZG A SZTUCZNA INTELIGENCJA I SZTUCZNE SIECI NEURONOWE

Większość czytelników zapewne zgodzi się ze stwierdzeniem Boguszewskiego¹⁸, że mózg należy do najbardziej tajemniczych organów, świadczy o tym zarówno jego złożona budowa, jak i fakt, że nie do końca wiadomo, co to jest świadomość, jaka jest jej natura, jak działa umysł. Na pytanie „Czy mózg równa się umysł?” Boguszewski odpowiedział: „Tak, chociaż mózg to hardware, na którym działa umysł, czyli software. Wiadomo, że w przypadku ludzi te dwie rzeczy są nierozdzielne tak jak w komputerze”.

Czy człowiek, korzystając z ludzkiego umysłu, może poznać mózg, czyli czy możliwe jest, by człowiek poznał samego siebie? Oczywiście, pytanie to nie ma jednoznacznej odpowiedzi, wspomniany Boguszewski uważa, że jest to kwestia czasu (Boguszewski 2003), choć można też spotkać większych sceptyków w tej dziedzinie. Badania mózgu są prowadzone bardzo intensywnie, nasza wiedza o tym fascynującym organie przyrasta lawinowo¹⁹.

W niniejszym rozdziale postaramy się nieco przybliżyć budowę mózgu, relacje między mózgiem a umysłem oraz ewolucję mózgu.

1.1. BUDOWA MÓZGU²⁰

Mózg to część centralnego układu nerwowego, znajduje się w czaszce, umożliwia myślenie, pamięć i emocje. Encyklopedycznie *mózg* to „największe skupienie tkanki

¹⁸ Profesor Paweł Boguszewski z Instytutu Biologii Doświadczalnej imienia Marcelego Nenckiego w Warszawie, w wywiadzie z dnia 10.03.2003, przeprowadzonym z okazji „Światowego Tygodnia Mózgu” przez Krzysztofa Grzesiowskiego i Henryka Szrubarza w programie Sygnały Dnia w Polskim Radiu, program I.

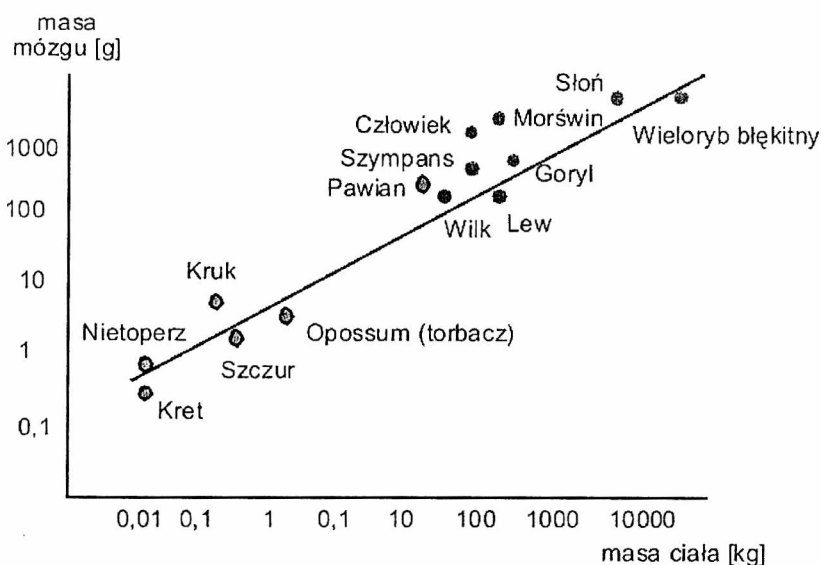
¹⁹ 90% wiedzy o naszym mózgu to są badania z ostatnich 20 lat (wg powyższego wywiadu).

²⁰ Na podstawie informacji zawartych na stronie www.W.Ducha (Duch 2007).

nerwowej u kręgowców, [...] zespół najwyższych ośrodków nerwowych, materialne podłoże zjawisk psychicznych”²¹. Z chemicznego punktu widzenia, mózg składa się w 80% z wody. Neurony są wypełnione wodą, woda jest też środowiskiem, w którym one żyją.

Ludzki mózg to szara, miękka tkanka, u dorosłego człowieka waży średnio 1375 gramów, stanowi około 2% masy ciała, u kobiet jest o około 150 gramów lżejszy. Warto zwrócić uwagę, że u niemowlęcia mózg waży średnio 350 gramów, co stanowi aż 12% całkowitej wagi ciała. Mózg już w trzecim roku życia ma moc elektryczną około 25 watów, u dorosłego człowieka mózg stanowiący 2% masy ciała zużywa 20% całkowitej energii.

Człowiek, istota o najwyższym stopniu rozwoju (spośród znanych nam istot) ma stosunkowo duży mózg, tylko walenie i słonie mają większy, jednakże nie przekłada się to na ich inteligencję²². U ssaków występuje mniej więcej stała proporcja wagi mózgu do wagi ciała. Na rysunku 1 przedstawiono w sposób jakościowy zależność masy mózgu od masy ciała u zwierząt i człowieka.



Rys. 1. Stosunek masy mózgu do masy ciała wybranych zwierząt (wykres poglądowy)

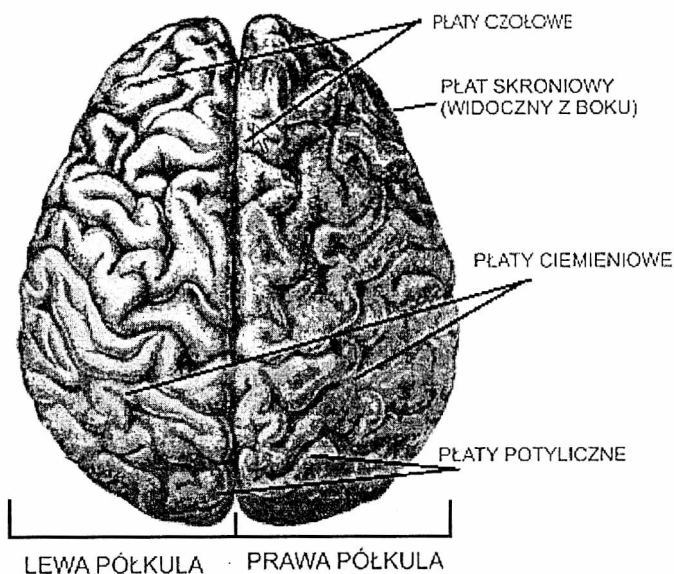
Zależnością pomiędzy wielkością mózgu a inteligencją zajmował się również Hebb (za Scot 1999). Mózg człowieka jest trzy do pięciu razy mniejszy od mózgow wieloryba i słonia, ale przecież człowieka nie uznajemy za mniej inteligentnego od tych zwierząt. Można znaleźć argumenty za tym, że stosunek masy mózgu do masy ciała ma jakieś znaczenie, ale są też i kontrargumenty – np. mózgi wielu wybitnych

²¹ *Encyklopedia Popularna*, PWN, Warszawa 1992, s. 532.

²² Microsoft ® Encarta ® Encyclopedia 2002. © 1993-2001 Microsoft Corporation.

osób były o około 20 gramów lżejsze od przeciętnej wagi mózgu człowieka, a mózgi „osób tępych” bywają cięższe. Mimo zatem istotności rozmiaru mózgu, co jest takiego w ludzkim mózgu, że przewyższa on mózgi innych ssaków? Hebb sugeruje, że są to neurony hamujące, które mogą pełnić funkcję „ukierunkowania procesu myślenia” (Scot 1999, s. 102). Względna liczba neuronów hamujących wynosi u człowieka 75%, podczas gdy np. u małpy jest to 45%, a u kota 35%.

Mózg ma złożoną strukturę. Jego górna część to *kresomózgowie*, dzieli się ono na dwie półkule, płaty ciemieniowe i skroniowe oraz płat czołowy i potyliczny (rys. 2). W tylnej części znajduje się mózdzek, za nim jest most i rdzeń przedłużony oraz wiele innych skomplikowanych elementów.



Rys. 2. Główne elementy mózgu (rysunek wykonany na podstawie Encyklopedii PWN)

Mózg składa się z około 40 mld neuronów, z czego 30 mld jest w mózdzku, 8 mld w korze mózgowej, a 2 mld to pozostałe neurony. Poszczególne komórki nerwowe łączą się poprzez tzw. synapsy, przekazując sygnały przez sieć komórek. Sygnały te są przetwarzane przez poszczególne komórki. Mózg to płatanina połączeń nerwowych, jest ich około 2×10^{13} – to ogromna liczba. Jeśli każdą synapsę potraktujemy jako bit, to pojemność naszego systemu nerwowego szacowana jest na 50×10^{12} terabitów. Liczba stanów mózgu jest praktycznie nieograniczona. W Duch_w możemy znaleźć następujące dane liczbowe charakteryzujące nasz mózg: Świadomy przepływ informacji wzrokowej to około 5000 bitów/sekundę, pozostałe zmysły – 100 bitów/sekundę. W ciągu 60 lat życia przepływa około 10 terabitów informacji.

„Chaos w małej skali, porządek w dużej” – to zdanie zawarte w Duch_w dobrze charakteryzuje ten skomplikowany narząd, jakim jest mózg człowieka.

1.2. MÓZG A UMYŚŁ

Powinniśmy tu odpowiedzieć na pytanie, czym jest umysł, jaki jest jego związek z mózgiem. Jednak odpowiedź na nie jest bardzo trudna. „Zdumiewająca hipoteza” postawiona przez Cricka mówi, że „Ty, Twoje radości i smutki, Twoje wspomnienia i ambicje, Twoje poczucie tożsamości i wolna wola, nie są w rzeczywistości niczym innym niż sposobem, w jaki zachowuje się ogromny zbiór komórek nerwowych i związanych z nimi cząsteczek” (Crick 1997, s.18). Krócej tę samą hipotezę ujmuje Duch (Duch 2006a): „Umysł jest częścią tego, co robi mózg”²³. Przedstawmy na początek – bardzo pobieżnie – jak działa mózg.

Zarówno kresomózgowie, jak i mózdzek pokryte są substancją szarą, zwaną *korą mózgową* i *korą mózdku*. Właśnie w tej substancji wykonują się różne zadania obliczeniowe. Substancja biała to włókna nerwowe, przez które komunikują się różne elementy mózgu. Działanie mózgu bynajmniej nie jest oczywiste. Pewną niespodzianką może być fakt, że prawa półkula odpowiada za kontrolowanie lewej części ciała, a lewa za prawej. Podobnie prawa część *kory wzrokowej* (umieszczonej w płacie potylicznym, w tylnej części mózgu) kontroluje lewą część pola widzenia obu oczu, a jej lewa część – prawą część pola widzenia. Podobna rzecz dzieje się z sygnałami dźwiękowymi (prawy i lewy płąt skroniowy), ale sygnały zapachowe – prawe nozdrze związane jest z prawą korą węchową, lewe z lewą (sama kora węchowa leży w płacie czołowym, blisko ośrodka węchu)²⁴. Określone obszary *kory czuciowej* leżącej w płacie czołowym odpowiadają poszczególnym częściom ciała. „Związek ten można przedstawić w postaci „czuciowego homunkulusa” – jest to zdeformowana postać ludzka leżąca na powierzchni kory czuciowej.” (cyt. Penrose 1996, s. 415). Podobnie jest z *korą ruchową* (też płąt ciemieniowy) – kontroluje ona ruchy poszczególnych części ciała, zachodzi tu związek pomiędzy poszczególnymi mięśniami a częściami kory i ten związek też można przedstawić w postaci *ruchowego homunkulusa*.

W korze też zachodzi łączenie i analiza informacji z różnych źródeł, zapamiętywane są dane, tworzy się obraz świata, plany działania, rozumiana i generowana jest mowa. Ośrodek mowy znajduje się w jednej, lewej, stronie mózgu. Należy podkreślić, że to właśnie zdolność do porozumiewania się za pomocą mowy i rozwój języka uważa się za ‘sprawcę’ wysokiego stopnia rozwoju człowieka.

Lewa półkula naszego mózgu odpowiada za: język pisany, język mówiony, liczenie, wnioskowanie, zdolności naukowe, kontrolę prawej części ciała (rys. 3).

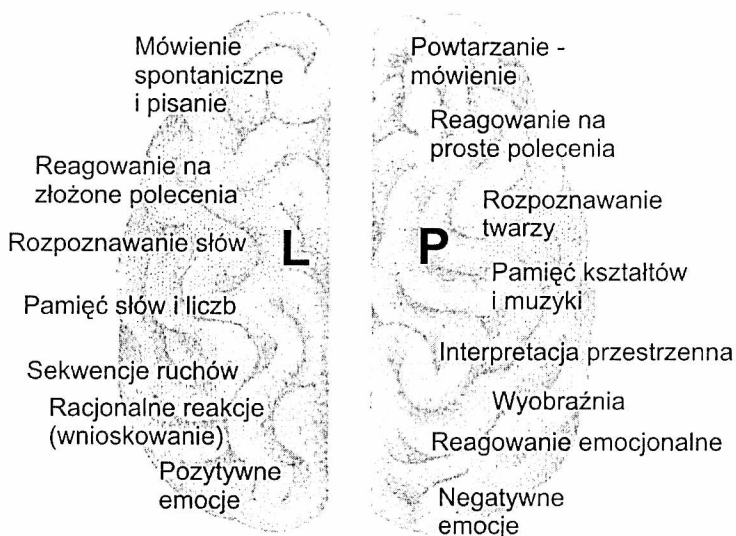
²³ Zainteresowanych czytelników zachęcam do przeczytania popularnonaukowych książek na ten temat, np.: (Crick 1997, Penrose 1996) oraz do odwiedzenia strony <http://www.fizyka.umk.pl/~duch>, na której można znaleźć liczne opracowania Włodzisława Duchy na ten temat.

²⁴ „Mózg rozwinął się z węzłów neuronów w wielosegmentowych robakach, które miały prosty odruch zwijania: dotyk z jednej strony wpływał na kurczenie się mięśni po stronie przeciwnej. Jedynie węch, rozwinięty z węzła położonego z przodu, nie ma skrzyżowanych szlaków nerwowych”. (Duch 2006a, s. 4).

Prawa półkula mózgu odpowiada za: wnikliwość, widzenie trójwymiarowe, wrażliwość na sztukę, muzykę, wyobraźnię, kontrolę lewej części ciała. Spróbujmy po kolei opisać działanie naszego kresomózgowia:

1. Odbiór danych docierających z różnych źródeł (danych wejściowych) – rejestrowanie ich w korze *pierwszorzędowej* (jest to kora wzrokowa, słuchowa, węchowa, czuciowa i ruchowa), znajdującej się głównie w tylnej części mózgu.

2. Analizowanie danych w obrębie *drugorzędowych* okolic kory (są to okolice kory pierwszorzędowej, zachodzą tam abstrakcyjne procesy) – powstają tu np. plany ruchu, które będą przetworzone na bezpośrednie rozkazy w pierwszorzędowej korze – korze ruchowej.



Rys. 3. Funkcje lewej i prawej półkuli

3. Pełne zrozumienie danych (np. mowy) w korze *trzeciorzędowej* (pozostałe obszary kory) – tu zachodzą najbardziej abstrakcyjne, wyrafinowane procesy mózgu.

4. Przekazanie zanalizowanych sygnałów do płata czołowego za pośrednictwem *pęczków łukowatych*, znajdujących się po obu stronach mózgu.

5. W trzeciorzędowej części płata czołowego tworzone są ogólne plany działania.

6. W drugorzędowej części plany te są tłumaczone na bardziej szczegółowe (np. ruchu).

7. Okolice pierwszorzędowe wysyłają impulsy do grup mięśni (jednocześnie od różnych grup).

Dane wejściowe docierają do tylnej części mózgu, a dane wyjściowe wysyłane są z przedniej części mózgu. Sygnały przechodząc przez mózg są przetwarzane i analizowane. Inaczej wyglądają dane zapachowe, ich przetwarzanie jest mniej poznane (Penrose 1996). Pozostałe części mózgu pełnią bardzo ważne funkcje, np. mózdzek, który odpowiada m.in. za precyzję ruchów, za funkcje samoczynne – oddychanie, bicie serca itp., *ciało modzelowate* odpowiadające za kontaktowanie się półkul między sobą, czy

twór siatkowy – „odpowiedzialny za ogólny stan gotowości i świadomości mózgu jako całości i jego części” (Penrose 1996, s. 419).

Analiza relacji między mózgiem a umysłem to głównie szukanie odpowiedzi na dwa pytania:

1. Jak to się dzieje, że mózg – obiekt materialny – staje się świadomy?
2. Jak to się dzieje, że świadomość – byt niematerialny – wpływa siłą woli na działanie bytów fizycznych, np. ruch ręką?

Świadomość (umysł) jest bytem niematerialnym, ale wywoływana jest przez świat materialny, na dodatek – może wpływać na ten świat materialny. Jak stwierdza Searle (Searle 1995), świadomość jest jedną z najważniejszych cech zjawisk umysłowych, trudnych do pogodzenia z materialnym (naukowym) obrazem świata. Bez świadomości nie byłyby możliwe różne inne aspekty ludzkiej egzystencji, jak język, miłość, humor i in. Jak dochodzi do jej istnienia? Nadal nauka nie radzi sobie z tym problemem: nie ma zgody badaczy w tym zakresie (Penrose 1996, Searle 1995). Penfield²⁵ twierdził, że siedzibą świadomości jest górny pień mózgu, głównie *twór siatkowaty*. Inni badacze potwierdzali, że uszkodzenie tego tworów powoduje, że człowiek traci przytomność. Z drugiej strony, twór siatkowaty to jedna z najstarszych (ewolucyjnie) części mózgu, czyli organizmy o niższym stopniu rozwoju (np. ryby) też mają świadomość. Wątpliwości odnośnie do umiejscowienia świadomości budzą badania, które wskazują, że kiedy człowiek świadomie przeżywa sen, to zwykle aktywne części tworów siatkowatych nie wykazują żadnej aktywności. Zdaniem innych badaczy, za świadomość odpowiada *hipokamp* (odpowiada on za formowanie trwałej pamięci).

Wśród filozofów i psychologów można znaleźć zwolenników tezy, że świadomość ludzka jest ściśle związana z ludzkim językiem. Przeciwno tej tezie zdaniem Penrose’a świadczy to, iż ośrodek mowy jest tylko w lewej półkuli i człowiek z uszkodzoną tą częścią mózgu byłby pozbawiony również świadomości, ale eksperymenty tego nie potwierdzają. Ciekawe badania prowadzi się na pacjentach, którzy z powodów zdrowotnych (niektóre rodzaje epilepsji) mają rozszczepiony mózg – przecięte są połączenia między jego lewą i prawą półkulą. Z badań tych wynika, że nie ma jednego miejsca, w którym „ umiejscowiona ” jest świadomość, że niektóre obszary kory mózgowej są bardziej związane ze świadomością niż inne itp.

Wielu uczonych zadaje sobie pytanie, czy jesteśmy w stanie poznać ludzki umysł za pomocą ludzkiego umysłu? Jak na razie, zadanie to nie jest jeszcze wykonane, mimo dużego zaangażowania badaczy.

²⁵ Wilder Penfield, kanadyjski neurochirurg, w latach 40. i 50. XX wieku tworzył mapę ruchowych i zmysłowych obszarów mózgu (za Penrose 1996).

1.3. EWOLUCJA MÓZGU

Za Penrose (Penrose 1996) możemy postawić pytanie: jaką przewagę w ewolucji daje świadomość? Czy zwierzęta różniące się tylko istnieniem świadomości zachowywałyby się inaczej, czyli czy te bez świadomości zachowywałyby się mniej skutecznie? Jak traktować i jaką rolę pełni *podświadomość*? Podświadomość jest czymś w rodzaju „nie-uświadomionego umysłu”. Doświadczenie wskazuje, że człowiek nieprzytomny (nie-świadomy), np. pod narkozą, jest w pewnym sensie świadomy, zasłyszane w tym czasie rozmowy mogą wpływać na późniejsze zachowanie takiej osoby, rozmowę taką osoba może sobie przypomnieć podczas hipnozy.

Ewolucja doprowadziła do wytworzenia się specyficznych struktur mózgu, które są odpowiedzialne za określone zachowania, zazwyczaj przydatne dla ich właścicieli. I tak, nie wszystkie zwierzęta rozróżniają kolory, ale te, które żywią się owocami – rozróżniają, jest im to przydatne do odróżniania dojrzałych owoców. Ssakom potrzebna jest orientacja przestrzenna, dlatego mają struktury do pamiętania map przestrzennych. Ludzie mają ośrodki mowy, których nie mają inne zwierzęta. Większość organizmów ma podobną ogólną anatomię i budowę komórkową, natomiast różnica w budowie mózgu wpływa na różnice w zachowaniach gatunków zwierząt. Ludzki mózg jest „najbardziej złożoną strukturą w znanym nam świecie”²⁶.

Nie wszystkie organizmy mają mózgi. Pierwotniaki, np. Ameby, które powstały blisko 4 mld lat temu nie mają układu nerwowego, dopiero meduza (krążkopław) ma zaczątki nerwów. Żyjące od około 500 milionów lat pierścienice mają w głowach zwój nerwowy i brzuszny pień nerwowy. Mózg, mózdzek, opuszkę nerwową i rdzeń kręgowy mają pochodzące sprzed około 440 mln lat ryby. Owady (400 mln lat) i gady (240 mln lat) mają też mózgi, konik polny ma około 16 000 neuronów, gady mają podzielony na dwie półkule mózg, który reguluje homeostazę²⁷, przetwarza dane zmysłowe i umożliwia stosunkowo złożone zachowania motoryczne. Ptaki mają złożone mózgi, z układem limbicznym²⁸, podobnie jak ssaki, ale te ostatnie mają znacznie większą korę mózgową. Ponad 200 mln lat temu pojawiły się ssaki. Ich mózgi są znacznie bardziej rozbudowane – mają układ limbiczny i rozrośniętą korę. Przodkowie człowieka mieli znacznie mniejsze mózgi, niż dzisiejszy człowiek (Duch_w):

- Australopitek (4–5 mln lat) – objętość mózgu wynosiła mniej niż 0,5 litra,
- Homo habilis (2,5 mln lat) – 0,65 l,
- Homo erectus (1,5 mln lat) – 1 l,

²⁶ Microsoft ® Encarta ® Encyclopedia 2002. © 1993–2001 Microsoft Corporation.

²⁷ Homeostaza – stan układu zapewniający mu utrzymywanie określonych wielkości w dopuszczalnych granicach (np. temperatury u stałocieplnych). *Słownik wyrazów obcych i zwrotów obcojęzycznych* Władysława Kopalińskiego.

²⁸ Układ limbiczny (układ rąbkowy) – jest to zespół struktur mózgowych stanowiących podłoże czynności popędowo-emocjonalnych, znajdują się w nim ośrodki regulujące czynności narządów wewnętrznych oraz ośrodki głodu, pragnienia, bólu, strachu, agresji i in. (za Encyklopedia PWN).

- Homo neandertalis (250 000 lat) – 1,6 l,
- Homo sapiens – 1,4 l.

U człowieka pierwsze neurony powstają już około piątego tygodnia ciąży. W dwunastym tygodniu powstają zawiązki mózdzku. Rozwój mózgu przebiega w dwóch fazach – szybka faza rozwoju oraz regresja (tzw. apoptoza). Duża część neuronów w korze mózgowej wymiera, co prowadzi do znacznej reorganizacji kory. Największe tempo wzrostu mózgu przypada na pierwszy rok życia dziecka. Zanim powstaną receptory, rozwijają się szlaki nerwowe.

Ewolucja mózgu, podobnie jak ewolucja każdego żywego organizmu, wciąż trwa. Ewolucja ta to dostosowywanie się mózgu do warunków, w których żyją ich posiadacze. Czy mózg jest niezbędny? Żachwy, gromada osłonic²⁹, kiedy są młode, mają mózg i powróżek nerwowy. W tym okresie rozwoju żachwy poruszają się, mózg jest im potrzebny do kontroli ruchu. Po osiągnięciu dojrzałości przytwierdzają się do skały, po czym stopniowo wchłaniają własny mózg – w życiu dorosłym nie potrzebują mózgu (Duch_w). W stadium dojrzałym pozostaje im tylko węzeł nerwowy, który jest potrzebny do kontroli filtrowania pokarmu. Wskazuje to na powszechnie uznawany fakt, że ewolucja raczej nie wytwarza organów, czy części ciała, które nie są potrzebne ich właścicielom. Choć wydaje się, że czasami to właśnie ewolucja doprowadza gatunek wręcz do wyginięcia. Jako przykład można przytoczyć jelenia plejstocenijskiego³⁰: „Były na przykład jelenie kopalne, które miały poroża trzymetrowej rozpiętości. Służyły one do walki o samice, były znakiem ich zdrowia i sprawności – wyznacznikiem wartości biologicznej. Koszty corocznego wytwarzania i zrzućcia tak wielkich rogów doszły jednak do granicy wydolności zwierząt.” (Dzik 2006).

1.4. MODELOWANIE MÓZGU

W analogii do terminów *silna sztuczna inteligencja* i *słaba sztuczna inteligencja*³¹ możemy mówić o dwóch podejściach do „sztucznego mózgu”: **budowa sztucznego**

²⁹ Osłonice (*Tunicata*) są podtypem zwierząt morskich (strunowców), jest ich około 1600 gatunków, są to formy osiadłe lub planktonowe, pojedyncze lub kolonijne – ogonice, żachwy, sprzagle.

³⁰ Jeleń olbrzymi (*Megaceros giganteus*) – zamieszkiwał w plejstocenie północną Europę (najwięcej jego szczątków znaleziono w Irlandii). Był wielkości łosia i miał poroże o rozpiętości 3 metry. Żywił się tym, co dzisiejsze jelenie. Z rysunków naskalnych wynika, że człowiek pierwotny polował na jelenia olbrzymiego. Wyginął prawdopodobnie z powodu niemożności zdobywania pokarmu znajdującego się wyżej (liście krzewów i drzew), co było utrudnione z powodu olbrzymiego poroża.

³¹ Silna sztuczna inteligencja (**strong AI**) – to tworzenie takich programów, które nie tylko spełnią wymagania testu Turinga, nie tylko będą modelem rozumu, lecz samym rozumem (jak każdy ludzki intelekt). Słaba sztuczna inteligencja (**weak AI**) to tworzenie komputerowych modeli inteligentnych zachowań, narzędzi, służących badaniu intelektu na podobnej zasadzie, jak inne komputerowe modele służą do badania zjawisk fizycznych, biologicznych czy ekonomicznych.

mózgu oraz *modelowanie mózgu*. Temat ten jest przedmiotem wielu prac wybitnych uczonych (Duch_w, Boguszewski 2003, Crick 1997, Duch 2006a, Duch 2006b, Jałochowski 2006, Kasperski_wa, Penrose 1996, Scot 1999, Tadeusiewicz 2005a, Tadeusiewicz 2005b), skupimy się tu na podejściu bliskim autorce. Czy możliwe jest zbudowanie sztucznego mózgu? Przywoływany już w tym rozdziale Duch, na postawione przez siebie pytanie, czy prawdziwy mózg nie jest zbyt skomplikowany, by dało się zrobić sztuczny, odpowiada przecząco³². Jego zdaniem liczą się nie pojedyncze neurony, ale moduły (kolumny korowe), a nawet proste modele wykazują wiele podobieństw do sposobu działania umysłu. Znane są modele pamięci, amnezji, halucynacji, zaburzeń mowy itp. (Duch 2006a). Problemem jest to, że świadomość – jeden z elementów umysłu – jest pojęciem bardzo nieuchwytnym. Możliwe jest takie zaprogramowanie komputera, aby on udawał, że jest świadomy, natomiast nigdy komputer nie będzie świadomy w ludzkim sensie. Pozostaje pytanie, jak szczegółowy powinien być model, aby odtwarzał istotne cechy działania umysłu? Poza tym, jakie cechy ludzkiego umysłu pozostają najbardziej tajemnicze? Wśród niewyjaśnionych cech zalicza się przede wszystkim świadomość. Ale także intuicja, emocje i wyobrażenia są specyficznymi i mało poznanymi cechami ludzkiego umysłu. Zdaniem Masao Ito (za Duch 2006a), najbardziej tajemniczą cechą umysłu człowieka jest kreatywność, a przecież kreatywność jest wykorzystywana w życiu częściej niż logika. Można znaleźć wiele definicji kreatywności, m.in., że kreatywność to zdolność znajdowania nowych rozwiązań, stosownych do sytuacji³³. Definicja ta uznaje, że kreatywność obejmuje, poza wynalazczością, również zdolność do znajdowania rozwiązań w odniesieniu do codziennych decyzji. Intuicję rozumie się jako „wiedzę bez możliwości wyjaśnienia skąd to wiemy”, w takim razie, jak ją zamodelować w komputerze? Czy komputer może mieć tak rozumianą intuicję?

Modelowaniem (uproszczonym) wyższych czynności poznawczych, myślenia, kontroli zachowania, świadomości, języka zajmuje się informatyka neurokognitywna (*computational cognitive neuroscience*) – jest to szybko rozwijająca się dziedzina (od połowy lat dziewięćdziesiątych ubiegłego wieku). Powstają głównie modele jakościowe, jako że nadal nie znamy dokładnie procesów zachodzących w mózgu. Okazuje się jednak, że nawet proste modele dają podobne rezultaty jak praca mózgu.

Badania mózgu dotyczą stanów mózgu w postaci czasoprzestrzennych pobudzeń neuronów, natomiast procesy poznawcze wykorzystują odpowiednio przetworzone sygnały dochodzące z różnych zmysłów. Cały czas pozostaje otwarty problem połączenia neuro i psyche.

Na pytanie, czy możemy zbudować sztuczny mózg odpowiada m.in. polski uczony Tadeusiewicz (Tadeusiewicz 2005b). Jego odpowiedź brzmi „to zależy”. Zależy od

³² Na podstawie dyskusji panelowej, organizowanej przez autorkę w ramach Dolnośląskiego Festiwalu Nauki, Politechnika Wrocławska, 2000.

³³ Definicja wg Handbook of Human Creativity, 1998, podana za Duchem. Zainteresowanych czytelników zachęcam do zapoznania się z pracami G. Wallas, *The art of thought* (1926), odnośnie do modelu rozwiązywania problemów w ramach psychologii postaci (Gestalt).

tego, co rozumiemy przez sztuczny mózg, jako że naturalne, w tym ludzkie, mózgi są sobie nierówne. Jeśli myślimy o konkretnym (czyimś) mózgu, to Tadeusiewicz odpowiada, że „System taki jest aktualnie niemożliwy do wykonania w postaci technicznego modelu – i to z wielu powodów.” Jako pierwszą przyczynę autor (Tadeusiewicz 2005b) podaje fakt, że choć umiemy już modelować poszczególne, pojedyncze neurony³⁴, nie jesteśmy w stanie wykonać ich tylu, aby wystarczyło do zamodelowania, czy też naśladowania naszego mózgu. Poza trudnością łączenia jednak ogromnej liczby sztucznych neuronów, pozostaje pewnie większa trudność – jak ukształtować strukturę ich połączeń, aby sieć połączeń w systemie technicznym odzwierciedlała strukturę naszego mózgu. Trzeci powód, to wciąż niewystarczająca wiedza na temat anatomii i fizjologii mózgu. Mimo ogromnego postępu badań mózgu wciąż istnieją zjawiska, które albo nie są wyjaśnione, albo też istnieje kilka konkurencyjnych teorii wyjaśniających je. Nie można więc zdefiniować schematu strukturalnego, który odpowiadałby mózgowi rzeczywistemu i nadawałby się do technicznego odtworzenia³⁵. Warto w tym miejscu zwrócić uwagę, że najbardziej złożone systemy współczesnej techniki składają się z kilkuset tysięcy elementów, a mózg składa się z wielu miliardów jednostek przetwarzających (neuronów). Tadeusiewicz podaje szacunek czasu potrzebnego na zbudowanie takiego sztucznego systemu – przy założeniu, że wytworzenie pojedynczego neuronu zajmie tylko jedną sekundę, to wyprodukowanie ich pierwszego miliarda zajęłoby około 30 lat ciągłej pracy, a stu miliardów to już trzy tysiące lat. Biorąc pod uwagę, że liczba połączeń między neuronami jest znacznie większa niż liczba samych neuronów, to wytworzenie takich połączeń w sztucznym systemie zajęłoby kolejne ... tysiące lat. Daje to pewne wyobrażenie o złożoności problemu.

O złożoności neuronu i mózgu oraz jego modelach traktuje Scot w (Scot 1999)³⁶. W rozdziale 6. autor stara się odpowiedzieć na pytanie o związek pomiędzy rejestrowanymi przez psychologię faktami a dynamiką neuronów w mózgu. Jak to się dzieje, że mamy w pamięci zakodowaną, np. babcię? Obraz babci prawdopodobnie pozyskiwany jest przez płaty wzrokowe, natomiast, np. głos – przez płaty skroniowe. Pamięć o babci wiąże się też z zapachem jej kuchni i emocjami, czyli muszą być zaktywizowane komórki mózgu, których lokalizacja nie jest znana. A przecież jest to stosunkowo proste – zapamiętać swoją babcię. Nie ma zgodnej teorii odnośnie do powstawania wspomnień, np. babci: w jaki sposób pamięć o babci powstaje z tak wielu informacji, które gromadzone są w mózgu i angażują różne jego obszary. Te informacje muszą zostać scalone, abyśmy mogli przywołać wspomnienie naszej babci. Za Scottem (Scot

³⁴ Według badań De Schuttera, 1999, bardzo dokładny model pojedynczego neuronu jest kosztowny. Aby zamodelować jedną komórkę pochodzącą z mózdzku, tzw. komórkę Purkiniego, trzeba zastosować układ 32 000 równań różniczkowych, a dostrojenie modelu wymagałoby wyznaczenia 19 200 nieznanymi wartościami parametrów (za Tadeusiewicz 2005b).

³⁵ Bardziej zainteresowanych odsyłam do książki prof. R. Tadeusiewicza, zatytułowanej *Sieci neuronowe*, można ją znaleźć pod adresem: <http://www.bg.agh.edu.pl/start.html>.

³⁶ Polecam książkę każdemu, kto jest zainteresowany naturą umysłu. Wiele trudnych zagadnień autor przybliża, używając trafnych metafor i ilustracji, co powoduje, że książka jest bardzo przystępna.

1999) warto przytoczyć, że według (Wilson, Mcnaughton 1993), maksymalna liczba jednocześnie monitorowanych komórek nerwowych u biegających szczurów wynosiła 150. Jest to i tak dużo, ale dalece niewystarczająco, jak na potrzeby wyjaśniania takich zjawisk. Autor przedstawia interesującą teorię *zespołów komórkowych*, zaproponował ją Hebb po obserwacji hodowanych w laboratorium szympanów³⁷. Scot wyjaśnia pojęcie zespołów komórkowych przez następującą analogię do społeczeństwa: mózg to miasto, a neurony to mieszkańcy miasta. Każdy mieszkaniec może być członkiem dowolnej liczby zespołów, np. partii, kościoła, drużyny sportowej itp. W każdym zespole może istnieć pewna grupa członków należących jednocześnie do innego zespołu, każdy zespół może łatwo zaktywizować swoich członków. Podobnie każdy neuron może być członkiem wielu różnych zespołów. Wracając do wizerunku babci, jego powstanie można wyjaśnić za pomocą teorii zespołów następująco: Komórki należące do płatów wzrokowych reagują na obraz babci i pobudziły komórki w płatach skroniowych, które wcześniej reagowały na głos babci. Te neurony łączą się z pamięcią o tym, co babcia robiła, jakie zapachy były w jej domu itp.

Badania nad teorią zespołów komórkowych były prowadzone stosunkowo szeroko, zdaniem Hebba, niektóre badania dostarczają silnych dowodów na jej poprawność.

Penrose w (Penrose 1996) poświęca rozdział 9. mózgowi i jego modelom. Pomiędzy tu wskazane przez Penrosa podobieństwa w działaniu komputera i mózgu, a skupimy się trochę na różnicach. Za ważną różnicę w działaniu mózgu i komputera uważa on zjawisko plastyczności mózgu. Mózg nie jest niezmiennym układem połączonych neuronów. W modelu komputerowym połączenia są stałe, podczas gdy w mózgu neurony komunikują się przez połączenia (kolbki) synaptyczne, które są dołączone do niewielkich wypustek na dendrytach – kolców dendrytycznych. Między kolbką synaptyczną a kolcem dendrytycznym jest szczelina o szerokości około 1/40 000 mm. Kolce dendrytyczne mogą się kurczyć, co daje zerwanie połączenia, mogą też rosnąć, co daje nowe połączenie międzyneuronowe. Układ komputerowy, aby był analogiczny do układu neuronowego mózgu, musiałby zmieniać się w czasie. Jedną z wiodących teorii o działaniu pamięci trwałej mówi, że takie zmiany w połączeniach stanowią mechanizm magazynowania informacji. Jeśli ta teoria jest prawdziwa, to plastyczność mózgu jest ważnym elementem jego działania. Jak szybko takie zmiany zachodzą, jaki mechanizm nimi kieruje – brakuje uznanej, wyjaśniającej teorii.

Jedną z teorii wyjaśniających mechanizm plastyczności mózgu podał w 1954 roku Hebb. Zgodnie z tą teorią istnieją specjalne synapsy, tzw. *synapsy Hebba*, mające specyficzną właściwość: niech istnieje połączenie synaptyczne między neuronami oznaczonymi jako A i B. Połączenie to wzmacnia się, gdy pobudzenie neuronu A powoduje pobudzenie neuronu B, natomiast w przeciwnym przypadku połączenie to osłabia się. Wzmacnianie i osłabianie połączenia zachodzi niezależnie od tego, czy synapsa

³⁷ Hebb zauważył histeryczną reakcję szympanów na widok glinianego modelu głowy szympansa – były przerażone, choć nigdy nie widziały ścinania głowy, nie była to ani reakcja odruchowa, ani reakcja na bodziec, ani też dla nagrody. Wniosek Hebba – naruszone zostały wysoko zorganizowane zespoły komórkowe, które kodowały oddzielną głowę jako obciętą ich własną (Scot 1999).

Hebba ma znaczenie dla pobudzenia neuronu B. Taki mechanizm jest podstawą wielu modeli matematycznych, które symulują proces uczenia się. Modele te to sztuczne sieci neuronowe lub krócej – sieci neuronowe. Mimo ich zdolności do uczenia się rozwiązywania wielu problemów, trudno jest uważać je za realistyczne modele mózgu.

Podsumowując ten rozdział, można stwierdzić, że sztuczne sieci neuronowe powstały z inspiracji neurobiologicznych. Prace, które miały istotny wpływ na powstanie tej dziedziny sztucznej inteligencji, to: model odruchów warunkowych Pawłowa, opis struktury układu nerwowego Goldiego i Cajala, model synapsy Ecclesa, model propagacji sygnałów w aksonie Hodgkina i Huxleya (Hodgkin 1951, Hodgkin, Huxley 1952). Należy tu wspomnieć o głośnej pracy McCullocha i Pittsa z 1943 roku, w której zaproponowali model neuronu – jednostkę przetwarzającą sygnały; pracy von Neumanna o związkach ludzkiego mózgu i komputerów (von Neumann 1958) oraz pracy Rosenblatta (Rosenblatt 1958) – realizacji perceptronu.

Opracowanie metod uczenia sztucznych sieci neuronowych umożliwiło zastosowanie sieci neuronowych do różnych problemów i dziedzin (Mel, Koch 1995, Duch i in. 2000, Schmitt 2002, Markowska-Kaczmar, Kwaśnicka 2005). Choć specjalizowane sieci neuronowe do specyficznych zadań znacznie odbiegają od swego pierwowzoru – sieci neuronów w mózgu (Pelc 1998, Spratling, Hayes 2000, Gupta i in. 2005), to pewne analogie biologiczne wciąż są zachowane. Sieci rozwiązujące takie zadania, jak klasyfikacji, rozpoznawania obrazów, aproksymacji funkcji, predykcji, pozyskiwania wiedzy i in. (Kurzyński 2002, Acharya i in. 2004, Winter i in. 2004, Curran i in. 2005, Hornik 1991, Shin, Ghosh 1992, Markowska-Kaczmar 2006). Tworzenie stosunkowo nieskomplikowanych sieci neuronowych do danych zadań jest jednym z kierunków badań. Drugi kierunek badań w obszarze sztucznych sieci neuronowych to budowa bardziej rozbudowanych struktur neuronowych, które mają realizować lub modelować różne funkcjonalności mózgu, np. samoświadomość czy kierunkowanie uwagi (Ols-hausen i in. 1993, Itti, Koch 2001, Tsotsos 2001, Huk 2006).

2. SZTUCZNE SIECI NEURONOWE

Wiemy już, że każdy mózg to sieć powiązanych komórek układu nerwowego, zwanych neuronami. Różne źródła szacują ich liczbę inaczej – od 11 miliardów³⁸ do 100 miliardów³⁹.

Każdy neuron ma jądro, ciało komórkowe, z którego wychodzą rozgałęziające się aksony, przez które wysyłane są sygnały z danego neuronu do innych. Każdy neuron odbiera docierające do niego sygnały z innych neuronów przez dendryty – dendryty łączą się z innymi neuronami, tworząc synapsy. Neuron może łączyć się z setkami, a nawet dziesiątkami tysięcy innych neuronów⁴⁰, tworząc sieci wytyczające szlaki nerwowe.

Kasperski w (Kasperski_wb) wyróżnia pięć etapów procesu powstawania sieci neuronowych w mózgu:

- Powstanie pojedynczych neuronów w fazie płodowej (podział komórkowy).
- Wzrost synaps powodujący łączenie się neuronów.
- Ćwiczenia poszczególnych grup połączonych neuronów w czasie nauki mowy.
- Obumieranie neuronów wraz z ciągłym ćwiczeniem i łączeniem neuronów, umożliwiając grupującym się neuronom wytworzenie szlaków istotnych dla dobrego, późniejszego działania mózgu.
- Ciągła zmiana szlaków neuronowych – poprzez stałe ćwiczenia całe szlaki ulegają zmianom, jedne są wzmacnianie, a inne zanikają, bo nie są używane.

Neurony komunikują się między sobą bardzo szybko, pozwalając ich „właścicielom” na szybką reakcję⁴¹. Mogą to robić na dwa sposoby – za pomocą sygnałów elektrycznych i biochemicznych (neuroprzekazniki). Ocenia się, że moc obliczeniowa

³⁸ M. S. Gazzaniga, *O tajemnicach ludzkiego umysłu*, s. 62 (za Kasperski_wb).

³⁹ S. Greenfield, *Mózg*, s. 103; *Tajemnice mózgu*, s. 44, 80; J. Skangiel-Kramska: *Neuroprzekazniki i ich receptory*, s. 24; ... Veggeberg: *Leczenie umysłu*, s. 18; Fischbach: *Psychika a mózg*, s. 23; A. Wróbel: *Neuron i sieci neuronowe*, s. 63 (za Kasperski_wb).

⁴⁰ W (Kasperski_wb) autor przedstawia szacunki podawane przez różne źródła, np. Duch podaje, że w korze jest 4000 połączeń na neuron, a razem jest 5×10^{13} połączeń.

⁴¹ „Kiedy pragniemy wstać z krzesła, około 200 milisekund wcześniej pojawiają się w mózgu odpowiednie „dyrektywy”, zaś wstajemy dzięki temu, że naraz zachodzi współpraca ok. 220 grup mięśniowych ciała.” [S. Lem, *Bomba megabitowa*, s. 166–167] (za Kasperski_wb).

ludzkiego mózgu wynosi około 5 Petaflopów (5×10^{15} operacji na sekundę) (Kasperski_wb).

W ostatnich latach coraz więcej uwagi poświęca się systemom uczącym się. Systemy takie są zdolne do doskonalenia swego działania na podstawie nagromadzonych doświadczeń. Systemy uczące się mają szczególne znaczenie w sytuacjach, gdy nie jest znany algorytm rozwiązania zadania. Można powiedzieć, że uczenie się jest procesem poprawiania jakości działania uczącego się systemu na podstawie doświadczeń z przeszłości (Cichosz 2000). System, który miałby wykazywać zdolności do uczenia się, musi mieć pewną wiedzę początkową, zdolność zdobywania nowej wiedzy, reprezentowania jej w swojej strukturze i takiego wykorzystania, by mógł efektywniej realizować swoje cele (Mitchell 1997, Cichosz 2000). Sieci neuronowe są jednymi z najbardziej znanych systemów uczących się (Tadeusiewicz 1993).

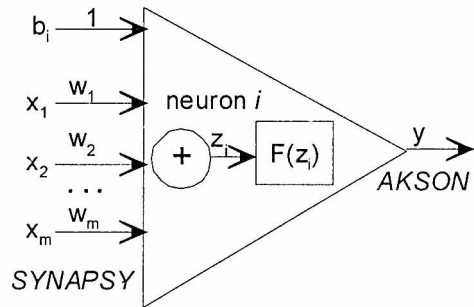
2.1. SZTUCZNY NEURON

Sztuczna Sieć Neuronowa, SNN (ang. ANN od *Artificial Neural Network*, lub krócej: NN od *Neural Network*) składa się z pojedynczych elementów zwanych neuronami. Są one ze sobą połączone, a każdemu połączeniu przypisana jest waga określająca moc tego połączenia. Każdy neuron sumuje przychodzące do niego sygnały pomnożone przez wagę połączeń. Ten sumaryczny sygnał jest następnie przetwarzany wewnątrz neuronu przez funkcję aktywacji. Tak powstały sygnał jest dalej przekazywany do innego neuronu lub, jeśli jest to neuron wyjściowy – jest sygnałem wyjściowym z sieci (rys. 4).

Wszystkie składowe wejścia x_i stanowią wektor wejściowy $\mathbf{X}$, zestaw wag w_{ij} na połączeniach dochodzących do danego neuronu stanowi wektor wag $\mathbf{W}$. Często do neuronu połączone jest dodatkowe wejście b nazywane biasem – jest to polaryzacja neuronu. Wyjście neuronu y jest zależne od sumy ważonej sygnałów wejściowych i przyjętej funkcji aktywacji F . Sygnał produkowany przez taki neuron (pomijając numer neuronu) wyraża się wzorem:

$$y = F(z) = F\left(b + \sum_{j=1}^m x_j w_j\right), \quad (1)$$

gdzie m – liczba wejść rozpatrywanego neuronu, x_j – j -te wejście neuronu, w_j – waga połączenia na j -tym wejściu neuronu, b – polaryzacja (ang. *bias*) neuronu, z – suma ważonych wejść neuronu, $F(z)$ – funkcja aktywacji, y – wartość na wyjściu neuronu.



Rys. 4. Schemat pojedynczego neuronu
(oznaczenia jak wzór (1))

Funkcja $F(z)$ nazywana jest funkcją aktywacji. Stosowane są różne typy funkcji, do najbardziej popularnych należą:

- Funkcja liniowa, $F(z) = kz$, k – stała

- Funkcje progowe

– unipolarna (skoku jednostkowego; Heaviside'a): $F(z) = \begin{cases} 1 \Leftrightarrow z > 0 \\ 0 \Leftrightarrow z \leq 0 \end{cases}$,

– signum: $F(z) = \begin{cases} 1 \Leftrightarrow z > 0 \\ 0 \Leftrightarrow z = 0 \\ -1 \Leftrightarrow z < 0 \end{cases}$,

– bipolarna (zmodyfikowana funkcja signum): $F(z) = \begin{cases} 1 \Leftrightarrow z > 0 \\ -1 \Leftrightarrow z \leq 0 \end{cases}$

– perceptronowa: $F(z) = \begin{cases} \beta \cdot z \Leftrightarrow z > 0 \\ 0 \Leftrightarrow z \leq 0 \end{cases}$

– BAM (*Bidirectional Associate Memory*): $F(z^{(t+1)}) = \begin{cases} 1 \Leftrightarrow z > 0 \\ F(z^{(t)}) \Leftrightarrow z = 0 \\ -1 \Leftrightarrow z < 0 \end{cases}$

– BSB (*Brain State in a Box*): $F(z) = \begin{cases} 1 \Leftrightarrow z > 1 \\ z \Leftrightarrow 1 \geq z \geq -1 \\ -1 \Leftrightarrow z < -1 \end{cases}$

– sinus: $F(z) = \begin{cases} 1 \Leftrightarrow z > \frac{\pi}{2} \\ \sin(\beta z) \Leftrightarrow \frac{\pi}{2} \geq z \geq -\frac{\pi}{2} \\ -1 \Leftrightarrow z < -\frac{\pi}{2} \end{cases}$

- Funkcje nieliniowe

– sigmoidalna: $F(z) = \frac{1}{1 + \exp(-\beta z)}$

– tangens hiperboliczny: $F(z) = \tanh(\beta z) = \frac{\exp(\beta z) - \exp(-\beta z)}{\exp(\beta z) + \exp(-\beta z)}$,

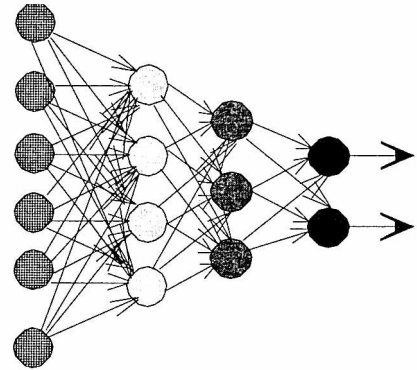
β – współczynnik określający kształt krzywej.

2.2. BUDOWA SIECI NEURONOWYCH

Neurony w sieci mogą być połączone w różny sposób, w zależności od tego wyróżniamy różne architektury sieci neuronowych. We wszystkich rodzajach sieci wyróżnia się warstwę neuronów wejściowych, do których podawana jest informacja wejściowa (są to buforory danych wejściowych), jej rozmiar jest równy wymiarowi wektora wejściowego $\mathbf{X}$ i warstwę neuronów wyjściowych, z których informacja jest odczytywana jako odpowiedź sieci $\mathbf{Y}$ na dane pobudzenia wejściowe ($\mathbf{Y}$ – wektor wyjściowy, jego składowe to wartości wyjściowe poszczególnych neuronów wyjściowych).

Pozostałe neurony (tzw. neurony wewnętrzne) mogą być połączone na różne sposoby, mogą być zorganizowane w warstwy, sygnał może przepływać tylko w jedną stronę lub mogą być sprzężenia zwrotne.

Sieci wielowarstwowe, jednokierunkowe (ang. *multilayer, feedforward*) to takie, w których neurony wewnętrzne (pomiędzy wejściowymi i wyjściowymi) są zorganizowane w warstwy, przy czym połączenia zachodzą tylko z warstwy niższej do wyższej. Rysunek 5 przedstawia przykład czterowarstwowej, jednokierunkowej, w pełni połączonej sieci neuronowej.



Rys. 5. Schemat jednokierunkowej warstwowej sieci neuronowej (dwie warstwy ukryte)

Jednokierunkowa sieć warstwowa jest najbardziej popularną architekturą. Bardziej ogólna architektura to jednokierunkowa NN (*feedforward NN*), w której istnieją połączenia pomiędzy neuronami z dowolnych warstw, ale zawsze w kierunku od wejścia do wyjścia sieci.

Inny rodzaj architektury sieci to *sieci rekurencyjne*, w których w połączeniach między neuronami występują sprzężenia zwrotne (Korbicz i in. 1994, Ossowski 1997, Tadeusiewicz 1993). W sieciach tych neurony nie muszą być pogrupowane w warstwy – mogą stanowić układ dowolnie połączonych neuronów, ale w praktycznych zastosowaniach często stosuje się architekturę warstwową, a połączenia zwrotne występują między warstwą następną a poprzednią. Przykładami sieci tego typu są sieci Hopfiel-da, BAM (*Bidirectional Associative Memory*), maszyna Boltzman i sieć Grossberga typu ART.

Sieci komórkowe (ang. *Cellular Neural Networks, CNN*) mają strukturę płaskiej lub przestrzennej siatki, w której poszczególne neurony są powiązane ze swoimi sąsiadami (promień sąsiedztwa jest wielkością założoną). Powiązania są dwukierunkowe. Architektura sieci komórkowej jest regularna, dla neuronów brzegowych wprowadza się sztuczne źródła pobudzeń o odpowiednich charakterystykach i wartościach lub łączy się je z neuronami z przeciwnych brzegów.

W literaturze spotyka się też *sieci hybrydowe*, np. CP – *Conter-Propagation* (Korbicz i in. 1994). Taka sieć składa się z dwóch różnych struktur i uczona jest częściowo bez nadzoru. W CP pierwsza warstwa przetwarzająca jest typową warstwą konkurencyjną⁴², której każdy neuron połączony jest z każdym neuronem wejściowym. Druga warstwa przetwarzająca (tzw. warstwa Grossberga) składa się z liniowych neuronów o połączeniach wstępujących od wszystkich neuronów warstwy konkurencyjnej. Uczenie takiej sieci jest wieloetapowe i polega na uczeniu poszczególnych warstw odpowiednimi algorytmami. Sieci hybrydowe mogą spełniać podobne zadania, jak np. perceptron wielowarstwowy, ich zaletą jest efektywność w działaniu.

O architekturze sieci neuronowych, tj. liczbie neuronów, sposobie ich organizacji i połączeniach, często decyduje projektant sieci, w zależności od zadania, opierając się na doświadczeniu i intuicji. Istnieją pewne ogólne zalecenia odnośnie do budowy sieci neuronowych, np.: liczba neuronów warstwy ukrytej ustalana jest jako pierwiastek z iloczynu liczby wejść i liczby wyjść sieci. Istnieją też różne metody automatycznego dobierania struktury sieci, jak konstruktywne CasCor, CasPer, FlexNet czy IncNet.

CasCor (ang. *Cascade Correlation Neural Networks*) ustala architekturę sieci w trakcie procesu uczenia. Proces rozpoczyna się tylko od warstwy wejściowej i wyjściowej. Taka prosta sieć jest uczona, jeśli nie daje dobrych wyników, to dodawane są neurony do warstwy ukrytej, aż błąd będzie akceptowalnie mały. Neurony dodawane są z tzw. puli neuronów-kandydatów. Każdy kandydat jest uczony tak, by maksymalizować kowariancję pomiędzy wartością jego wyjścia a błędem wyjścia pozostałej sieci. Do dołączenia (jako nowa, jednoneuronowa warstwa) wybierany jest najlepszy kandydat, a w powiększonej sieci uczone są tylko wagi połączeń do neuronów wyjściowych. Stosowanie do oceny kandydatów miary kowariancji powoduje, że wygenerowana sieć ma słabe właściwości uogólniania, a zamrażanie wag połączeń dochodzących do dodawanych neuronów powoduje, że często sieci te są zbyt duże (Kwok, Yeung 1993).

CasPer (ang. *Constructive Cascade Networks*), podobnie jak CasCor, stopniowo dodaje neurony do warstwy ukrytej (Treadgold Gedeon 1997). Nie stosuje miary kowariancji, daje zwykle nieduże sieci o nieco lepszych właściwościach uogólniających niż CasCor, szczególnie dla zadania regresji (Treadgold Gedeon 1997a). Wady tego podejścia to liczba neuronów w warstwie ukrytej nie jest ograniczona, struktura sieci jest nieregularna, co utrudnia implementację VLSI takich sieci. Liczba wag dodawanych do neuronów ukrytych rośnie eksponencjalnie.

FlexNet (ang. *Flexible Neural Network Construction Algorithm*) jest algorytmem tworzenia sieci neuronowych, który nie zakłada ograniczeń na liczbę ukrytych warstw

⁴² W uproszczeniu, warstwa konkurencyjna (rywalizacji) jest to warstwa wyjściowa, każdy neuron warstwy konkurencyjnej otrzymuje kompletny wzorzec podawany na wejście (ma połączenia dochodzące od wszystkich neuronów wejściowych). Neurony w warstwie rywalizacji współzawodniczą o przywilej wysłania sygnału wyjściowego, pojedynczy neuron z najmocniejszą odpowiedzią na wejściowy wzorzec jest douczany i to on generuje wyjście dla danego wzorca (Korbicz i in. 1994, Tadeusiewicz 1993, Markowska-Kaczmar, Kwaśnicka 2005)

i neuronów. Można w nim stosować różne strategie połączeń neuronów, uczenia neuronów kandydujących lub zamrażania wag. Niektóre publikacje wskazują na lepsze efekty algorytmu FlexNet niż CasCor, ale należy zauważyć, że więcej elementów algorytmu musi ustalić projektant sieci neuronowych (Mohraz, Protze 1966). Jak sama nazwa wskazuje, FlexNet jest elastyczną metodą, pozwalającą na stosowanie różnych strategii połączeń neuronów, różnych metod, różną liczbę warstw i neuronów ukrytych. Podobnie jak CasCor, FlexNet składa się z dwóch faz uczenia – głównej i kandydatów. Pozwala on na tworzenie wieloneuronowych warstw ukrytych, co może dawać lepsze właściwości uogólniania. Użytkownik może wybrać sposób, w jaki dodawane neurony mają być połączone z wejściami i wyjściami sieci. FlexNet, w odróżnieniu od CasCor, nie uczy pojedynczych kandydatów (neuronów), ale zbiory kandydatów, co może mieć pozytywny efekt zarówno na zbieżność uczenia, jak i na uogólnianie. Możliwe jest też zamrażanie wejściowych wag do neuronów ukrytych w sieci, po ich dołączeniu do sieci. Nadal wskazywana jest potrzeba rozwoju FlexNet w celu polepszania zdolności uogólniania produkowanych sieci.

Algorytm IncNet – przyrostowa budowa sieci neuronowych, pozwala na rozbudowę sieci oraz ich przycinanie tak, aby dopasować je do złożoności danych uczących (Jankowski, Duch 1999). Uczenie sieci pozwala na autokontrolę złożoności architektury sieci – w zależności od zmian wag neurony w sieci mogą być usuwane lub łączone. Powoduje to problemy związane z ustaleniem, jak często mają być sprawdzane warunki do łączenia lub usuwania neuronów. Cały proces jest stosunkowo złożony. Jako algorytm uczący wykorzystywany jest aktywny filtr Kalmana (Jankowski 1999a).

Wszystkie wymienione metody wymagają wielokrotnego procesu uczenia sieci, douczania pojedynczych neuronów (kandydatów) bądź zbiorów kandydatów (jak w FlexNet). Najlepszy kandydat (np. dający największe zmniejszenie błędu sieci lub maksymalizujący kowariancję pomiędzy wartością jego wyjścia a błędem wyjścia pozostałej sieci) jest dołączany do sieci i powiększona sieć jest znowu uczona (część wag może być zamrożona, a tylko część jest uczona). Popatrzmy na złożoność budowy oraz złożoność obliczeniową sieci neuronowych. Złożoność obliczeniowa sieci neuronowych zależy od wielu czynników, takich jak (Kroll 2003): architektura sieci (graf złożony z neuronów i połączeń), sposób przepływu sygnału przez sieć, stosowane funkcje aktywacji w poszczególnych neuronach, początkowy zestaw wag, złożoność przetwarzania w pojedynczych neuronach. Do oceny złożoności obliczeniowej sieci stosuje się ocenę złożoności sieci neuronowej, wymaganej do realizacji danego zadania.

W ogólności, złożoność sieci neuronowych składa się z dwóch części (Kon, Plaskota 2003):

1. Złożoności związanej z uczeniem sieci funkcji odwzorowującej wejścia w wyjścia, jest to tzw. złożoność informacyjna (ang. *information complexity*). Złożoność tę mierzy się liczbą przykładów wymaganych do nauczania sieci założonej funkcji z zadaną dokładnością ϵ .

2. Złożoności działania, jest to tzw. złożoność neuronowa (ang. *neural complexity*), która jest mierzona liczbą neuronów koniecznych do aproksymacji zadanej funkcji z dokładnością ϵ .

Architektura sieci analizowana jest pod kątem jej rozmiaru, głębokości i połączeń. Sam rozmiar sieci może być różnie mierzony, np. liczbą neuronów lub liczbą bramek potrzebnych do fizycznej realizacji sieci. Ważnym aspektem złożoności jest sposób przepływu sygnału w sieci. Głównie rozróżnienie to sieci jednokierunkowe i ze sprzężeniami zwrotnymi. Sieci jednokierunkowe zawsze produkują wyjście, podczas gdy rekurencyjne mogą nigdy się nie zatrzymać. Funkcje aktywacji, w połączeniu z wagami na wejściach, odgrywają kluczową rolę w decydowaniu o tym, jakie klasy funkcji mogą być obliczane przez daną sieć. Można tu wyróżnić dwie klasy funkcji – binarne (progowe) i analogowe (ciągłe).

Złożoność sieci neuronowych jest przedmiotem intensywnych badań, jednak zwłaszcza złożoność informacyjna jest wciąż w początkowej fazie (Kon, Plaskota 2000a).

2.3. DZIAŁANIE SIECI NEURONOWYCH

Po to, by sieć realizowała stawiane jej zadanie, należy ją tego zadania wyuczyć. Są dwa tryby pracy sieci: pierwszy – **tryb uczenia** i drugi – **tryb odtwarzania**. Uczenie sieci polega na takiej modyfikacji wag połączeń, aby sieć realizowała dane zadanie. Wyróżnia się **uczenie nadzorowane** (inaczej zwane uczeniem z nauczycielem), kiedy znany jest zbiór par – wektor wejściowy i odpowiadający mu wektor wyjściowy, oraz **uczenie nienadzorowane**, kiedy znany jest jedynie zbiór wektorów wejściowych. Sieć neuronowa musi w procesie uczenia nienadzorowanego sama odkryć właściwości, regularności, kategorie w danych wejściowych. Takie sieci stosowane są do grupowania wzorców podobnych. Najczęściej uczenie bez nauczyciela realizowane jest przez rywalizację pomiędzy neuronami: wszystkie neurony w sieci otrzymują te same wartości wejściowe, każdy z nich przetwarza je i produkuje wartość wyjściową. Teraz wybierany jest ten neuron, który ma największy sygnał wyjściowy, on zostaje zwycięzcą i następuje douczenie jego oraz jego najbliższych sąsiadów (mapa cech Kohonena) (Korbicz i in. 1994, Ossowski 1997).

Po wyuczeniu się sieć jest gotowa do pracy w trybie odtwarzania, w którym sieć rozwiązuje zadane jej zadanie, działa niezwykle szybko, wykorzystując swoje możliwości równoległego przetwarzania.

Większość zastosowań wykorzystuje takie zdolności sieci, jak kojarzenia (sieć spełnia rolę pamięci asocjacyjnej odwzorowującej wejście w wyjście); klasyfikacji (na wejście sieci podawany jest wzorec wejściowy, na wyjściu otrzymujemy klasę, do której ten wzorec należy). Podanie na wejście zaburzonego wzorca nie powoduje zakłóceń w pracy sieci.

W literaturze opisanych jest wiele algorytmów uczenia sieci o różnych architekturach (Duch i in. 2000, Freeman, Skapura 1992, Korbicz i in. 1994, Ossowski 1997, Tadeusiewicz 1993). Powszechnie znaną i stosowaną metodą uczenia z nauczycielem jest algorytm

wstecznej propagacji błędów. Przedstawimy tu tylko jego ideę, a odpowiednie wzory do obliczeń można znaleźć w podstawowych podręcznikach. Algorytm wstecznej propagacji błędów polega na takim doborze wektora wag, aby minimalizować funkcję błędu na wyjściu sieci. Bierze on pod uwagę sumy kwadratów błędów na poszczególnych neuronach wyjściowych (co pozwala uniknąć kompensowania się błędów dodatnich i ujemnych). Produkowane przez wyjściowe neurony błędy są propagowane wstecz do neuronów, których wyjścia są wejściami neuronów wyjściowych sieci, następnie do tych neuronów, które są wejściami przedostatniej warstwy itd. Pozwala to na określenie wielkości modyfikacji wagi każdego połączenia w sieci. Do optymalizacji sumy kwadratów błędów stosuje się metodę największego spadku, wymagane jest, aby funkcje aktywacji były różniczkowalne i ciągłe. Idea algorytmu uczenia metodą wstecznej propagacji błędów jest następująca: Prezentujemy sieci pierwszy wzorzec, obliczamy wyjście z sieci. Jeśli to konieczne, następuje korekcja wag. Dalej prezentujemy wzorce, aż do wyczerpania wszystkich wzorców ze zbioru uczącego. Takie jedno przejście przez wszystkie wzorce zawarte w zbiorze uczącym nazywa się **epoką uczenia**. Czas trwania uczenia sieci neuronowej jest najczęściej mierzony liczbą epok. Rzadziej mierzy się go w cyklach. Przez to pojęcie rozumie się prezentację pojedynczego wzorca uczącego. Dalej proces ten powtarza się, jednak aby uniknąć zapamiętania przez sieć kolejności podawania wzorców, należy pamiętać, że w każdej *epoce* wzorce podawane powinny być w losowej kolejności. Po każdej *epoce* sprawdzany jest warunek zatrzymania algorytmu uczenia, który dla sieci uczonych metodą nadzorowaną sprowadza się do badania akceptowalnego błędu. Może to też być założona liczba epok. Proces uczenia możemy zapisać w postaci ogólnego pseudokodu, składającego się z iteracji czterech kroków:

1. Powtarzaj dla wszystkich wzorców w ciągu uczącym:
 - 1.1. podaj kolejny wzorzec z ciągu uczącego i policz wyjścia sieci,
 - 1.2. policz błędy na poszczególnych neuronach wyjściowych sieci i ich sumy kwadratów,
 - 1.3. wykonaj propagację błędów wstecz, na poszczególne neurony w sieci,
 - 1.4. zmodyfikuj wagi połączeń zgodnie z uogólnioną regułą delty.
2. Sprawdź warunek zatrzymania, jeśli nie jest prawdziwy, przejdź do 1 (powtórz epokę).

Sieć najpierw poddawana jest procesowi uczenia, a następnie badana jest jej zdolność do uogólniania. Oznacza to sprawdzanie poprawności działania sieci na wzorcach, których sieć „nie widziała” podczas uczenia. W tym celu cały zbiór danych dzieli się na rozłączne zbiory: zbiór uczący i zbiór testowy. Zbiór testowy stanowi około 10% całego zbioru. Jeśli wszystkich dostępnych wzorców jest zbyt mało, możemy powiększyć nasz zbiór przez wygenerowanie z wzorców istniejących, wzorców zaszumionych.

Inna techniką testowania zdolności uogólniania sieci jest tzw. walidacja skrośna. Polega ona na podziale dostępnego zbioru wszystkich wzorców na k podzbiorów. W każdej epoce uczenia sieć uczona jest na wzorcach utworzonych z $k-1$ podzbiorów wzorców, a testowana na wyłączonym k -tym zbiorze wzorców. W następnej epoce ten zbiór dołączany jest do zbioru wszystkich wzorców uczących, a wyłączany jest inny zbiór, który posłuży do badania zdolności do uogólniania sieci. Ta technika znana jest również pod angielską nazwą *k-cross validation* lub *k-leave-out*.

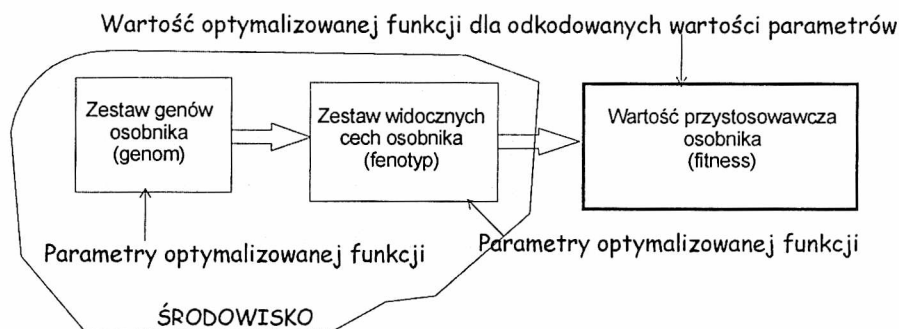
3. ALGORYTMY EWOLUCYJNE

Rozważmy takie pojęcia, jak optymalizacja, adaptacja i samoorganizacja. Optymalizacja to szukanie optimum globalnego. Nie jest ważne, co dzieje się po drodze, ważne, by w skończonym czasie dojść do optimum. Adaptacja jest to nadążanie za zmianami środowiska, otoczenia. Jeśli zmienia się otoczenie, to system próbuje dostosować się do tej zmiany. Natomiast samoorganizacja, to próba stworzenia takich wzajemnych relacji pomiędzy częściami systemu, żeby spełnić oczekiwany cel. Optymalizacja jest więc najwęższym pojęciem, w tym sensie, że system ma spełnić jeden cel – znaleźć optimum. Adaptacja jest szerszym pojęciem, ponieważ gdy zmieni się środowisko, to system ma wykryć tę zmianę i dostosować się do środowiska, czyli znaleźć nowe optimum. Natomiast samoorganizacja jawi się jako najszersze pojęcie, bo system ma spełnić założony cel, czyli tak wzajemnie dostosować powiązania między częściami, aby ten cel osiągnąć. Choć w naturze obserwujemy systemy mające zdolności do optymalizacji, adaptacji i samoorganizacji, to człowiekowi jest bardzo trudno skonstruować system komputerowy o takich właściwościach. Jest to zadanie pozostające nadal w sferze marzeń, czy oczekiwań, mimo wielu prac na ten temat. Podglądanie i próba naśladowania natury może podpowiedzieć nam pewne rozwiązania. Jednym z naturalnych procesów, który jest powszechnie naśladowany, jest ewolucja biologiczna.

Na podstawie analogii do procesów obserwowanych w ewolucji biologicznej powstało wiele algorytmów, nazywanych wspólnym mianem Algorytmy Ewolucyjne, **AE**, (ang. EA, *Evolutionary Algorithms*). Algorytmy ewolucyjne są wykorzystywane do przeszukiwania (efektywnego) przestrzeni alternatywnych rozwiązań (Arabas 2001, Fogel 1992, Galar 1990a, Goldberg 1989, Kwaśnicka 1999, Michalewicz 1966, Rutkowska i in. 1997). Według najbardziej popularnej klasyfikacji, wśród algorytmów ewolucyjnych wyróżnia się trzy główne klasy, wywodzące się z różnych korzeni i stosujące różne techniki (Arabas 2001, Goldberg 1989, Fogel 1996, Kwaśnicka 1999, Michalewicz 1966): Algorytmy Genetyczne, **AG**, (ang. GA, *Genetic Algorithms*), Strategie Ewolucyjne, **SE**, (ang. ES, *Evolutionary Strategies*) i Programowanie Ewolucyjne, **PE**, (ang. EP, *Evolutionary Programming*). Popularne Programowanie Genetyczne, **PG**, (ang. GP, *Genetic Programming*) bywa zwykle uważane za podgrupę algorytmów genetycznych.

3.1. TEORIA EWOLUCJI

Teoria, iż wszystkie organizmy powstały w rezultacie jednorazowego, krótkotrwałego aktu stworzenia utrzymywała się bardzo długo, aż do późnych lat XVIII wieku. Dopiero odkrycie gigantycznych kości w Ohio i w Europie zwróciło uwagę Francuza Georges’a Cuvier’a (1779–1832) na to, że wiele gatunków przetrwało do dziś jedynie w postaci skamieniałości – wymarły one przed milionami lat. Dzień 1 lipca 1858 roku to początek rewolucji w teorii ewolucji: Darwin i Wallace zaprezentowali wyniki swoich badań. Zrozumienie mechanizmów ewolucji biologicznej jest ważne dla ewolucjonistów, ale też dla tych, którzy chcieliby zaprojektować sztuczne systemy adaptacyjne, wzorując się na naturze.



Rys. 6. Odzworowanie genotypu na fenotyp i wartość przystosowania

Każdy organizm biologiczny rozwija się w określonym środowisku, w populacji osobników tego samego gatunku. Osobniki te różnią się między sobą, lepiej radzące sobie w środowisku (np. szybsze, mądrzejsze) zazwyczaj przeżywają i dają więcej potomstwa. Inne, te „gorsze”, też mogą przeżyć i dać potomstwo, ale mają na to zdecydowanie mniejsze szanse. W ten sposób populacja dąży do coraz lepszego przystosowania się do środowiska, które w rzeczywistości jest zmienne. Można powiedzieć, że środowisko „ocenia” poszczególne osobniki populacji danego gatunku, biorąc pod uwagę ich zewnętrzne cechy. Zestaw zewnętrznych cech osobnika nazywany jest jego fenotypem⁴³. Zewnętrzne cechy osobnika zależą od zestawu jego genów (genomu⁴⁴). Mimo iż presja selekcyjna działa bezpośrednio na fenotyp osobnika, to jego przystosowanie zależy od zestawu jego genów (rys. 6). Preferowanie lepszych osobników jest w pewnym sensie procesem optymalizacji struktur genotypowych, aby kodowane fenotypy były możliwie najlepiej przystosowane. W ten sposób cała populacja poprawia swoje przystosowanie do aktualnego środowiska (Dawkins 1995, Hoffman 1983).

⁴³ *Fenotyp* (w biologii) – zespół cech organizmu (anatomicznych, fizjologicznych, morfologicznych i in.) ukształtowanych pod wpływem czynników dziedzicznych (genów) i warunków środowiska.

⁴⁴ *Genom* (w biologii) – podstawowy, haploidalny (pojedynczy) zespół chromosomów jądra komórkowego, zawierający pojedynczy zestaw genów. U organizmów diploidalnych (większość) występują w komórkach somatycznych (tzn. wszystkich komórkach organizmu z wyjątkiem komórek rozrodczych i ich prekursorów) dwa homologiczne genomy, w gametach – jeden.

W 1932 r. Wright zaproponował pojęcie adaptacyjnej topografii do opisu przystosowania osobników i ilustracji, w jaki sposób ewolucja może przesuwac się w nowe obszary adaptacyjne (Fogel 1992). Osobnik w populacji jest zdefiniowany przez zespół potencjalnych genów, tzw. genotyp. Genotypy osobników (populacji) są odwzorowywane na ich fenotypy, które są dalej odwzorowywane na adaptacyjną topografię (krajobraz przystosowawczy). Szczyt przystosowawczy odpowiada optymalnemu fenotypowi, a ten z kolei jednemu lub większej liczbie genotypów. Ewolucja statystycznie przesuwa populację wzdłuż nachylenia krajobrazu, w kierunku szczytu przystosowawczego, w miarę usuwania przez selekcję gorzej dostosowanych osobników.

Podsumowując, żywy organizm może być postrzegany dualnie: jako jego genotyp (ukryty w kodowaniu genów) i jego fenotyp (sposób odpowiedzi organizmu na genotyp). Przestrzeń genotypowa (informacyjna) jest zwykle dużo większa od przestrzeni fenotypowej (behawioralnej) (Atmar 1986, Lewontin 1974, za Fogel 1992).

Niezależnie od chęci stworzenia sztucznego systemu o dużych zdolnościach adaptacyjnych, modelowanie ewolucji służy ewolucjonistom do lepszego poznania jej mechanizmów. Niewłaściwe symulacje, wnikające z mylenia aspektów ewolucji, dawały niezadowolające rezultaty, np. odnośnie do szybkości ewolucji⁴⁵. Nadal dla biologów istotne jest pytanie o przedmiot optymalizacji: kto zyskuje w ewolucji: pojedynczy gen, genom, cała populacja? (Dawkins 1995, Fogel 1992).

Należy postawić pytanie, czy i kiedy do rozwiązywania problemów możemy stosować metody wzorowane na ewolucji biologicznej? Odpowiedź na pytanie *czy?* brzmi: tak, bo ewolucja jest procesem o naturalnych zdolnościach do adaptacji. Na pytanie *kiedy?* należy odpowiedzieć: wtedy, gdy z jakiś powodów niemożliwe jest zastosowanie innych, dokładnych metod i jednocześnie nie jest bezwzględnie wymagane, aby znalezione rozwiązanie było optymalne. Zagadnienie optymalizacji można potraktować w sposób przypominający codzienne działanie człowieka: ocena decyzji człowieka będzie dobra, jeśli podejmuje on właściwą decyzję spośród możliwych w danym czasie. Czyli „dobroć” decyzji jest oceną relatywną w stosunku do innych, możliwych w tym czasie decyzji. Akceptując takie podejście do zadania optymalizacji, możemy wzorować się na ewolucji biologicznej. Otrzymane rozwiązanie będzie akceptowalne, być może nawet optymalne,

⁴⁵ Opozycjoniści Darwina uważają, że tak złożone struktury, jak np. oko, muszą się rozwijać jako całkowicie ukształtowane, inaczej będą nieużyteczne. Tymczasem przypadkowa zmiana – mutacja dająca tak złożone i spójne zmiany jest wręcz nieprawdopodobna. Prawdą jest, że połowa oka nie jest użyteczna, ale oko w połowie rozwinięte – tak. Na przykład oko z siatkówką, ale bez soczewki gromadziłoby światło i wykrywało ruch, co ułatwiłoby jego posiadaczom wykrywanie drapieżników. D. Nilsson i S. Pelger w 1994 r. dokonali symulacji komputerowej ewolucji oka. Symulację rozpoczęli od matematycznego modelu płaskiego obszaru komórek, dopuszczając rozmaite rodzaje „mutacji” (np. zakrzywienie kształtu obszaru komórek, zmiana wrażliwości komórki na światło itp.). Wykrywanie światła i analizowanie widzianych wzorów jest podstawą selekcji. W symulowanym czasie (odpowiadał on 400 000 lat), obszar komórek zwiął się w głęboką kulistą jamę z małym otworem podobnym do tęczówki i – co najdziwniejsze – z soczewką, w której współczynnik załamania światła zmienia się w zależności od miejsca, jak u człowieka (J. Stewart, *Liczyby natury*, Wydawnictwo CIS, Warszawa 1996, s. 33).

ale jeśli nawet takie będzie, to my i tak nie możemy tego stwierdzić. Nie mamy gwarancji, że znalezione rozwiązanie jest optimum globalnym.

3.2. ALGORYTM GENETYCZNY

Algorytm Genetyczny, AG, (ang. GA, *Genetic Algorithm*) to ewolucja sztucznych „osobników”, z których każdy jest zakodowanym potencjalnym rozwiązaniem. Osobniki te ewoluują w sztucznym środowisku, ponieważ należą do tego samego gatunku (kodują rozwiązanie tego samego zadania), dlatego konkurują o zasoby w tym środowisku. Podobnie jak w biologii, o szansach przeżycia i wydania potomstwa przez sztucznych osobników decyduje „dobór naturalny” – im lepiej jest przystosowany osobnik do danego środowiska, tym większe ma szanse przeżyć i wydać potomstwo. Potomstwo – podobnie jak w naturze – różni się od swoich rodziców dzięki działaniu specjalnie zaprojektowanych, wzorowanych na naturze, operatorów genetycznych (mutacji i krzyżowania). Środowisko odzwierciedla rozwiązywane przez algorytm genetyczny zadanie, ocena zatem przystosowania osobnika jest oceną jakości kodowanego przez niego rozwiązania. W ten sposób, w kolejnych pokoleniach populacji sztucznej osobników zaczynają dominować coraz to lepsze rozwiązania. Najlepszy osobnik, po rozkodowaniu, jest szukanym rozwiązaniem. W ostatnich latach algorytmy genetyczne stały się bardzo popularne jako narzędzie optymalizacyjne.

Ewolucja populacji jest procesem przeszukiwania przestrzeni potencjalnych rozwiązań. W procesach takich jest ważne zachowanie równowagi pomiędzy przekazywaniem najlepszych cech do następnego pokolenia, czyli wykorzystaniem dotychczas znalezionych „obietujących” rozwiązań (ang. *exploiting*) z jednej strony, a szerokim przeszukiwaniem przestrzeni (ang. *exploring*) z drugiej strony. Algorytm genetyczny umożliwia zachowanie takiej równowagi.

Chcąc rozwiązać rzeczywisty problem za pomocą algorytmów genetycznych, musimy wykonać następujące czynności (etapy):

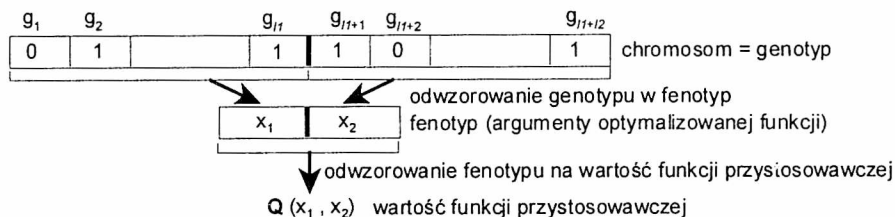
1. Dokładnie sprecyzuj problem – zakresy parametrów, ograniczenia itp.
2. Zdefiniuj problem w terminach algorytmów genetycznych: funkcję przystosowania, potencjalne rozwiązanie (chromosom), operatory genetyczne itp.
3. Utwórz początkową populację.
4. Oceń każdego osobnika w populacji.
5. Sprawdź, czy istnieje zadowalające rozwiązanie (lub czy zachodzi inny warunek zatrzymania algorytmu).
 - a) Jeśli tak, przejdź do punktu 8.
 - b) Jeśli nie: przejdź do punktu 6.
6. Utwórz następne pokolenie populacji.
 - a) Wybierz potencjalnych rodziców – lepsze osobniki są preferowane.
 - b) Zastosuj operatory genetyczne (krzyżowanie, mutację itp.).
7. Przejdź do punktu 4.
8. Zatrzymaj algorytm, zwróć jako rozwiązanie najlepsze, dotychczas znalezione rozwiązanie.

Dwa pierwsze etapy zawsze muszą być wykonane „ręcznie”, są one trudne, zwłaszcza dla niedoświadczonych użytkowników AG. Pozostałe etapy algorytmu wy-

konuje program komputerowy, można do tego wykonać własną implementację AG lub też wykorzystać komercyjne bądź dostępne w sieci programy.

Etap wstępny obejmuje: ustalenie zakresu zmienności i dokładności rozwiązań, zdefiniowanie ograniczeń, sposobu kodowania, itp.

Kodowanie rozwiązań (definiowanie chromosomu): najbardziej popularnym sposobem kodowania jest kodowanie binarne – zazwyczaj dwójkowa reprezentacja liczb.⁴⁶ Przykład reprezentacji osobnika (dla funkcji dwuwymiarowej) w prostym algorytmie genetycznym pokazany jest na rysunku 7.



Rys. 7. Reprezentacja binarna osobnika w prostym algorytmie genetycznym

Przekształcanie funkcji celu na funkcję przystosowania: W rzeczywistych zastosowaniach funkcja f , którą chcemy optymalizować zwykle nie nadaje się bezpośrednio na funkcję przystosowania. Przykłady sytuacji, kiedy optymalizowana funkcja musi być przekształcana, to:

- Kiedy na argumenty optymalizowanej funkcji f nałożone są ograniczenia.

Prostą i skuteczną metodą jest nakładanie kar na przystosowanie osobnika za przekroczenie dopuszczalnych wartości. Kara może być różna dla poszczególnych argumentów funkcji oraz może zależeć od stopnia przekroczenia zakresu.

- Kiedy algorytm genetyczny może produkować niepoprawne, nie dające się ocenić rozwiązania.

Przykładem takiego zadania jest automatyczne projektowanie sieci neuronowych, gdzie w trakcie ewolucji mogą powstawać wręcz rozwiązania nie mające sensu, np. sieci zawierające neurony, które nie mają połączeń wejściowych⁴⁷. Możliwe są dwie strategie: usuwanie niepoprawnych osobników z populacji i generowanie na ich miejsce nowych (czasochłonny sposób) oraz naprawianie powstających „degeneratów”.

- Funkcja celu ma być minimalizowana lub przyjmuje ujemne wartości.

Ewolucja „dąży” do powstawania coraz to bardziej przystosowanych osobników, naturalna zatem skłonność AG to maksymalizacja przystosowania. W wielu zadaniach

⁴⁶ Czasem lepsze efekty daje kod Graya – kolejne liczby całkowite różnią się od siebie wartością jednego bitu, np.: 0000 → 0; 0001 → 1; 0011 → 2; 0010 → 3; 0110 → 4; 0111 → 5; 0101 → 6; 0100 → 7; 1100 → 8 itd.

⁴⁷ Oczywiście, idealnym rozwiązaniem byłoby zastosowanie takich reprezentacji rozwiązań i/lub takich operatorów, które nie mogą wytwarzać niepoprawnych rozwiązań, np. specjalne operatory genetyczne dla zadań sekwencyjnych (zadanie komiwojażera), reprezentacja sieci neuronowych w postaci formuł gramatycznych (Kwaśnicka 1999, Kwaśnicka, Paradowski 2005, Lieske 1995).

należy minimalizować funkcję celu. W tej sytuacji za funkcję przystosowania Q możemy przyjąć różnicę pomiędzy ustaloną, dostatecznie dużą, stałą wartością MAX a naszą funkcją celu $f(x)$:

$$Q = \begin{cases} \text{MAX} - f(x), & \text{dla } f(x) < \text{MAX} \\ 0, & \text{dla } f(x) \geq \text{MAX} \end{cases} \quad (2)$$

Jeśli funkcja celu $f(x)$ przyjmuje ujemne wartości, to należy znaleźć możliwie małą stałą MIN, która dodana do $f(x)$ da wynik większy od zera:

$$Q = \begin{cases} \text{MIN} + f(x), & \text{dla } f(x) + \text{MIN} > 0 \\ 0, & \text{w innych przypadkach} \end{cases} \quad (3)$$

Można też stosować rankingową metodę selekcji, która jest niewrażliwa na bezwzględne wartości oceny osobników, a jedynie na ich uszeregowanie.

Selekcja osobników do reprodukcji: Można stosować różne metody selekcji, jednak wszystkie one muszą mieć jedną wspólną cechę: lepszy osobnik musi mieć większe szanse na posiadanie potomstwa, zgodnie z zasadą naturalnej selekcji „przeżywa najlepszy”. Najbardziej popularne metody selekcji, to:

- *Metoda ruletki* (nazywana również metodą stochastyczną z powtórzeniami).

Wartości przystosowania wszystkich osobników w populacji są sumowane, suma stanowi całe koło ruletki. Następnie każdemu osobnikowi przypisywany jest wycinek koła proporcjonalny do jego przystosowania. Koło ruletki jest ‘obracane’ (wybierana jest losowo liczba) i wybierany jest osobnik odpowiadający temu sektorowi na ruletce, w którym mieści się wylosowana liczba. Czynność losowania powtarza się N razy (N – rozmiar populacji).

- *Metoda próbkowania deterministycznego* – wartość oczekiwana liczby potomków dla każdego osobnika liczona jest ze wzoru:

$$E_i = Npr_i, \quad (4)$$

gdzie: E_i – oczekiwana liczba potomków i -tego osobnika, N – liczba osobników w populacji, pr_i – prawdopodobieństwo wybrania i -tego osobnika do reprodukcji, wynosi:

$$pr_i = \frac{f_i}{\sum_{j=1}^N f_j} \quad (5)$$

f_j – wartość funkcji przystosowania (fitness) j -tego osobnika.

Każdy osobnik z populacji ma tylu potomków, ile wynosi część całkowita wartości oczekiwanej E_i . Następnie osobniki są szeregowane według ułamkowych części E_i

(malejąco). Do reprodukcji dobierane są osobniki z początku tej listy, w liczbie potrzebnej do zachowania stałego rozmiaru populacji.

- *Metoda stochastyczna według reszt z powtórzeniami*

Jest podobna do poprzedniej, opiera się na liczeniu wartości oczekiwanej liczby potomków dla każdego osobnika z populacji i przydzieleniu mu całkowitej części E_i . Części ułamkowe E_i są wykorzystywane do stworzenia koła ruletki, za pomocą którego wybierane są pozostałe osobniki do reprodukcji.

- *Metoda stochastyczna według reszt bez powtórzeń*

Analogicznie do poprzednich dwóch metod, część całkowita E_i stanowi liczbę potomków i -tego osobnika, natomiast część ułamkowa jest traktowana jako prawdopodobieństwo dla rozkładu Bernoulliego, brakujące osobniki są losowane zgodnie z tymi prawdopodobieństwami.

- *Metoda turniejowa*

Wybierane są dwa osobniki (można stosować turniej więcej niż dwóch osobników) za pomocą metody ruletki, do reprodukcji wybierany jest najlepszy osobnik spośród biorących udział w turnieju.

- *Metoda rankingowa (nadawania rang)*

Osobniki w populacji są porządkowane malejąco według wartości funkcji celu. Liczba potomków osobnika zależy od jego rangi, przy czym ranga osobnika to jego miejsce w tym uszeregowaniu. Ustalana jest liczba potomków dla osobnika o najwyższej randze (max) i liczba potomków dla osobnika o najniższej randze (min), pozostałym osobnikom przydzielana jest liczba kopii proporcjonalnie do ich rangi. Metoda ta bywa krytykowana, ponieważ osłabia ona związek pomiędzy funkcją przystosowania (rangą) a funkcją celu. Mimo to, w niektórych zastosowaniach daje dobre wyniki.

Operatory genetyczne: Podstawowe operatory genetyczne to krzyżowanie (mieszanie materiału genetycznego różnych osobników) i mutacja (błąd reprodukcji osobnika). Można zdefiniować specjalizowane operatory genetyczne, wykorzystując np. wiedzę o rozwiązywanym problemie.

Jako **warunek zatrzymania** stosuje się zadaną z góry liczbę pokoleń (iteracji), liczbę pokoleń, w których nie jest znajdowane lepsze rozwiązanie lub akceptowalną wartość funkcji przystosowania (jeśli jest to możliwe).

3.3. PROGRAMOWANIE GENETYCZNE

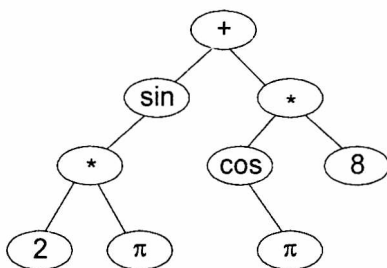
W Programowaniu Genetycznym, **PG** (ang. **GP**, *Genetic Programming*) osobnik podlegający ewolucji nie jest binarnym łańcuchem, lecz złożoną strukturą drzewiastą (Kozą 1990). Zbiór możliwych struktur jest zbiorem wszystkich możliwych kombinacji funkcji, które mogą być rekurencyjnie wyprowadzone ze zbioru funkcji

$$F = \{f_1, f_2, \dots, f_{N_F}\}$$

o mocy N_F i zbioru symboli terminalnych

$$T = \{t_1, t_2, \dots, t_{N_T}\}$$

o mocy N_T . Przykłady takich zbiorów, to: $F = (\text{AND}, \text{OR}, \text{NOT}, \text{SIN}, \text{COS}, +, -, *)$, $T = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \pi)$, przykładowa funkcja w postaci drzewa pokazana jest na rysunku 8.



Rys. 8. Przykład struktury drzewa w programowaniu genetycznym: $F = \sin(2\pi) + 8\cos(\pi)$

Koza (1990) zaproponował programowanie genetyczne jako metodę automatycznego generowania programów. Węzły w drzewie są funkcjami (lub operatorami) jedno lub wieloargumentowymi, a liście są symbolami terminalnymi. Dla takich struktur należy inaczej niż w **AG** zdefiniować krzyżowanie i mutację. Krzyżowanie polega na wymianie poddrzew między dwoma osobnikami (strukturami), węzeł przecięcia jest losowany w każdym drzewie oddzielnie. Krzyżowanie odgrywa kluczową rolę w **PG**. Mutacja bywa różnie implementowana: losowa zmiana funkcji w węźle, losowa zmiana wartości liścia, zamiana wybranego poddrzewa innym, losowo wygenerowanym, zamiana dwóch poddrzew wychodzących z jednego węzła (permutacja). Dodatkowo na drzewach można wykonywać:

- Edycję (upraszczanie) – nie zmienia się znaczenia wyrażenia reprezentowanego przez drzewo, lecz upraszcza się formę (np. x OR x zastępuje się x),
- Enkapsulację (ADF – ang. *Automatically Defined Function*) – wyodrębnianie potencjalnie użytecznego poddrzewa i nadanie mu nazwy, tak by można go było stosować jak symbol terminalny. Enkapsulacja zapewnia niepodzielność wybranego poddrzewa w wyniku krzyżowania.

Decydującą rolę w **GP** pełni dobór odpowiednich zbiorów funkcji i symboli terminalnych. Wybór zbyt dużej ich liczby powoduje znaczny wzrost przestrzeni poszukiwań, jeśli jest ich za mało, lub są źle dobrane dla danego problemu, to niemożliwe jest znalezienie satysfakcjonującego rozwiązania.

3.4. STRATEGIE EWOLUCYJNE

Początek Strategii Ewolucyjnych, **ES**, (ang. **ES**, *Evolutionary Strategy*) to lata sześćdziesiąte (za: Fogel 1995, Galar 1990b). Zadaniem **SE**, podobnie jak **AG** i **PG**, jest rozwiązywanie problemów optymalizacyjnych metodą wzorowaną na ewolucji biologicznej.

W **SE** nie stosuje się kodowania potencjalnych rozwiązań. Każdy chromosom jest łańcuchem liczb rzeczywistych: bezpośrednio wartości argumentów funkcji celu oraz prawdopodobieństw mutacji i rekombinacji. We wczesnych pracach ewoluowano populację składającą się z pojedynczego osobnika, a jako czynnik różnicujący potomka od rodzica wykorzystywano mutację. Małe zmiany mutacyjne były bardziej prawdopodobne niż duże. Potomek mógł zastąpić rodzica tylko wtedy, gdy był od niego lepszy. Takie podejście, w którym potomek konkuruje tylko ze swoim rodzicem, nazywane jest *strategią dwuelementową* ($(1 + 1)$ -ES, ang. *two-membered evolution strategy*). Ewolucja populacji μ -elementowej (gdzie $\mu > 1$) nazywana jest *strategią wieloelementową* (ang. *multi-membered evolution strategy*). W tej strategii stosowane jest również krzyżowanie. Podobnie jak w strategii $(1 + 1)$ -ES, w jednym pokoleniu produkowany jest tylko jeden potomek, który zastępuje najgorszego osobnika w populacji. Jeśli wygenerowany potomek jest gorszy od wszystkich w populacji, jest on usuwany i populacja nie zmienia się w tym pokoleniu. Taka strategia jest nazywana $(\mu + 1)$ -ES. W kolejnych latach strategię ewolucyjną rozwinęto do postaci $(\mu + \lambda)$ -ES i (μ, λ) -ES. W strategii $(\mu + \lambda)$ -ES populacja liczy μ osobników, produkujących w jednym pokoleniu λ potomków. Do następnego pokolenia wybiera się najlepsze osobniki zarówno spośród rodziców, jak i potomków. Różnica pomiędzy strategiami $(\mu + \lambda)$ -ES i (μ, λ) -ES polega na tym, że w tej ostatniej do następnego pokolenia wybierane są najlepsze osobniki tylko spośród potomków (populacja jednopokoleniowa).

Zarówno w **SE**, jak i **AG** wykorzystuje się darwinowską metodę selekcji oraz populację osobników będących potencjalnymi rozwiązaniami. Ważniejsze różnice pomiędzy **AG** a **SE** to:

- Reprezentacja osobnika: w klasycznym **AG** koduje się rozwiązania w postaci binarnego łańcucha, natomiast w **SE** ewoluują wprost rozwiązania – łańcuchy liczb rzeczywistych, powiększone o prawdopodobieństwa mutacji i rekombinacji.
- Proces selekcji i reprodukcji: w **SE** selekcja jest deterministyczna i zachodzi po procesie reprodukcji, natomiast w **AG** zachodzi przed reprodukcją (osobniki są wybierane do reprodukcji) i nie jest deterministyczna.
- W **AG** prawdopodobieństwa mutacji i rekombinacji są z reguły stałe dla wszystkich osobników i w czasie całej ewolucji, natomiast w **SE** są one różne dla różnych osobników i same podlegają ewolucji stanowiąc część chromosomu.
- W strategiach ewolucyjnych potomki nie spełniające wymaganych warunków są eliminowane z populacji, nie ma mechanizmów typu nakładanie kar.

3.5. PROGRAMOWANIE GENOWE (GEP)

Programowanie genowe⁴⁸ (**GEP** od ang. *Gene Expression Programming*) to, mówiąc w uproszczeniu, programowanie genetyczne wykorzystujące liniowe chromoso-

⁴⁸ Programowanie genowe to używana w tej pracy nazwa na *Gene Expression Programming* (GEP). Dla tego podejścia będziemy stosować angielskojęzyczny skrót GEP (aby nie mylić go z programowaniem genetycznym, zaproponowanym przez J. Kożę). Jest to stosunkowo nowe i mniej znane podejście, dlatego też zostanie tu nieco bliżej zaprezentowane. Rozdział ten napisany jest na podstawie pracy autorki podejścia, C. Ferreira (Ferreira 2002).

my (Ferreira 2002). Podstawowa różnica pomiędzy podejściami **GEP** a programowaniem genetycznym (**PG**) polega na sposobie reprezentacji osobnika:

- PG: osobniki są nieliniowymi (drzewiastymi) strukturami o różnych rozmiarach i kształtach.

- GEP: osobniki kodowane są jako liniowe łańcuchy o określonej długości, które są tożsame z nieliniowymi strukturami drzewiastymi (*Expression Trees* – ET).

Idea Gene Expression Programming ma swoje podłoże w oddzieleniu struktur odpowiadających za genotyp i fenotyp, umożliwiając efektywną konwersję pomiędzy nimi (Ferreira 2002, Kwaśnicka, Woźniak 2006). Modyfikacja genotypu jest prosta, operatory genetyczna są proste w implementacji i szybkie w działaniu. Natomiast fenotyp drzewa ET oferuje podobną funkcjonalność, jak w GP.

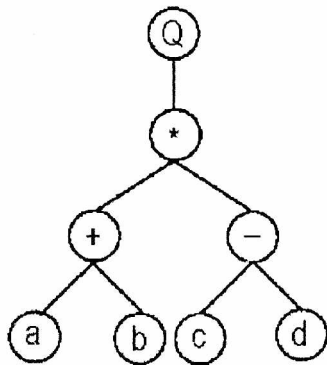
Kodowanie osobnika: Chromosom w GEP składa się z liniowego łańcucha symboli, zbudowanego z jednego bądź więcej genów. Pomimo stałej długości, łańcuch taki pozwala na tworzenie struktur o zróżnicowanym rozmiarze i kształcie. Rozważmy dla przykładu następujące wyrażenie algebraiczne:

$$\sqrt{(a+b)(c-d)} \quad (6)$$

Można je w prosty sposób przedstawić jako drzewo (*Expression Tree*) pokazane na rysunku 9, gdzie Q oznacza pierwiastek kwadratowy. Jest to fenotyp osobnika, a jego chromosom jest następujący:

Chromosom 1: 01234567 {numer pozycji w chromosomie}

$Q^{*+}abcd$ {wartość genu na danej pozycji}



Rys. 9. ET odpowiadające wyrażeniu (6) i chromosomowi 1

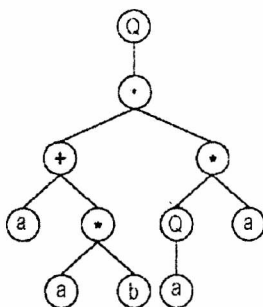
Cyfry w górnej linii oznaczają pozycję danego symbolu – funkcji lub symbolu terminalnego. Chromosom osobnika w GEP nazywamy K-wyrażeniem (od języka Karva – taką nazwę wybrała autorka metody (Ferreira 2002). K-wyrażenie łatwo utworzyć z drzewa: wystarczy odczytać drzewo od lewej do prawej i poziomami od góry

do dołu. Warto zwrócić uwagę, że taka reprezentacja różni się od spotykanych w różnych implementacjach PG wyrażeń post- i prefiksowych.

Proces odwrotny (konwersja K-wyrażenie na drzewo) jest bardzo istotny dla algorytmu, a jednocześnie prosty i efektywny. Rozważmy inne K-wyrażenie:

Chromosom 2: 01234567890
 $Q^+ * a^+ Q a a b a$

Początek wyrażenia (pozycja 0) odpowiada korzeniowi drzewa. Do każdego węzła doczepiamy tyle gałęzi, ile parametrów wymaga dana funkcja (0 dla symbolu terminalnego). Drzewo konstruujemy z góry na dół, od lewej do prawej, poziomami. Wyrażenie czytamy w prawo. Proces kończy się, gdy we wszystkich liściach znajdują się symbole terminalne (rys. 10).



Rys. 10. ET odpowiadające chromosomowi 2

Patrząc na K-wyrażenie trudno dostrzec jakiejkolwiek zalety takiej reprezentacji genomu w stosunku do znanych z PG, być może z wyjątkiem ich prostoty i elegancji. Jeżeli jednak zinterpretować K-wyrażenie jako gen, zalety stają się jasne. Chromosomy GEP mają stałą długość, składają się z jednego bądź więcej genów o jednakowej długości. Wynika z tego, że pojedyncze geny także mają stałą długość, a zmienia się jedynie długość obszaru kodującego drzewo – jest to esencja GEP, pozwala na zastosowanie dowolnego operatora genetycznego bez żadnych ograniczeń, dając w rezultacie zawsze poprawne wyrażenia. Nie ma więc konieczności czasochłonnego poprawiania wyrażeń (aby można było z nich skonstruować drzewo).

Gen: Pojedynczy gen w GEP składa się z głowy (*head*) i ogona (*tail*). W głowie mogą znaleźć się zarówno symbole reprezentujące funkcje, jak i symbole terminalne. Ogon natomiast zawiera wyłącznie symbole terminalne. Długość głowy h dobierana jest w zależności od oczekiwanej złożoności problemu (h definiuje pośrednio maksymalny rozmiar drzewa, które można będzie uzyskać z K-wyrażenia). Długość ogona t jest funkcją h oraz liczby argumentów funkcji z największą ich liczbą (n):

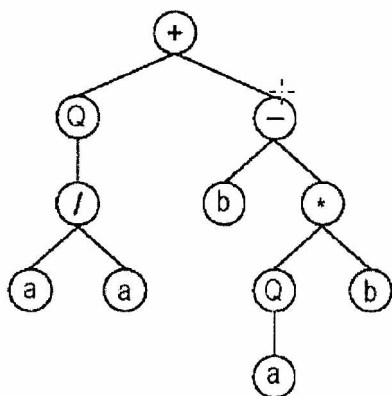
$$t = h(n-1) + 1 \quad (7)$$

Rozważmy gen składający się z alfabetu $\{Q, *, /, -, +, a, b\}$. W tym przypadku $n = 2$. Przyjmijmy $h = 10$, a więc $t = 11$. Długość genu wynosi zatem $10 + 11 = 21$. Jeden z takich genów znajduje się poniżej (chromosom 3, ogon wytłuszczono):

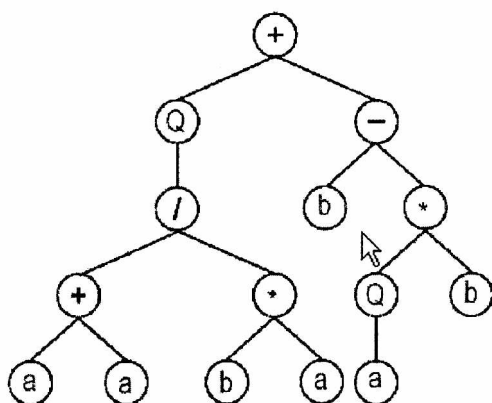
Chromosom 3: 012345678901234567890
 +Q-/b*aa**Qbaabaabbbaaab**

Koduje on następujące drzewo pokazane na rysunku 11. Widać, że do budowy drzewa wykorzystanych jest 11 symboli (0–10) K-wyrażenia, natomiast gen zawiera 21 symboli. Załóżmy, że na pozycjach 6 i 7 symbole zostały zmienione (np. przez mutację) na odpowiednio + i *. Wówczas otrzymujemy chromosom 4:

Chromosom 4: 012345678901234567890
 +Q-/b*+***Qbaabaabbbaaab**



Rys. 11. ET odpowiadające chromosomowi 3



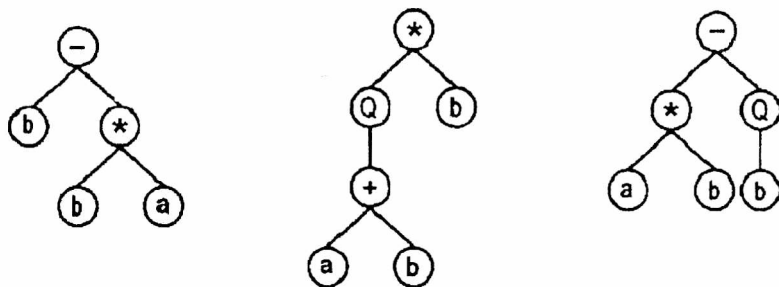
Rys. 12. ET dla chromosomu 4

Widać, że koniec obszaru kodującego drzewo przesunął się o kilka miejsc w prawo (na pozycję 14). Widać także, że otrzymaliśmy zupełnie inne drzewo, pomimo stałej długości genu (rys. 12). Dzięki odpowiedniej długości i składowi ogona (tylko symbole terminalne) mamy pewność, że dowolna zmiana któregośkolwiek z symboli K-wyrażenia nie zakłóci poprawności składniowej całego wyrażenia i możliwe będzie skonstruowanie drzewa. Pozwala to na pominięcie czasochłonnego procesu edycji i obwarowywania operatorów genetycznych warunkami.

Chromosom: Chromosomy GEP składają się zwykle z więcej niż jednego genu, wszystkie geny muszą mieć jednak taką samą długość. Liczba genów i długość głowy powinny być dobierane w zależności od rozwiązywanego problemu. Każdy gen koduje osobne poddrzewo, natomiast poddrzewa mogą formować bardziej złożone struktury. Rozważmy następujący chromosom (3-genowy) i jego drzewa:

Chromosom 5: 012345678012345678012345678
b*babbab*Qb+abbbba*Qabbaba

Chromosom 5 składa się z trzech genów, czyli są trzy obszary kodujące poszczególne poddrzewa. Pierwszy obszar kończy się na pozycji 4, drugi i trzeci na pozycji 5 (rys. 13). W zależności od zastosowania algorytmu, poddrzewa mogą być poddawane selekcji indywidualnie, w zależności od ich przystosowania (użyteczne w przypadku problemów z kilkoma wyjściami), lub też mogą tworzyć bardziej skomplikowaną strukturę, która będzie oceniana jako całość.

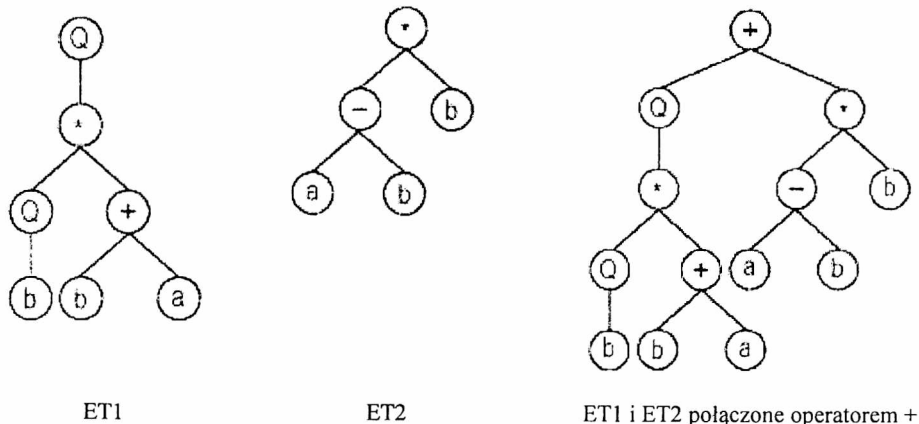


Rys. 13. ET odpowiadające chromosomowi 5

Łączenie poddrzew może być wykonywane w zadany z góry sposób, np. chromosom 6 dekoduje się na dwa poddrzewa, które mogą być połączone operatorem '+' jak na rysunku 14.

Chromosom 6: 012345678012345678
Q*Q+bbaaa*-babaabb

Reprodukcja z modyfikacjami: istnieje wiele operatorów różnicujących potomka. W odróżnieniu od PG, osobnik jest albo poddawany zmianom przez wszystkie operatory, albo przez żaden. Zmiany wprowadzane podczas reprodukcji przez kilka operatorów kumulują się, dając w rezultacie potomstwo istotnie różne od rodziców. Stosowane operatory zachowują poprawną strukturę chromosomu.



Rys. 14. ET odpowiadające poszczególnym genom i całemu chromosomowi 6

Mutacja – może nastąpić w dowolnym miejscu chromosomu z takim ograniczeniem, że symbol w głowie może być zamieniony na dowolny inny symbol, natomiast w ogonie jedynie w symbol terminalny. Gwarantuje to poprawność wyrażenia po mutacji. Mutacja poza obszarem kodowania drzewa jest dozwolona, a wręcz wskazana, i nazywa się *mutacją neutralną* (Kwaśnicka 1999). Wpływa ona na zmienność ukrytych cech osobnika.

*Transpozycja*⁴⁹ – to wybór fragmentu chromosomu i przeniesienie go na inną pozycję w danym chromosomie. Występują trzy rodzaje transpozycji:

- Wybór krótkiego fragmentu chromosomu rozpoczynającego się symbolem funkcji lub terminalnym i przeniesienie go lub skopiowanie na dowolną pozycję do głowy dowolnego genu oprócz pozycji 0 (czyli korzenia poddrzewa) – tzw. *Insertion sequence* (IS). Chromosom, początek sekwencji, jej długość i pozycja docelowa są losowo wybierane. Symbole głowy, do której wstawiana jest sekwencja są przesuwane w prawo o długość sekwencji, natomiast symbole z końca głowy są usuwane z chromosomu.

- Wybór krótkiego fragmentu rozpoczynającego się od symbolu funkcji, który może być przemieszczony na początek głowy dowolnego genu (do korzenia poddrzewa) – tzw. *Root insertion sequence* (RIS). Cała głowa przesuwa się w prawo, nadpisując przy tym ostatnie symbole głowy (w liczbie równej długości przesuwanego fragmentu).

- Przeniesienie całego, losowo wybranego genu na początek chromosomu – tzw. *transpozycja genów*.

Krzyżowanie – stosowane są klasyczne 1- i 2-punktowe krzyżowania oraz tzw. krzyżowanie genów, w którym wymieniane są pomiędzy rodzicami całe geny, zachowując pozycję.

⁴⁹ Transpozycja i inwersja to znane operatory z klasycznego algorytmu binarnego, stosowane są zwłaszcza w zadaniach porządkowych, np. w rozwiązywaniu problemu komiwojażera.

W GEP wprowadzone są tzw. efemeryczne stałe (*Ephemeral random constant*) – jest to rozwiązanie mające na celu dobór wartości stałych liczbowych występujących w chromosomie. Stosowane są specjalne tablice wartości liczbowych, z których odczytywane są wartości i podstawiane w drzewie zamiast symbolu stałej liczbowej. Początkowo są one losowane z zadanego przedziału, w trakcie ulegają losowym modyfikacjom, co pozwala dostrajać liczbowe wartości występujące w drzewach. Może to mieć zastosowanie podczas projektowania sieci neuronowych wraz z doбором ich wag.

4. HYBRYDOWE SYSTEMY INTELIGENTNE

Zanim przedstawione zostaną hybrydowe systemy inteligentne, będące połączeniami sztucznych sieci neuronowych i innych technik inteligentnych, scharakteryzujemy krótko, czym są inteligentne systemy hybrydowe.

Od kilkunastu lat obserwuje się intensywny rozwój hybrydowych systemów inteligentnych, w tym hybrydowych systemów neuronowych (Wermter, Sun 2000, Medsker 1995, Wermter, Weber 1997, Markowska-Kaczmar 2006). Znajdują one zastosowanie w takich obszarach, jak rozumienie mowy/języka naturalnego, diagnozowanie medyczne, robotyka, analiza finansowa itp.

Badania mózgu wskazują, że człowiek wykorzystuje różne struktury (architektury) w trakcie przetwarzania wiedzy. Czysto neuronowa reprezentacja jest stosunkowo atrakcyjna, zwłaszcza z punktu widzenia kognitywistyki, ale mózg, na którym wzorowane są sieci neuronowe ma zdolności przetwarzania wiedzy symbolicznej, dlatego też wydaje się celowa symboliczna interpretacja architektury sieci neuronowej. Rodzi się pytanie, w jaki sposób można połączyć neuronowe przetwarzanie sygnałów i wnioskowanie symboliczne. Wiadomo, że mózg wykorzystuje do tego różne swoje struktury, ale trudno jest powiedzieć, jakie (inne) reprezentacje wiedzy (Sun 1996).

4.1. TECHNIKI ŁĄCZENIA SYSTEMÓW INTELIGENTNYCH

Łączenie różnych technik sztucznej inteligencji pozwala na budowanie systemów o znacznie większych możliwościach. Systemy hybrydowe są tu rozumiane jako systemy łączące w sobie kilka technik sztucznej inteligencji, np. systemy ekspertowe, sieci neuronowe, algorytmy genetyczne. Takie systemy łączą w sobie zalety poszczególnych technik, eliminując ich wady (Medsker 1995). Są niezbędne, ponieważ nie da się zamodelować w taki sam sposób i za pomocą takich samych narzędzi wszystkich problemów. Część problemów lepiej jest zamodelować jako **modele symboliczne** (*symbolic models*), np. jako systemy ekspertowe, część zaś jako **modele połączeniowe** (*connectionist models*), np. jako sieci neuronowe. Systemy hybrydowe mogą okazać się interesującym sposobem do tworzenia nowych i bardziej wszechstronnych architektur, dotyczących zarówno modelowania wiedzy, jak i systemów inteligentnych.

Tworzenie systemów hybrydowych nie jest zadaniem łatwym. Nasuwa się wiele pytań, na które należy odpowiedzieć, zanim przystąpimy do budowy takiego systemu, chociażby:

- Jakie techniki sztucznej inteligencji najlepiej połączyć dla danego zadania?
- Jak powiązać rozmaite struktury znajdujące się w różnych częściach systemu, aby uzyskać optymalne rezultaty?
- Jak reprezentować wiedzę w poszczególnych modułach systemu?

Można wymienić trzy cele tworzenia hybrydowych systemów (za Medsker 1995):

1. Wzmocnienie techniki (*Technique enhancement*) – jest to integracja różnych technik w celu ominięcia ograniczeń każdej z nich.

2. Zwielokrotnienie możliwości zastosowań (*Multiplicity of application task*) – system hybrydowy tworzony jest, ponieważ pojedyncza technika nie daje się zastosować w wielu podproblemach, z którymi dana aplikacja powinna sobie radzić (np. problemy, z którymi radzi sobie dobrze system ekspertowy oraz takie, które nie są zbyt dobrze zrozumiałe i sieci neuronowe mogą sobie z nimi dobrze poradzić).

3. Realizacja wielofunkcyjności (*Realising multi-functionality*) – chodzi o stworzenie systemu hybrydowego, zdolnego do różnorodnego przetwarzania informacji: system funkcjonalnie może naśladować różne techniki przetwarzania informacji.

Systemy hybrydowe można klasyfikować według różnych kryteriów. Na warsztatach podczas 14. Międzynarodowej Konferencji Sztucznej Inteligencji (ang. *International Joint Conference on Artificial Intelligence* – IJCAI'95) zaproponowano następującą ich klasyfikację⁵⁰.

Ze względu na **architekturę systemu**, systemy hybrydowe dzielą się na:

I. *Systemy o architekturze jednomodułowej.*

II. *Systemy o architekturze wielomodułowej.*

Systemy hybrydowe o architekturze jednomodułowej można dalej dzielić ze względu na reprezentację wiedzy lub na odwzorowania pomiędzy strukturami symbolicznymi a połączeniowymi (sieciami neuronowymi). Ze względu na reprezentację dzieli się je na architektury symboliczne, lokalne i rozproszone.

Architektury symboliczne (*symbolic*) są konwencjonalnymi modelami symbolicznymi i raczej nie wchodzi w skład wielu systemów hybrydowych.

Architektury lokalne (*localist*) mają węzły przetwarzające. Jeden węzeł jest przeznaczony na reprezentację jednego problemu.

Architektury rozproszone (*distributed*) zawierają spójne, zachodzące na siebie węzły reprezentujące poszczególne problemy.

Ze względu na odwzorowanie pomiędzy strukturami symbolicznymi a połączeniowymi systemy o architekturze jednomodułowej dzieli się na architektury translacyjne i transformujące.

⁵⁰ Na podstawie (Medsker 1995): Ron Sun *Hybrid Connectionist-Symbolic Models: a report from the IJCAI'95 workshop on connectionist-symbolic integration*.

Do czynienia z **architekturami translacyjnymi** (*direct translational*) mamy wtedy, gdy utworzona struktura połączeniowa (sieć) dokładnie odpowiada strukturze zaimplementowanej w postaci symbolicznej. Przykładem może być zaimplementowanie reguł w sieci z propagacją wsteczną.

Architektury transformujące (*transformational*) polegają na tworzeniu równoważnych struktur w sieciach połączeniowych, ale bez osadzania struktury symbolicznej w sieci.

Systemy hybrydowe o architekturze wielomodułowej ze względu na budowę dzielą się na: **homogeniczne** – składają się z części, z których każda jest architekturą jednomodułową jednakowego typu (symboliczną, lokalną lub rozproszoną). W odpowiedzi na ten sam zestaw danych wejściowych, każda część może dawać różne wyjście, na wzór grupy ekspertów głosujących za prawidłową odpowiedzią, oraz **heterogeniczne**, w których poszczególne części mają różną architekturę.

Systemy **heterogeniczne** dalej można podzielić, np. ze względu na stopień powiązania modeli:

1. **Luźno powiązane** (ang. *loosely coupled*) – jeden moduł systemu komunikuje się z drugim modułem przez odpowiedni interfejs. Interfejs umożliwia współpracę modułów, która może odbywać się przez:

(a) Transformację danych wejściowych i/lub poprawianie danych wyjściowych (ang. *pre/post processing*). Może to być przejście z systemu ekspertowego na sieć neuronową lub z sieci neuronowej na system ekspertowy.

(b) Współpracę modułu głównego i modułów podrzędnych (ang. *master-slave*). Przykładowo, system ekspertowy może zamiast reguł używać sieci neuronowej (np. do podejmowania częściowych decyzji).

(c) Monitorowanie zadań – jako kombinacja dwóch poprzednich sposobów. Zadanie wykonuje się do chwili wystąpienia jakiegoś wydarzenia.

(d) Współpraca partnerska, w której moduły składają się z odpowiadających sobie procesów. Są one równorzędne pod względem funkcjonalności, ale nie reprezentacji.

2. **Ściśle powiązane** (ang. *tightly coupled*) – możliwe są różne formy współpracy pomiędzy modułami, interakcja pomiędzy modułami odbywa się poprzez:

(a) Zwielokrotnione kanały (ang. *multiple channels*).

(b) Połączenie dwóch modułów na zasadzie „węzeł-do-węzła”.

(c) Połączenie węzła w jednym module z odpowiadającym mu węzłem w drugim module.

Systemy hybrydowe o architekturze wielomodułowej (zarówno homogeniczne, jak i heterogeniczne) można podzielić ze względu na ziarnistość (ang. *grain*) na:

1. **Systemy o dużej ziarnistości** (*coarse grained*) – zawierają niewielką liczbę modułów, zwykle dwa, trzy.

2. **Systemy o małej ziarnistości** (*fine grained*) – zawierają wiele modułów.

Bardzo popularnym szkieletowym hybrydowym systemem inteligentnym jest model IRIS – skrót od *Integration of Reasoning, Informing and Serving* (Soucek 1991, Medsker 1995), wykorzystywany jest do budowania hybrydowych systemów inteligentnych.

realizujących konkretne zadania. IRIS jest modelem składającym się z połączonych ze sobą elementów programowych i sprzętowych. Takie połączenie powoduje, że tworzone technologie, produkty i usługi są znacznie bardziej wydajniejsze. Model IRIS składa się między innymi z różnych technologii (np. sieci neuronowych, systemów ekspertowych, automatycznego odkrywania reguł), standardowych modułów programowych, specjalnego języka, standardowych modułów kontroli i archiwizacji itp.

Połączenia algorytmów genetycznych z sieciami neuronowymi są systemami heterogenicznymi. Z uwagi na spełniane funkcje poszczególnych modułów można wyróżnić dwa główne podejścia (Rutkowska i in. 1997):

- **Podejście wspomagające**, w którym algorytm genetyczny i sieć neuronowa pracują sekwencyjnie po sobie.

Przykładem takiego systemu jest zastosowanie algorytmu genetycznego do ekstrakcji reguł z sieci neuronowej, gdzie najpierw uczona jest sieć neuronowa, a następnie na tej sieci pracuje algorytm pozyskujący reguły (Markowska-Kaczmar 2006) lub system, w którym algorytm genetyczny dobiera dane uczące dla sieci neuronowej.

- **Podejście współdziałające**, w którym oba algorytmy: genetyczny i sieć neuronowa działają jednocześnie.

Przykładem takiego systemu może być klasyfikator złożony (ang. *ensemble of classifiers*), w którym różne klasyfikatory – drzewo decyzyjne, sieci neuronowe, klasyfikatory genetyczne (i ewentualnie inne, np. drzewa decyzyjne) współdziałają w zadaniu klasyfikacji.

4.2. PRZESŁANKI GENETYCZNEGO PROJEKTOWANIA SIECI NEURONOWYCH

Przez długi czas algorytmy genetyczne (**AG**) i sieci neuronowe (**SSN**) były rozwijane niezależnie od siebie, lecz od pewnego czasu obserwuje się wzrastającą liczbę prac, w których obie techniki są łączone (Schafer et al 1992). **GA** mogą być stosowane do optymalizacji liczby warstw i liczby neuronów i/lub połączeń pomiędzy neuronami, do ustalenia wag połączeń w sieci zamiast stosowania algorytmu uczenia (Ossowski 1997), do ustalenia optymalnych wartości parametrów uczenia (Pearlmutter 1995) oraz do projektowania całej topologii sieci,⁵¹ czyli jej struktury wraz z wagami (Bornholdt, Graudenz 1992, Harp 1989).

Prace opisujące łączenie **GA** i **SSN** zaczęły pojawiać się w latach osiemdziesiątych ubiegłego wieku, ich rozwój nastąpił w następnej dekadzie. Obecnie, zachowanie równowagi pomiędzy technikami ewolucyjnymi a sieciami neuronowymi w syste-

⁵¹ Przez **architekturę** sztucznej sieci neuronowej będziemy rozumieć liczbę neuronów, ich organizację (np. w warstwy) oraz ustalenie połączeń między neuronami. Pojęcie „struktura sieci” jest utożsamiane z „architektura sieci”. Natomiast przez **topologię** sieci będziemy rozumieć architekturę (strukturę) sieci wraz z ustalonymi wartościami wag połączeń oraz funkcjami aktywacji neuronów. **Topologia** sieci definiuje całkowicie sieć neuronową.

mach hybrydowych jest ciągle przedmiotem badań. Wydaje się, że potrzebne są dalsze badania w celu określenia zarówno sposobu łączenia obu technik, jak i zadań, w których takie połączenie jest opłacalne.

Obecnie integracja sieci neuronowych i algorytmów genetycznych to obszar bardzo intensywnych badań (Rutkowska i in. 1997, Yao 1999). Cechą algorytmów genetycznych użyteczną w systemach hybrydowych jest ich zdolność do przeszukiwania dużych, złożonych przestrzeni potencjalnych rozwiązań. Zastosowanie algorytmów genetycznych do sieci neuronowych wiąże się z pewnymi problemami, należą do nich:

- reprezentacja w postaci chromosomu odpowiedniej informacji o sieci neuronowej (szczegóły architektury sieci i jej parametry),
- testowanie sieci neuronowej w każdym pokoleniu algorytmu genetycznego,
- ocena przystosowania sieci neuronowej,
- dostrajanie wag i innych parametrów sieci,
- wytwarzanie nowych genotypów w procesie ewolucji, stosując odpowiednie operatory genetyczne.

Algorytmy genetyczne, jak to już zostało wspomniane, mogą być wykorzystane w projektowaniu sieci neuronowych na kilka różnych sposobów, m.in.:

- wybór i przygotowanie danych uczących (Nadel, Stein 1993),
- optymalizacja wag w sieci o ustalonej strukturze (genetyczne lub ewolucyjne uczenie) – (Montana, Davis 1989),
- optymalizacja parametrów algorytmu uczącego lub reguł uczących (*Brain Maker*),
- optymalizacja struktury sieci neuronowych (Bornholdt, Graudenz 1992, Murray 1994):
 - optymalizacja liczby neuronów i/lub połączeń, i/lub liczby warstw,
 - projektowanie topologii sieci (struktury razem z wagami i/lub funkcjami aktywacji).

Najbardziej interesującym kierunkiem badań ze względów praktycznych zastosowań jest wykorzystanie algorytmów genetycznych jako narzędzia projektującego całą sieć neuronową – jej architekturę i zestaw wag.

Problem projektowania sieci sprowadza się do problemu poszukiwania architektury sieci, która będzie najlepiej wykonywała określone zadanie według określonych kryteriów działania. Proces ten może być sprowadzony do przeszukiwania przestrzeni zdefiniowanej przez powierzchnię działania wytrenowanej sieci nad przestrzenią możliwych architektur sieci neuronowych. Ponieważ liczba możliwych architektur sieci jest nieograniczona, to przestrzeń jest nieskończenie duża. Zmiany liczby jednostek lub połączeń muszą być dyskretne i mogą dawać nieciągły efekt w działaniu sieci, przestrzeń zatem jest nieróżniczkowalna. Nie ma bezpośredniego odwzorowania z projektu sieci do jej działania (po procesie uczenia), zależy ono od warunków początkowych (np. losowych wag początkowych). Sieci podobne strukturalnie mogą przejawiać zupełnie inne możliwości przetwarzania informacji, natomiast niepodobne strukturalnie mogą wykazywać bardzo podobne możliwości, proces szukania optymalnej struktury jest bardzo trudny. Potrzebna jest automatyczna metoda prze-

szukiwania tej rozległej, nieróżniczkowalnej, złożonej, zaszumionej i zwodniczej (ang. *deceptive*) przestrzeni. Metody wyliczeniowego przeszukiwania, aczkolwiek skuteczne, ze względu na rozmiar przestrzeni architektur sieci nie mogą być zastosowane. Metody przeszukiwania losowego, lepsze od wyliczeniowych tylko przy długich przebiegach, nie dają dużych szans na znalezienie użytecznego projektu. Metody gradientowe są nieużyteczne, ponieważ wymagają różniczkowalnych przestrzeni. Przeszukiwania, wykorzystujące heurystyki powstałe na podstawie wiedzy, prowadzone przez człowieka wydają się być – z powodów dyskutowanych wcześniej – nieefektywne, powolne i kosztowne. Twierdzenie schematów Hollanda (Arabas 2001, Goldberg 1989, Michalewicz 1966) wskazuje na ogólną użyteczność genetycznych przeszukiwań dużych, zwodniczych i złożonych przestrzeni. W przeciwieństwie zatem do wymienionych wyżej metod optymalizacji, algorytmy genetyczne mogą przeprowadzić szybką, skuteczną ewolucję genotypów określających użyteczne architektury sieci. Dlatego coraz częściej podejmowane są próby zastosowania algorytmów genetycznych jako techniki ewolucyjnego przeszukiwania przestrzeni architektur sieci do automatycznego projektowania sieci.

Może warto też zwrócić uwagę na argument, który wydaje się mniej uzasadniony: ludzki mózg powstał w wyniku długotrwałej ewolucji. Może warto zastosować sztuczną ewolucję do tworzenia sztucznych sieci neuronowych? Wszak sztuczny algorytm ewolucyjny wykorzystuje główny paradygmat ewolucji biologicznej: reprodukcję osobników z ich modyfikacją oraz dobór naturalny, preferujący osobniki lepiej przystosowane.

5. PRZEGLĄD SYSTEMÓW GENETYCZNO-NEURONOWYCH

W literaturze przedmiotu można znaleźć liczne przykłady łączenia tych dwóch popularnych technik inspirowanych przez naturę – algorytmów ewolucyjnych (AE) i sztucznych sieci neuronowych (SSN). Większość prac traktuje sieci neuronowe jako produkt końcowy, a algorytm ewolucyjny wykorzystuje do rozwiązywania problemów związanych bezpośrednio z samą siecią neuronową, takich jak potrzeba odpowiednio wyselekcjonowanych danych wejściowych, wyjaśnienie działania sieci, dostrojenie sieci itp. Dalej przedstawiony jest krótki przegląd takich systemów. Problematyce doboru atrybutów dla systemów uczących się poświęcony jest osobny rozdział (rozdz. 6); podobnie najważniejszej, zdaniem autorki, problematyce – projektowaniu architektury i/lub topologii sieci neuronowych (rozdz. 7).

5.1. SPOJRZENIE NA PROBLEM PROJEKTOWANIA SIECI NEURONOWYCH

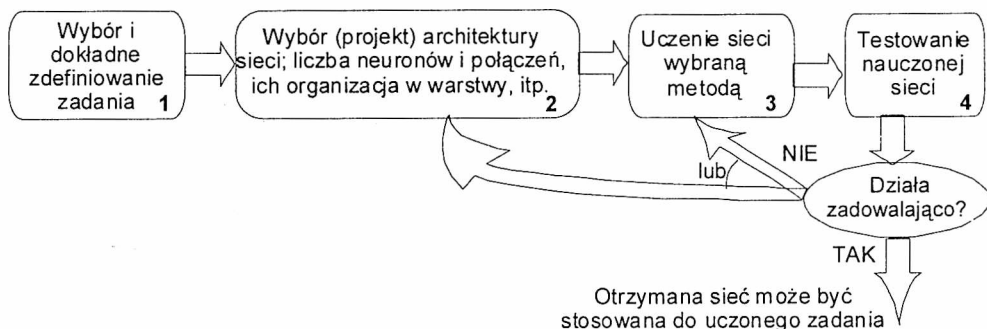
Proces rozwoju modelu sieci neuronowej dla danego zadania obejmuje zwykle cztery etapy, które mogą być powtarzane, aż osiągnięte zostaną zadowalające cele (Miller i in. 1989):

1. Wybór i dokładne zdefiniowanie problemu (np. rozpoznawanie określonych obiektów znajdujących się na obrazach, czy rozpoznawanie pewnej liczby komend wydawanych głosem przez różne osoby). Dobór danych uczących. Projektant sieci powinien kierować się tu zainteresowaniami i – przede wszystkim – doświadczeniem. Do doboru zestawu danych uczących można korzystać z pomocy eksperta w danej dziedzinie (np. lekarz pomoże wybrać zestaw cech obrazów medycznych do uczenia sieci neuronowej klasyfikacji pacjentów na podstawie obrazów zdjęć rentgenowskich).

2. Projektowanie architektury sztucznej sieci neuronowej. Biorąc pod uwagę wiedzę o wybranej dziedzinie zastosowania oraz doświadczenie i intuicję projektanta, określa on architekturę sieci. Architektura ta jest szkieletem struktury sieci: określa liczbę użytych neuronów, ich organizację w warstwy czy moduły i połączenia pomiędzy nimi.

3. Uczenie sieci wybranym algorytmem. Należy wybrać odpowiednią metodę uczenia, w zależności od tego, czy sieć ma być uczona z nauczycielem, czy bez (może to być dobór wag przez wsteczną propagację błędu, jeśli sieć jest jednokierunkowa).

4. Ocena nauczonej sieci. Należy sprawdzić, na ile wykonuje ona swoje zadanie oraz czy szybkość uczenia i zdolność uogólniania są zadowalające.



Rys. 15. Etapy projektowania sztucznych sieci neuronowych

Mimo wielu prac na temat tworzenia, uczenia i zastosowań SNN, jak stwierdzono w (Miller i in. 1989), projektowanie ich architektury nadal pozostaje bardziej sztuką niż techniką⁵². Badacz zazwyczaj polega na własnym doświadczeniu w projektowaniu sieci i nieformalnych heurystykach, które zostały zaproponowane na podstawie dotychczasowych wyników prac innych badaczy sztucznych sieci neuronowych.

Zadanie znalezienia optymalnej struktury sieci jest bardzo trudne. Wiele podstawowych problemów przetwarzania informacji w sieci (jak rozproszona reprezentacja) jest trudnych do zrozumienia nawet w niewielkiej sieci neuronowej. Zadanie wydaje się jeszcze trudniejsze, jeśli weźmiemy pod uwagę projektowanie nowych architektur sieci. Efektem tego jest to, że wielu badaczy sieci neuronowych stosuje tylko kilka standardowych typów sieci, np. jednokierunkowe warstwowe (*feedforward*) lub proste schematy rekurencyjne. Szukanie radykalnie nowych architektur to działanie „po omacku”. Standardowe techniki inżynierii projektowania systemów komputerowych nie radzą sobie z sieciami neuronowymi. Złożone, rozproszone interakcje pomiędzy jednostkami sieci zwykle powodują, że nawet metoda modularnego projektowania *dziel-i-zwycięzaj* nie jest tutaj użyteczna. Złożoność ta wydaje się uniemożliwiać opracowanie prostych, analitycznych metod projektowania. Obecnie brak jest uzasadnionych odpowiedzi na pytania typu: w jaki sposób można optymalizować projektowanie sieci neuronowych przy zadanych kryteriach działania, takich, jak szybkość uczenia, zwartość sieci, jej zdolność uogólniania i odporność na szumy? Jak można określić właściwe zmiany projektu sieci tak, aby dały one pożądane zmiany funkcjonowania sieci? Wiadomo, że mała modyfikacja w topologii sieci może dać duże różnice w funkcjonowaniu sieci i odwrotnie: sieci o istotnie różnych strukturach mogą wyka-

⁵² Pobieżne informacje na temat konstruktywnych metod budowy sieci są w rozdziale 2.2.

zywać podobne działanie. Wraz ze wzrostem liczby zastosowań sieci neuronowych i ich złożoności, takie heurystyczne podejście, naśladujące pracę projektanta–człowieka (*human-engineering*) nie jest wystarczające. Potrzebne są sposoby automatycznego, bardziej wydajnego projektowania, pozwalające obejść problemy związane z intuicyjnym projektowaniem NN przez człowieka.

Sieci neuronowe budowane są jako analogia biologicznego mózgu. Mózg powstał w wyniku wieloletniej ewolucji biologicznej. Wydaje się naturalne podejście, w którym sztucznym algorytmom genetycznym pozwala się budować sztuczne sieci neuronowe, rozwiązujące określone zadania.

5.2. GENETYCZNE UCZENIE SIECI NEURONOWYCH

Pierwsze, często cytowane podejście do uczenia sieci neuronowych algorytmem genetycznym to praca D. Montanay i L. Davisa (Montana, Davis 1989). Dobierali oni genetycznie wagi i próg funkcji aktywacji dla sieci o ustalonej strukturze. Zadaniem sieci było klasyfikowanie podwodnych sygnałów dźwiękowych „lofogramów” do dwóch klas. Sieć miała 4 wejścia, 2 warstwy ukryte zawierające 7 i 10 neuronów. Chromosom był listą 126 wag, autorzy nie stosowali kodowania binarnego, chromosom był listą liczb rzeczywistych. Autorzy zastosowali operatory mutacji i krzyżowania, dostosowane do chromosomu składającego się z zestawu liczb rzeczywistych. Populacja liczyła 50 osobników, a otrzymane rezultaty porównano z prostą metodą propagacji wstecznej i metodą *Hill Climbing*. Zaproponowane genetyczne podejście okazało się lepsze od obydwóch pozostałych metod.

V. Porto, D.B. Fogel i L.J. Fogel (Porto i in. 1995) opisują porównanie trzech metod uczenia sieci neuronowej: metodą propagacji wstecznej bez współczynnika momentu, metodą stochastyczną symulowanego wyżarzania (ang. *Simulated Annealing*) oraz metodą programowania ewolucyjnego. W metodzie ewolucyjnej chromosom był zestawem liczb rzeczywistych (wagi i bias). Podczas reprodukcji osobnika każda waga i polaryzacja (bias) były zaszumiane (mutacja) zgodnie z rozkładem Gaussa o zerowej średniej i wariancji równej średniokwadratowemu błędowi rodzica. Za funkcję przystosowania przyjęto sumę średniokwadratowych błędów dla wszystkich wzorców. Zadaniem uczonej sieci było klasyfikowanie fałszywych alarmów (do dwóch klas). Zaprezentowane przez autorów wyniki pokazują przewagę metod stochastycznych nad metodą wstecznej propagacji błędów.

W ramach zajęć dydaktycznych studenci⁵³ wykonali wiele prób genetycznego uczenia sieci neuronowych. Wiele z nich dotyczyło prostych zadań klasyfikacji, np. rozpoznawanie ręcznie pisanych cyfr lub liter, dodawanie małych liczb zapisanych

⁵³ Cytowane są tu prace studentów kierunku informatyki, specjalności inżynierii oprogramowania, którzy w ramach projektów i prac dyplomowych porównywali uczenie genetyczne i tradycyjne.

dwójkowo. W większości przypadków, poza zadaniem XOR⁵⁴, uczenie genetyczne było mniej efektywne, zarówno pod względem jakości (dokładności) działania nauczonej sieci, jak i szybkości uczenia. Po takich doświadczeniach, zaimplementowaliśmy uczenie sieci neuronowej metodą kombinowaną, genetyczne uczenie naprzemian z metodą *Instar* Grossberga. Sieci w ewoluującej populacji były uczone metodą *Instar*, wynik uczenia był traktowany jako przystosowanie sieci dla operatorów genetycznych. Taka kombinacja dała całkiem dobre rezultaty dla zadania rozpoznawania cyfr pisanych ręcznie na ekranie monitora za pomocą myszki. Zarówno czas obliczeń, jak i efekty działania sieci uczonej metodą kombinowaną oraz metodą *Instar* są porównywalne. Na rysunku 16 pokazano przykładowe cyfry podawane zaprojektowanej sieci neuronowej do rozpoznawania: pierwsza cyfra została rozpoznana jako 1 (92,93% pewności), druga jako 5 (96,25%), natomiast trzecia została rozpoznana błędnie jako 5 (91,73%). W przeprowadzonych testach, przy dość zróżnicowanej próbce uczącej, ponad 90% podawanych cyfr było rozpoznawanych poprawnie.



Rys. 16. Przykład rozpoznawania cyfr przez SSN uczoną metodą kombinowaną *Instar* i uczenie genetyczne – cyfra 3 została błędnie rozpoznana

W projektach METEO i WOCC⁵⁵ testowaliśmy genetyczne uczenie sieci neuronowych dla zadania przewidywania pogody oraz do klasyfikacji typów charakterologicznych osób na podstawie dwóch rodzajów danych: cech ich pisma odręcznego oraz cech ich wyglądu. We wszystkich trzech przypadkach uczenie za pomocą wstecznej propagacji błędów dało wyraźnie lepsze rezultaty.

W czasie realizacji wspomnianych projektów nauczyliśmy sieć neuronową również zadania XOR. Interesujące jest to, że prawie w każdym eksperymencie sieć była w stanie genetycznie nauczyć się tego zadania stosunkowo szybko, często szybciej niż podczas tradycyjnego uczenia. Dla innych zadań, wyniki genetycznego uczenia nie

⁵⁴ XOR – różnica symetryczna (inne nazwy to alternatywa rozłączna, alternatywa wykluczająca, ang. *exclusive or*) jest funkcją logiczną dwóch argumentów logicznych: wartość 1 jest wtedy, gdy oba argumenty mają różne wartości, wartość zero, gdy oba argumenty są prawdziwe lub oba są fałszywe.

⁵⁵ Kwaśnicka H., Micór A.: *Neural Networks Supported a Short-Time Weather Forecasting* oraz Kwaśnicka H., Przygoda W.: *WOCC – Hybrid Expert System for Characterological Type determining*. Obie prace opublikowane w Materiałach Konferencyjnych III Konferencji Inżynierii Wiedzy i Systemów Ekspertowych, Wrocław 1997.

były lepsze, a zwykle gorsze od uczenia klasycznymi metodami (wstecznej propagacji błędów z momentum i Instar). Mimo niezachęcających wyników uczenia genetycznego w omawianych tu projektach, przeprowadzono bardziej wnikliwe badania tego podejścia, w połączeniu z genetycznym projektowaniem architektury SSN (rozdz. 7).

5.3. OPTIMALIZACJA PARAMETRÓW ALGORYTMU UCZENIA I PARAMETRÓW SIECI

D. Chalmers podjął próbę ewolucji reguł uczących dla prostej, jednowarstwowej sieci neuronowej (Nadel, Stein 1993). Przy stosowaniu reguły delta, dla sieci z pojedynczą warstwą, zmiana wag na danym połączeniu jest proporcjonalna do aktualnej wartości wagi, wartości wejściowej, wartości wyjściowej i oczekiwanego (poprawnego) wyjścia. Chalmers przyjął, że zmiana wag powinna być określona jako liniowa zależność wszystkich wymienionych wyżej wartości. Zmiana wag następowała zgodnie z podaną zależnością:

$$\Delta w_{ij} = k_0 \left(k_1 w_{ij} + k_2 x_i + k_3 y_j + k_4 t_j + k_5 w_{ij} x_i + k_6 w_{ij} t_j + k_7 w_{ij} t_i + k_8 x_i y_j + k_9 x_i t_j + k_{10} y_j t_j \right) \quad (8)$$

gdzie: w_{ij} – waga na połączeniu od neuronu i do j , x_i – wartość wejściowa, y_j – wyjście j -tego neuronu, t_j – oczekiwana odpowiedź z j -tego neuronu, k_0 – współczynnik uczenia.

Zadaniem algorytmu genetycznego było znalezienie wszystkich wartości k_i . Chromosom składał się ze zbioru współczynników k_i , zakodowanych w łańcuchy bitów. Jako cel pracy autor przyjął sprawdzenie czy, dla prostej sieci, GA może wyprodukować regułę uczenia tak dobrą, jak reguła delta. Zbiór uczący liczył 30 odwzorowań, sieć miała jeden neuron wyjściowy i od jednego do siedmiu neuronów wejściowych. Autor uznał rezultaty za całkiem dobre, po tysiącu pokoleniach, przystosowanie najlepszej reguły wzrosło z 40 do ponad 95%. Po analizie literatury przedmiotu można stwierdzić, że nie jest to popularne podejście. Znacznie częściej wykorzystuje się podejście ewolucyjne do doboru parametrów uczenia oraz parametrów sieci (np. liczbę neuronów i/lub warstw).

BrainMaker jest komercyjnym programem do projektowania sieci neuronowych. Użytkownik sam musi określić takie parametry sieci, jak liczba warstw czy liczba neuronów. *BrainMaker* ma wbudowany moduł *GTO* – *Genetic Training Option*. Umożliwia on genetyczny wybór parametrów uczenia sieci neuronowych⁵⁶. Ewolująca populacja dostarcza różnych kombinacji parametrów wykorzystywanych w pro-

⁵⁶ Pierwsze kroki z programem *BrainMaker* for DOS (*The first steps with the BrainMaker program*). Arkus Electronics, Wrocław 1994 (napisany na podstawie *Getting Started with BrainMaker*, California Scientific Software).

cesie uczenia sieci. W systemie tym ewoluje zbiór parametrów uczących, zbiory te są oceniane w zależności od efektów uczenia sieci przy ich stosowaniu.

D. Murray (Murray 1994) stosował **AG** do optymalizacji liczby ukrytych neuronów i współczynnika uczenia dla 3-warstwowej sieci uczonej metodą wstecznej propagacji błęd. Zadaniem sieci jest przewidywanie ceny określonego artykułu. Chromosom nie jest kodowany, składa się z dwóch genów: pierwszy – wartość całkowita, jest liczbą neuronów, drugi – wartość rzeczywista, jest to współczynnik uczenia. Jako wartość przystosowawcza traktowana jest wartość zarobiona lub stracona, w dolarach, na rynku w zadanym przedziale czasu.

Problem optymalizacji struktury jednokierunkowej sieci neuronowej jest rozpatrywany w pracy (Harp i in. 1989). W omówionym tam systemie o nazwie *GENESYS* chromosom jest łańcuchem bitów, które kodują wszystkie parametry sieci: liczbę neuronów, sposób organizacji ich w warstwy, połączenia pomiędzy neuronami oraz parametry algorytmu uczenia. Te parametry są dobierane przez **AG**, natomiast sieć jest uczona metodą wstecznej propagacji błędów. Zaimplementowany w *GENESYS* algorytm genetyczny może produkować niepoprawne sieci, na przykład bez ścieżki od wejścia do wyjścia (może się tak zdarzyć, gdy sieć zawiera neurony, które nie mają połączeń na wejściu lub na wyjściu). W celu eliminacji takich sytuacji zaimplementowano dwie strategie: sieci z tzw. fatalnymi wadami są usuwane z populacji, natomiast sieci z niewielkimi wadami są „naprawiane”. Autorzy prezentują wyniki dla dwóch aplikacji (identyfikacja cyfr na podstawie ich binarnego wzorca oraz XOR), które są dość obiecujące.

NeuroForecaster, podobnie jak *BrainMaker*, jest programem do budowy sieci neuronowych dla zadań przewidywania⁵⁷. *NeuroForecaster* ma wbudowany moduł *GENETICA Net Builder Option* do ewolucyjnego tworzenia i optymalizowania architektury sieci neuronowych. *GENETICA* automatycznie steruje generacją populacji początkowej, uczeniem, testowaniem, stosowaniem operatorów genetycznych, ustala współczynnik uczenia i poziom tolerancji.

G.F. Miller i in. (Miller i in. 1989) pokazują, że **AG** jest obiecującą metodą automatyzacji projektowania sieci neuronowych. W pracy autorzy ograniczyli się do sieci jednokierunkowych o ustalonej liczbie neuronów, struktura połączeń (połączenia i ich wagi) były dobierane przez **AG**. Sieć była reprezentowana jako binarna, dwuwymiarowa tablica możliwych połączeń (od neuronu do neuronu), zakodowana w postaci łańcucha. Każde połączenie wsteczne produkowane przez **AG** było ignorowane. Do uczenia sieci wykorzystano metodę wstecznej propagacji błęd. Dla zadania XOR i kodowania wzorców **AG** był w stanie wyprodukować odpowiednie sieci.

⁵⁷ *NeuroForecaster – State of the Art Business Forecasting System, Information About the System*, Futura, Singapore, 1994.

Na Uniwersytecie im. L. Pasteura w Strasburgu, AG stosowane były do optymalizacji struktury sieci neuronowych do rozwiązywania trudnych, praktycznych zadań⁵⁸, np. rozpoznawanie obrazu satelitarne, składającego się z 36 milionów pikseli, prognozowanie giełdy przy danych pochodzących z 12 lat, obserwacje prowadzone raz dziennie, a czasami – co jedną minutę, ustawiania anteny satelitarnej itp. Z uwagi na złożoność obliczeniową, korzystano z oprogramowania pracującego na maszynach równoległych. I tak czas generowania optymalnej sieci neuronowej sięgał tygodnia lub dłużej.

Wydaje się, że najbardziej intuicyjnym i oczywistym sposobem połączenia sieci neuronowych i algorytmów ewolucyjnych jest ewolucyjne projektowanie sieci neuronowych. Jest to próba imitacji natury przez poszukiwanie kodu genotypu optymalnej architektury sieci neuronowej (określającego liczbę neuronów oraz sposób ich połączeń) (Bornholdt, Graudenz 1992, Kwaśnicka, Kilińska 1995, Kwaśnicka, Szerszon 1997, Lieske 1995, Murray 1994, Yao 1999). Uczenie genetyczne jest na tym etapie zadaniem dyskusyjnym: potrafi dać zaskakująco dobre rozwiązanie, ale przestrzeń poszukiwań jest bardzo duża i trudno jest takie dobre rozwiązanie znaleźć.

⁵⁸ *Genetic Search in NNs Optimizations*, wykład profesora J. Korczaka ze Strasburga, Francja, na konferencji *Sieci neuronowe i ich zastosowania*, Kule k/Częstochowy, 14–18.10.1997. Część oprogramowania dostępna jest pod adresem <http://dpt-info.u-strasbg.fr/lisiit/orii/gria>.

6. ALGORYTMY EWOLUCYJNE JAKO METODA SELEKCJI CECH

Selekcja cech ma szczególne znaczenie w sytuacji, kiedy dane wykorzystywane do maszynowego uczenia są wielowymiarowe. Powoduje to nie tylko wydłużenie procesu uczenia, ale też pogorszenie jego jakości – łatwo to zauważyć zwłaszcza przy zadaniu klasyfikacji. Zwykle dane są w postaci zbioru wektorów, każdy element wektora jest wartością pewnej cechy obiektu opisywanego danym wektorem. Najbardziej prawidłową i pożądaną metodą wyboru istotnych cech jest wykorzystanie do tego celu ekspertów dziedzinowych, którzy wskażą cechy istotne na podstawie swojej wiedzy i doświadczenia. Jednakże sposób ten jest bardzo uciążliwy i kosztowny, dlatego też powstają metody automatyzacji tego procesu.

6.1. WPROWADZENIE W PROBLEMATYKĘ SELEKCJI CECH

Selekcja cech polega na wybraniu minimalnych podzbiorów cech z uwzględnieniem następujących ograniczeń (Dash, Liu 1997):

- Dokładność klasyfikacji nie powinna się istotnie pogorszyć,
- Rozkład klas, biorąc pod uwagę tylko wartości wyselekcjonowanego zbioru cech, jest możliwie bliski rozkładowi z uwzględnianiem pierwotnego (całego) zestawu cech.

Algorytm selekcji cech składa się z czterech podstawowych kroków:

1. Generowanie kolejnego podzbioru kandydującego (procedura może rozpocząć pracę z pustym zbiorem cech, pełnym zbiorem cech lub podzbiorem losowo wybranych cech).
2. Ocena wygenerowanego podzbioru.
3. Sprawdzenie kryterium końca.
4. Walidacja wybranego podzbioru.

Metody generowania zbiorów kandydujących można podzielić na trzy grupy, stosujące:

- *Przegląd zupełny*: generowane są wszystkie możliwe podzbiory. Jest to technika nieefektywna, aczkolwiek zapewniająca znalezienie najlepszego podzbioru.
- *Podejście heurystyczne*: w każdej iteracji do zbioru wybranych cech dodawana jest cecha „najlepsza”, wybrana zgodnie z przyjętą heurystyką.
- *Podejście losowe*: cechy wybierane są losowo, każda cecha wybierana jest z jednakowym prawdopodobieństwem (losowanie bez powtórzeń). Wybrany zbiór cech jest testowany. Wybór zbioru powtarzany jest zadaną liczbę razy, jako wyjście zwracany jest najlepszy podzbiór.

Istotnym elementem procesu selekcji cech jest funkcja oceniająca zbiory kandydujące. Nowo wygenerowany zbiór musi być porównany z tym(i) wcześniej wygenerowanym(i). Spotyka się różne miary wykorzystywane do oceny podzbioru cech:

- *Miara odległości (Distance Measure)*, znana również pod nazwami miara rozbieżności, dyskryminacji. Dana cecha x_i jest preferowana, jeśli, na podstawie prawdopodobieństwa warunkowego, indukuje lepsze różnice między klasami niż inne cechy.
- *Miara informacji (Information Measure)* definiuje zazwyczaj przyrost informacyjny poszczególnych cech. Cecha x_i jest preferowana, jeśli jej przyrost informacji jest większy niż pozostałych (można tu wykorzystać entropię).
- *Miara zależności (Dependence Measure)* określa zdolność przewidywania wartości jednej cechy na podstawie wartości innej. Klasyczną miarą tej zależności jest korelacja, którą można wykorzystać np. do znalezienia korelacji między cechą a klasą. Preferowana jest cecha bardziej skorelowana.
- *Miara zgodności (Consistency Measure)* jest jedną z nowszych miar, preferuje spójne hipotezy⁵⁹ zbudowane z możliwie małej liczby cech.

• *Błąd klasyfikacji (Classifier Error Rate)* – metody biorące pod uwagę dokładność klasyfikacji charakteryzują się dużą dokładnością, ale są złożone obliczeniowo. Każdy kandydacki podzbiór jest wykorzystany jako dane wejściowe do systemu uczącego i jako wyjściowy zbiór zwracany jest ten, dla którego dokładność klasyfikacji jest satysfakcjonująca.

Kryterium końca jest zależne od metody generowania podzbiorów–kandydatów oraz od metody ich oceny. Może to być założona liczba iteracji, założony rozmiar podzbioru cech, czy założona dokładność klasyfikacji.

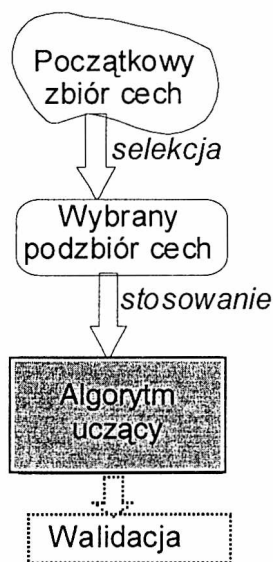
Sprawdzenie wybranego podzbioru cech (walidacja) jest zazwyczaj procesem niezależnym od samego generowania tego podzbioru. Bez tego etapu trudno byłoby z przekonaniem stosować wybrany podzbiór. Może to odbywać się poprzez stosowanie zbiorów testowych.

Przeprowadzone rozważania pokazują, że jest wiele podejść do zadania selekcji cech, ale w większości prac wyróżnia się dwa główne, istotnie różne podejścia:

⁵⁹ Hipoteza jest spójna, jeśli żadne dwa przykłady spełniające hipotezę nie należą do różnych klas.

Podjęcie filtrujące (ang. *filter approach*) (Komori i in. 2002, Raman, Ioerger 2003, Caruana, Freitag 1994, Dong, Kothari 2003, John i in. 1994, Liu i in. 1998, Dash, Liu 1997, Kovahi, John 1997).

Podjęcie to jest niezależne od stosowanej metody uczenia. Podczas selekcji i oceny otrzymanego podzbioru bierze pod uwagę zależności występujące pomiędzy cechami, np. miarę odległości probabilistycznej, miarę zależności probabilistycznej, entropię, miarę odległości międzyklasowej itp. (Raman, Ioerger 2003). Schemat ilustrujący to podjęcie pokazano na rysunku 17.

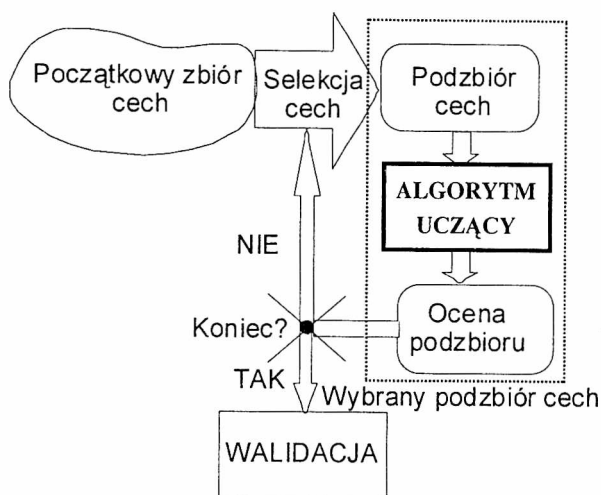


Rys. 17. Podjęcie filtrujące

Podjęcie opakowujące (ang. *wrapper approach*) (Komori i in. 2002, Caruana, Freitag 1994, Dong, Kothari 2003, Sun i in., 2002, John i in. 1994, Koller Sahami 2006, Langley 1994, Raman, Ioerger 2003, Dash, Liu 1997, Vafaie, De Jong 1992–1995, Bala i in. 1996, Pekalska i in. 2005, Kwaśnicka, Michalak 2006a, 2006b).

Podjęcie to, pokazane na rysunku 18, wymaga stosowania wybranego algorytmu uczącego już na etapie selekcji cech. Jest on potrzebny do oceny podzbioru-kandydata. Zdaniem niektórych badaczy właśnie podjęcie opakowujące staje się dominujące. Do generowania zbioru cech można stosować jedną z metaheurystyk, np. symulowane wyżarzanie, algorytmy genetyczne (Dong, Kothari 2003, Kwaśnicka, Orski 2004) bądź też wybrane zależności pomiędzy cechami (Pekalska i in. 2005, Kwaśnicka, Michalak 2006a, 2006b).

Podjęcie opakowujące pozwala wygenerować zbiór istotnych cech, ale jest złożone obliczeniowo. Dla każdego potencjalnego podzbioru wymagane jest wyuczenie klasyfikatora, z zastosowaniem walidacji skrośnej (ang. *cross-validation*).



Rys. 18. Podejście opakowujące

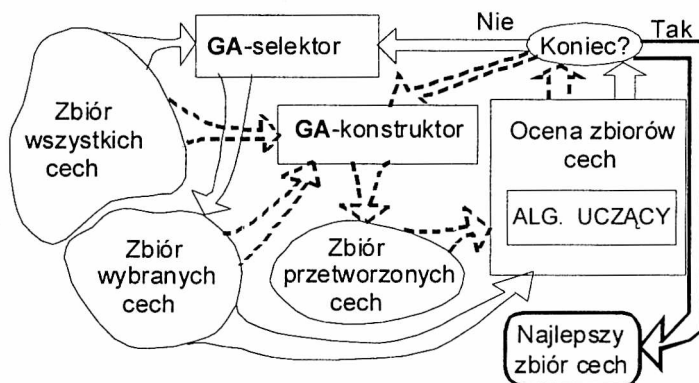
Obecnie, jak już zostało wspomniane, dość popularnym podejściem są metody opakowujące. Jednak wady, jakie one mają (duże koszty obliczeniowe dla wielowymiarowych zbiorów danych), sprawiają, że pojawiają się prace, w których duży nacisk kładzie się na efektywność metody. Efektywność ta jest ważna przy założeniu, że działanie wyuczonego klasyfikatora nie ulegnie pogorszeniu (Pekalska i in. 2005, Kwaśnicka, Michalak 2006a, 2006b).

6.2. ALGORYTM GENETYCZNY W PRZYGOTOWANIU DANYCH UCZĄCYCH

Do przygotowania odpowiedniego zestawu cech można z powodzeniem stosować algorytm genetyczny (Kwaśnicka, Orski 2004, Neri, Zini 1994, Chang, Lippmann 1991). Przygotowanie danych często obejmuje przekształcenie przestrzeni cech (danych wejściowych) oraz wybór najlepszego podzbioru przekształconych cech dla klasyfikatora sieciowego. Kelly i Davis (1991, za Neri, Zini 1994) omawiają zastosowanie **AG** do obrotu zbioru danych wejściowych (wektora wejściowego) i wyznaczenia współczynników skalowania dla każdej cechy. Pomogło to znacznie poprawić efekty stosowanego algorytmu „*k* najbliższych sąsiadów” (ang. *k nearest neighbours* – *KNN classifier*). Po wykorzystaniu **AG** dane były grupowane w ten sposób, że różnice w obrębie klasy były minimalne, natomiast różnice pomiędzy poszczególnymi klasami były znacząco duże.

Chang i Lippmann (1991, za Neri, Zini 1994) zastosowali **AG** do redukcji liczby cech dla klasyfikującego algorytmu „*k* najbliższych sąsiadów” w złożonym zadaniu rozpoznawania mowy. Za pomocą **AG** zredukowali liczbę cech ze 153 do 33. Stosowali oni również **AG** do tworzenia nowych cech na podstawie wyjściowego zbioru (przekształcenie zbioru cech).

Wśród prac stosujących algorytmy genetyczne na etapie wstępnego przetwarzania cech, uwagę zwracają prace Vafaie i De Jonga (1995), w których podejście to wykorzystywane jest do rozpoznawania zaszumionych obrazów, przy zmiennym oświetleniu. W tej pracy, obok genetycznego modułu selekcji cech, występuje też genetyczny moduł konstrukcji cech, w którym zastosowane jest programowanie genetyczne. Ogólny schemat takiego systemu pokazany jest na rysunku 19.



Rys. 19. Schemat systemu uczącego się z wstępnym doбором cech przez AG

Wyniki badań, przedstawione przez Vafaie De Jong (1995) są bardzo obiecujące. Omówiony jest tam przykład zastosowania AG do przygotowania danych dla takich algorytmów, jak ID3 czy C4.5⁶⁰. Dane wejściowe dla systemu stanowi zbiór możliwych cech. Zaimplementowano oddzielnie dwa niezależne moduły stosujące AG, jeden, nazwijmy go **GA-selektor** ma za zadanie wybrać te cechy, które są istotne dla zadania rozpoznawania, drugi to **GA-konstruktor**, jego zadanie to stworzenie nowych cech będących kombinacją cech wejściowych. Na wejście **GA-konstruktor**, można podać zarówno surowy zestaw cech (będzie działał sam, bez **GA-selektora**), jak i ten wybrany przez **GA-selektor** (gdy włączone są oba moduły). Moduły te stosują inną reprezentację chromosomów.

Moduł selekcji cech składa się z osobników – ciągów binarnych o liczbie bitów równej liczbie dostępnych cech F_i (z wyjątkiem cechy będącej numerem klasy). **GA-selektor** operuje na osobnikach będących podzbiorem możliwych cech. Znana jest licz-

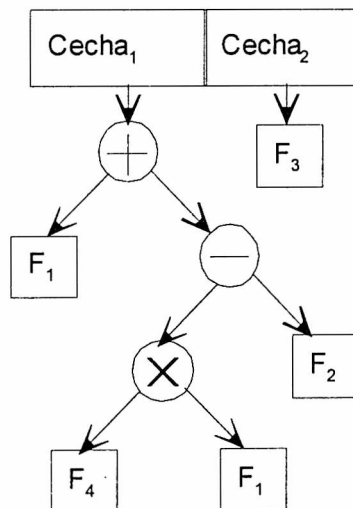
⁶⁰ Zarówno ID3, jak i C4.5 (są już następne wersje tego ostatniego, jak C5.0) należą do metod wnioskowania indukcyjnego. Stosowane są jako metoda klasyfikacji. Uczona funkcja jest reprezentowana w postaci drzewa. W węzłach drzewa są atrybuty, z każdego węzła wychodzi tyle gałęzi, ile dopuszczalnych wartości ma dany atrybut. W liściach drzewa są etykiety klas. Nauczone drzewo może być przedstawione w postaci reguł *jeśli-to* dla poprawienia jego czytelności. Drzewa decyzyjne należą do najpopularniejszych algorytmów indukcyjnego wnioskowania, mają możliwości uczenia wyrażeń dysjunkcyjnych (alternatyw). C4.5 (i jego następni) są rozwinięciem pierwszego historycznie i najprostszego algorytmu ID3. Ich rozszerzenie to możliwość działania na danych ciągłych (mają wbudowany mechanizm dyskretyzacji) i automatyczne przycinanie drzew, aby uniknąć nadmiernego dopasowania, co skutkuje brakiem zdolności uogólniania. Więcej informacji można znaleźć np. w (Cichosz 2000, Mitchell 1997).

ba wszystkich cech, efektywne zatem będzie kodowanie osobnika na tylu bitach, ile jest cech. Jedynek na danej pozycji oznacza, że dana cecha jest włączona do zestawu wybranych cech, zero – eliminację cechy. Można tu łatwo stosować typowe, proste operatory mutacji i krzyżowania. Przykład takiego osobnika dla zadania z dziewięcioma cechami pokazany jest na rysunku 20. Zero na danej pozycji oznacza, że ta cecha (F_1 , F_3 i F_8 na tym rysunku) nie jest brana pod uwagę w uczeniu klasyfikatora i podczas klasyfikacji, jedynka zaś oznacza branie danej cechy pod uwagę w uczeniu i klasyfikacji (odpowiednio cechy o numerach 2, 4, 5, 6, 7, 9).

0	1	0	1	1	1	1	0	1
---	---	---	---	---	---	---	---	---

Rys. 20. Reprezentacja osobnika w module selekcji cech

GA-konstruktor musi wykorzystywać inną reprezentację. Nowe cechy są kombinacjami cech początkowych, lub tych wyselekcjonowanych przez **GA-selektor**. Osobniki w module konstruktora są reprezentowane przez strukturę drzewiastą – jest to programowanie genetyczne (rys. 21), na którym nowe cechy są następującymi kombinacjami: $Cecha_1 = (F_1 + F_4 \cdot F_1 - F_2)$, $Cecha_2 = F_3$. Cechy mające wartości rzeczywiste można łączyć poprzez stosowanie podstawowych operatorów arytmetycznych. Chromosom ma zmienną długość. Naturalną reprezentacją dla takich zadań jest struktura drzewiasta (rys. 21). Zastosowano tu krzyżowanie jak w tzw. *programowaniu genetycznym*, zaproponowanym przez Kozę (Kozą 1992), oraz – dla zachowania koniecznej różnorodności – mutację, jako losową zmianę operatora czy cechy na inny operator lub cechę z dopuszczalnego zbioru.



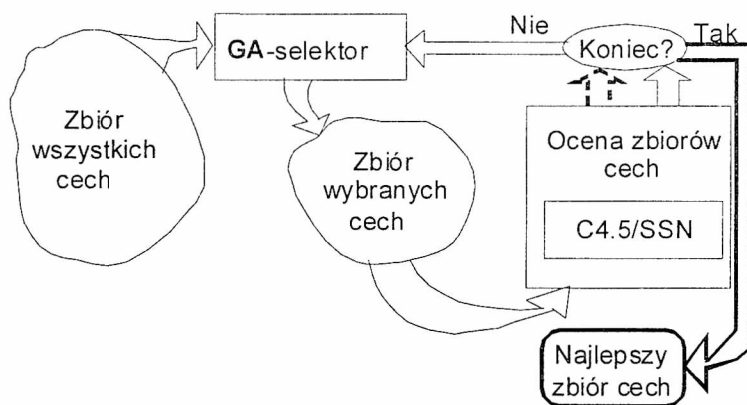
Rys. 21. Reprezentacja osobnika w module konstrukcji cech

Oba moduły wykorzystują tę samą procedurę oceny przystosowania osobników. Po wybraniu podzbioru cech następuje modyfikacja danych treningowych przez usuwanie i/lub dodanie z przykładów wartości usuniętych i/lub skonstruowanych cech. Następnie uruchamiany jest algorytm ID3 (lub C4.5) do tworzenia drzewa decyzyjnego na podstawie zmodyfikowanych przykładów uczących. Utworzone drzewo jest oceniane, biorąc pod uwagę jego zdolności klasyfikujące na danych testowych (rys. 19).

Algorytm testowano dla zadania rozpoznawania obrazów. Wymiar zbioru cech opisujących obrazy został zmniejszony o 50%, a błąd klasyfikacji dla rozpatrywanego przykładu zmniejszył się z 34,2 do 30,7%. Dodatkowy moduł, konstruktor, pozwolił na dalszą redukcję liczby cech, przy jednoczesnym niewielkim zmniejszeniu błędu klasyfikacji do 29,6%. W tego typu podejściach ważnym elementem jest funkcja oceny (funkcja przystosowania) zastosowana w algorytmie genetycznym, jej dobór jest ważnym elementem systemu. Ten sam system może być zastosowany do doboru i/lub konstrukcji cech dla sieci neuronowej.

6.3. GENETYCZNA SELEKCJA CECH DLA NN I C4.5 – STUDIUM PRZYPADKU

Przedstawimy tu badania, które miały dać odpowiedź na pytania, czy prosty algorytm genetyczny jest w stanie skutecznie zmniejszyć rozmiar przestrzeni cech nie pogarszając jakości klasyfikacji, oraz czy podzbiór cech wybrany metodą opakowującą z zastosowaniem sieci neuronowej jest taki sam jak przy stosowaniu drzew decyzyjnych generowanych z wykorzystaniem algorytmu C4.5. W tym celu zaprojektowano i zaimplementowano system komputerowy o architekturze jak na rysunku 22.



Rys. 22. Schemat systemu uczącego się z selekcją cech przez AG

6.3.1. OPIS SYSTEMU

Moduł selekcji cech jest klasycznym algorytmem genetycznym, z binarnie kodowanymi osobnikami (rozwiązaniem). Każdy bit (zero lub jedynka) koduje jedną cechę. Długość osobnika (mierzona liczbą bitów) jest równa liczbie cech w wyjściowym zbiorze danych (cecha oznaczająca klasę obiektu nie występuje w osobniku, bo nie występuje w przesłankach reguł klasyfikujących). W tym rozwiązaniu ważna jest pozycja genu: zero na danej pozycji oznacza, że cecha o danym numerze nie jest brana pod uwagę w kodowanym przez danego osobnika zbiorze, jedynka oznacza uwzględnienie danej cechy.

Użytkownik podaje zestaw wymaganych parametrów, które mają istotny wpływ na działanie algorytmu genetycznego, a przez to na działanie całego systemu. Są to typowe parametry, takie jak:

- rozmiar populacji,
- prawdopodobieństwo mutacji,
- prawdopodobieństwo krzyżowania,
- warunek zatrzymania – maksymalna liczba pokoleń, alternatywne kryterium to minimalna granica procentowego przyrostu wartości przystosowania.

Dodatkowe parametry związane są z wyznaczaniem wartości przystosowania osobników:

- kara za 1% błędu klasyfikacji,
- kara za każdą cechę aktywną.

Wykorzystana w systemie sieć neuronowa to jednokierunkowa sieć z jedną warstwą ukrytą, jedną wejściową oraz jedną warstwą wyjściową. Każdy neuron warstwy niższej ma połączenie z każdym neuronem warstwy następnej. Liczba neuronów warstwy wejściowej odpowiada liczbie aktywnych cech danego osobnika. Liczba neuronów warstwy wyjściowej jest równa liczbie klas, do których należą przykłady ze zbioru uczącego. Liczba neuronów warstwy ukrytej ustalana jest jako pierwiastek z iloczynu liczby wejść i wyjść sieci. Warto zauważyć, że dla każdego osobnika wymagana jest sieć neuronowa z inną liczbą neuronów wejściowych, dlatego należało ustalić parametrycznie liczbę neuronów w warstwie ukrytej. Jako funkcja aktywacji jest stosowany tangens hiperboliczny. Dane wejściowe są skalowane do przedziału $[-1, 1]$. Wszystkie parametry do uczenia sieci oraz sposób podziału danych na zbiór uczący i testowy są podawane jako parametry programu. Procentowy błąd klasyfikacji, wykorzystywany do ustalania przystosowania osobnika, liczony jest na zbiorze testowym.

Jako drugi klasyfikator wykorzystano algorytm C4.5⁶¹. W systemie tym zastosowano przycinanie drzewa z wykorzystaniem danych ze zbioru testującego. Minimalna liczba potomków dla każdego węzła drzewa wynosi dwa. Wymagane parametry pro-

⁶¹ Wersja YaDT version 1.1.0, udostępniona przez Salvatore Ruggieriego (*Dipartimento di Informatica, Università di Pisa*), udostępniona na stronie: <http://www.di.unipi.it/~ruggieri/YaDT> do celów badawczych (oferta była ważna w grudniu 2003 r.)

gramu dotyczą przycinania drzewa (czy przycinać, jeśli tak, to graniczny poziom ufności) oraz sposobu podziału na dane testowe i uczące.

6.3.2. PRZEPROWADZONE BADANIA

Użyteczność algorytmu genetycznego do selekcji cech została sprawdzona przy wykorzystaniu obydwóch algorytmów zaimplementowanych w module oceniającym (algorytmu C4.5 oraz sieci neuronowej). Zakres przeprowadzonych badań oraz ich rozmiar podlegał ograniczeniom związanym z akceptowalnym czasem wykonania pojedynczego badania⁶².

Z uwagi na czas obliczeń nie stosowano walidacji skrośnej. Ze względu na cel badań – możliwość zmniejszenia liczby cech w zbiorze uczącym bez pogorszenia jakości klasyfikowania, wydaje się, że rezygnacja z pełnej walidacji skrośnej jest rozwiązaniem dopuszczalnym.

Do badań wykorzystano cztery zbiory danych⁶³ z cechami o wartościach numerycznych, w których etykiety klas to wartości całkowite, zbiory danych nie zawierały brakujących wartości. Są to:

- *Arrhythmia*: 452 przykłady, 297 cech i 13 klas. Nierównomierny rozkład klas, najliczniejsza klasa – 245 przykłady, najmniej – 2 przykłady.

- *Waveform*: 5000 przykładów, 40 cech (19 z nich, to szumy), 3 klasy, równomierny rozkład klas.

- *Sonar*: 208 przykładów, 60 cech, dwie klasy, rozkład klas – 97 do 111.

- *SatImage*: 6435 przykładów, 36 cech, 6 klas. Autorzy bazy wskazują, że 4435 pierwsze przykłady to ciąg uczący, następne 2000 to testowy. Eksperymenty potwierdzają trafność takiego podziału danych.

Seria wstępnych eksperymentów pozwoliła ustalić parametry systemu, przy których działa on w sposób akceptowalny. Jako domyślne parametry ustalono: prawdopodobieństwo mutacji równe 0,001 dla długości osobników większej niż 100, 0,01 dla mniejszych, prawdopodobieństwo krzyżowania 0,7, 50 osobników, 130 pokoleń. Ocena osobnika polegała na ustaleniu maksymalnej wartości przystosowania osobnika w danej populacji (dla danego zadania), po czym, dla każdego osobnika, od tej maksymalnej wartości odejmowana jest wartość kary, wynikająca z liczby aktywnych cech oraz procentowej wartości błędu klasyfikacji, dawanego przez wyuczony na tym zbiorze klasyfikator.

W badaniach uwzględniano dwa warianty kar:

A. kara za cechę aktywną: 3, kara za 1% błędu: 2 (**podejście A**, zorientowane na zmniejszanie liczby cech),

⁶² Eksperymenty przeprowadzono na komputerze z procesorem Pentium III 733, 512 MB RAM, system operacyjny WINDOWS XP.

⁶³ Do badań wykorzystano popularne, benchmarkowe bazy danych, udostępnione na stronie <http://www.ics.uci.edu/~mllearn/MLSummary.html>.

B. kara za cechę aktywną: 2, kara za 1% błędu: 3 (**podejście B**, zorientowane na zmniejszanie błędu klasyfikacji).

Pierwsze rozwiązanie (A) wywierało większą presję na liczbę aktywnych cech, natomiast drugie (B) – na dokładność klasyfikacji. Dla obu tych podejść przeprowadzono serie badań.

Wykonano serię wstępnych badań w celu ustalenia podstawowych parametrów systemu oraz wpływu na uzyskiwane wyniki rozmiaru zbioru trenującego i przycinania drzewa (dla C4.5). Następnie przeprowadzono właściwe badania, w których interesujące były dwie wielkości: stopień redukcji liczby cech i wpływ tej redukcji na dokładność klasyfikacji, przy dwóch danych wariantach karania.

6.3.3. ZESTAWIENIE I PORÓWNANIE WYNIKÓW

Wszystkie eksperymenty były powtarzane wielokrotnie dla tych samych wartości parametrów systemu (od pięciu do dziesięciu razy). Czas ewolucji ustalono na 100 pokoleń, jako że najlepsze osobniki zwykle pojawiały się w populacji około pięćdziesiątego pokolenia.

Dla wszystkich testowanych zbiorów danych system był w stanie zredukować liczbę cech przy obu podejściach (A i B). Otrzymane wyniki wskazują, że klasyfikator C4.5 jest nieco lepszy. Krótkie podsumowanie wyników zawiera tabela 1.

Na podstawie otrzymanych wyników można jednoznacznie stwierdzić, że obie metody indukcyjne miały problemy z bazą Arrhythmia. Zaproponowany system selekcji cech nie był w stanie, dla bazy Arrhythmia, uzyskać znacznej redukcji liczby cech z jednoczesnym zachowaniem dokładności klasyfikacji (a tym bardziej, jej poprawą). Jedynym sukcesem była sama redukcja liczby atrybutów. Najlepiej oceniony osobnik wykorzystywał jedynie 97 cech (spośród dostępnych 297). Niepowodzenie obu metod indukcyjnych może być spowodowane rozmiarem bazy Arrhythmia (liczbą jej cech). Dla dużej liczby klas oraz wielowymiarowego wektora danych uczących, należało w każdym kroku trenować dość złożoną sieć neuronową. Z uwagi na długi czas eksperymentów zbyt mocno ograniczono maksymalną liczbę epok. Do wytrenowania tak złożonej sieci neuronowej mogło to być niewystarczające.

Dużo lepsze wyniki udało się uzyskać dla bazy Waveform. Przy wykorzystaniu mechanizmu drzewa decyzyjnego nastąpiła redukcja liczby cech aż o 75%, przy jednoczesnej nieznacznej poprawie zdolności klasyfikacji. Kiedy w miejscu algorytmu uczącego użyto sieci neuronowej, również nastąpiła znaczna redukcja liczby używanych cech (z 40 do 16). Także w tym przypadku zaobserwowano nieznaczną poprawę dokładności klasyfikacji.

Warto wspomnieć, że sieci neuronowe zazwyczaj z większą precyzją są w stanie klasyfikować nieznane przykłady (mają lepszą zdolność uogólniania), co znalazło odzwierciedlenie również w wynikach badań przeprowadzonych przy wykorzystaniu bazy Waveform. Dla sieci neuronowych uzyskiwano błąd klasyfikacji mniejszy nawet o dziesięć punktów procentowych od wyników, jakie zwracał algorytm C4.5. Na podstawie przeprowadzonych dodatkowych badań można jednoznacznie stwierdzić,

Tabela 1. Zadanie klasyfikacji – redukcja liczby cech przez algorytm genetyczny

Zbiór danych, rozmiar i sposób tworzenia ciągu uczącego	Eksperyment	Liczba cech	Redukcja [%]	Błąd klasyfikacji [%]	Polepszenie [punkty proc.]
Arhythmia; 20%; od początku	cały zbiór, C4.5	297	–	32,87	–
	podejście A, C4.5	97	67,34	35,63	–2,76
	podejście B, C4.5	118	60,27	33,15	–0,28
	cały zbiór, NN	297	–	38,95	–
	podejście A, NN	126	57,24	45,02	–6,07
	podejście B, NN	124	58,25	41,99	–3,04
WaveForm; 40%; od początku	cały zbiór, C4.5	40	–	24,97	–
	podejście A, C4.5	10	75	24,27	0,70
	podejście B, C4.5	16	60	24,63	0,34
	cały zbiór, NN	40	–	18,19	–
	podejście A, NN	14	65	18,39	0,20
	podejście B, NN	16	60	15,86	2,33
Sonar; 90%; od początku	cały zbiór, C4.5	60	–	19,05	–
	podejście A, C4.5	21	65	0	19,05
	podejście B, C4.5	24	60	0	19,05
	cały zbiór, NN	60	–	0	–
	podejście A, NN	20	66,67	0	0
	podejście B, NN	21	65	0	0
Sonar; 70%; losowo	cały zbiór, C4.5	60	–	26,98	–
	podejście A, C4.5	19	68,33	17,46	9,52
	podejście B, C4.5	27	55	9,52	15,46
	cały zbiór, NN	60	–	17,46	–
	podejście A, NN	26	56,67	3,17	14,29
	podejście B, NN	28	53,33	4,76	12,70
SatImage; 40%; od początku	cały zbiór, C4.5	36	–	30,79	–
	podejście A, C4.5	10	72,22	30,30	0,49
	podejście B, C4.5	12	66,67	22,66	8,13
SatImage; 40%; losowo	cały zbiór, C4.5	36	–	15,68	–
	podejście A, C4.5	11	69,44	14,53	1,15
	podejście B, C4.5	12	66,67	13,81	1,87
	cały zbiór, NN	36	–	22,01	–
	podejście A, NN	13	63,89	17,92	4,09
	podejście B, NN	14	61,11	18,20	3,81
SatImage; 90%; od początku	cały zbiór, C4.5	36	–	14,66	–
	podejście A, C4.5	9	75	14,13	0,53
	podejście B, C4.5	12	66,67	12,58	2,08
	cały zbiór, NN	36	–	20,65	–
	podejście A, NN	13	63,89	15,68	4,97
	podejście B, NN	14	61,11	12,27	8,38

Opis danych uczących w kolumnie 1 jest następujący: nazwa zbioru; procent wzorców, na których uczony był system; sposób wyboru wzorców uczących: „od początku” oznacza, że wybór wzorców był według ich kolejności w bazie; „losowo” – wzorce do uczenia były wybierane z całego zbioru danych losowo, z rozkładem równomiernym.

że obie metody trafnie typowały podzbiory cech. Sprawdzenie tego faktu polegało na budowie mechanizmu decyzyjnego, np.: drzewa decyzyjnego z wykorzystaniem podzbioru cech, wyselekcjonowanego przez drugą metodę indukcyjną – sieć neuronową. Jeśli dla danego podzbioru cech (oraz odpowiedniego zbioru trenującego) obie metody uzyskiwały błąd klasyfikacji na tym samym poziomie, to świadczyło to jednoznacznie, że wybrane zbiory danych są trafnie wyselekcjonowane.

Równie dobre rezultaty dały badania przeprowadzone dla bazy Sonar. Dla obu metod indukcyjnych udało się uzyskać redukcję liczby używanych cech aż o 65%. Zmniejszenie rozmiarów wektora z danymi nie wpłynęło jednak, w negatywny sposób, na zdolności klasyfikacyjne metod uczących się. Wprost przeciwnie – uzyskano obniżenie błędu klasyfikacji nawet o 19 punktów procentowych, aż do poziomu 0% (wszystkie decyzje są poprawne). Również w tym przypadku zostały przeprowadzone badania sprawdzające poprawność selekcji podzbiorów cech. Okazało się, że dane wyselekcjonowane z użyciem metody C4.5 implikują niemal identyczny błąd klasyfikacji dla algorytmu C4.5 oraz dla sieci neuronowej. Niestety zależność ta nie działała w drugą stronę – dane wyselekcjonowane przez sieć neuronową powodowały duże różnice błędu klasyfikacji (nawet o 30 punktów procentowych). Może to świadczyć o tym, że w tym przypadku system z włączonym algorytmem C4.5 dokonuje trafniejszych wyborów.

Ostatnią bazą danych wykorzystaną w trakcie badań była SatImage. Dla obu metod, zarówno dla drzewa decyzyjnego, jak i dla sieci neuronowej, uzyskano bardzo dobre wyniki. W obu przypadkach doszło do zredukowania liczby cech o ponad 60% (dla drzewa decyzyjnego nawet o 75%) oraz zmniejszenia błędu klasyfikacji o około osiem punktów procentowych (błąd klasyfikacji wynosił 12% po eliminacji niektórych cech). Dla wyselekcjonowanych podzbiorów cech uzyskiwano bardzo zbliżone wartości błędu klasyfikacji, niezależnie od tego, jaką metodę indukcyjną zastosowano. Świadczy to jednoznacznie o poprawności wyboru podzbiorów cech (obie metody reagowały podobnie na te same zestawy cech).

Oprócz omówionych badań, przeprowadzono, ale już na mniejszą skalę, doświadczenia, polegające na uzupełnieniu baz danych nadmiarowymi cechami oraz na wstawieniu jako dodatkowej kolumny, kolumny determinującej etykietę klasy. Zastosowano dwie techniki doboru nieistotnych, nadmiarowych danych:

- losowo,
- same zera.

Dla danych o wartościach losowych często dochodziło do sytuacji, w której nie podlegały one eliminacji, a wprost przeciwnie – znajdowały się w puli wyselekcjonowanych cech (metoda selekcji cech potrafiła znaleźć zależności między nowo wstawionymi cechami a klasą przykładu).

W momencie, kiedy do bazy danych były wstawiane same wartości zerowe, zazwyczaj następowała eliminacja takiej cechy.

Kiedy do bazy danych była wstawiana kolumna determinująca etykietę klasy (jako cechę), dochodziło do sytuacji, w której błąd klasyfikacji dla większości osobników

wynosił zero, a metoda ewolucyjna starała się jedynie wyeliminować nadmiarowe cechy. Zazwyczaj po 100–150 pokoleniach dochodziło do sytuacji, gdzie liczba cech była o połowę mniejsza niż uzyskiwana dla standardowych, niezmodyfikowanych baz danych. Wydłużenie procesu ewolucji, ewentualnie wraz z zwiększeniem karania za każdą aktywną cechę, prawdopodobnie doprowadziłby do wyeliminowania wszystkich zbędnych cech⁶⁴.

Zaprezentowane badania jednoznacznie wskazują na skuteczność zastosowania algorytmów genetycznych jako metody selekcji cech. W trakcie prowadzenia badań dość często zdarzało się, że w pierwszym pokoleniu (generowanym losowo) pojawiały się bardzo dobre osobniki o dużej wartości funkcji przystosowania (występowała u nich redukcja cech rzędu 50% i więcej).

W początkowych etapach ewolucji średnie przystosowanie całej populacji bardzo szybko rosło, jednak często po osiągnięciu 20–30 pokoleń dochodziło do zwolnienia lub zahamowania dalszego wzrostu przystosowania. Przyczyny takiej sytuacji można upatrywać w małej liczności populacji (jedynie 50 osobników), co skutecznie ograniczało możliwość efektywnej ewolucji. Po pewnym czasie dochodziło do sytuacji, gdzie dalszy wzrost przystosowania przebiegał bardzo wolno (następowała zbieżność populacji do jednego, niekoniecznie globalnego optimum). Duża złożoność obliczeniowa badań nie pozwoliła jednak na zwiększenie liczności populacji oraz na wydłużenie procesu ewolucji (np.: do 1000 pokoleń).

Analiza uzyskanych wyników w tych wstępnych badaniach wskazuje, że zadowalające podzbiory cech, uzyskiwane przy stosowaniu sieci neuronowych jako funkcji oceniającej, nie były takie same jak przy stosowaniu klasyfikatora C4.5. Jest to niewątpliwie wadą metody – wyniki nie są przenoszalne między różnymi klasyfikatorami.

Warto zweryfikować zauważone w przedstawionych badaniach wnioski, stosując inne metody selekcji cech (np. Kwaśnicka, Michałak 2006a, b). Będzie to przedmiotem kolejnych badań.

⁶⁴ Podobne badania, mające na celu wyeliminowanie zbędnych danych w zadaniu modelowania danych za pomocą algorytmów ewolucyjnych, potwierdziły zdolność selekcji istotnych danych, patrz Kwaśnicka H., Szpunar-Huk E.: *Genetic Programming in Data Modelling*, rozdział w: *Genetic Systems Programming. Theory and Experiences*, Springer, Series: Studies in Computational Intelligence, Vol. 13 Nedjah N., Abraham A. Mourelle L. de Macedo (ed.), s. 105–130, 2006.

7. GANN⁶⁵ – METODA GENETYCZNEGO PROJEKTOWANIA SSN

Próby wykorzystania algorytmu genetycznego do projektowania topologii sieci neuronowych można znaleźć w literaturze, np. (Bornholdt, Graudenz 1992, Dasgupta, McGregor 1992, Kwaśnicka, Kilińska 1995). W niniejszym rozdziale przedstawiona jest autorska metoda i badania dotyczące wad i zalet bezpośredniego i pośredniego sposobu kodowania sieci neuronowych w osobniku oraz różnych funkcji przystosowania, preferujących jakość działania nauczonych sieci oraz ich rozmiar. Omówione są stosowane podejścia oraz uzyskane wyniki wraz z ich dyskusją. W badaniach ograniczono się do automatycznego generowania jednokierunkowych sieci neuronowych, oraz do rekurencyjnych sieci neuronowych z dyskretnym czasem, realizujących zadania generowania sekwencji.

7.1. OGÓLNY SCHEMAT GENETYCZNEGO PROJEKTOWANIA SIECI NEURONOWYCH

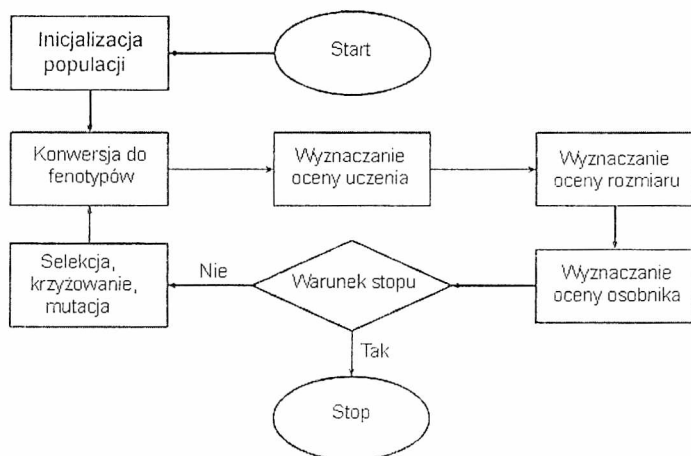
Przystępując do genetycznego projektowania sieci neuronowych rozwiązujących dane zadanie, należy podjąć ważne decyzje, od których w dużym stopniu zależy skuteczność i efektywność naszego podejścia. W celu zaprojektowania systemu do automatycznego projektowania sieci neuronowych należy wykonać następujące czynności:

1. Określenie zadania dla algorytmu ewolucyjnego – szukanie topologii sieci, tylko struktury lub też struktury sieci i wartości parametrów uczenia
2. Określenie sposobu kodowania osobnika (reprezentacji sieci neuronowej)
3. Zdefiniowanie funkcji oceniającej (funkcji przystosowania)
4. Zdefiniowanie operatorów genetycznych
5. Dobór wartości parametrów algorytmu ewolucyjnego.

Ogólny schemat takiego systemu przedstawiony jest na rysunku 23. Jak widać, czynności wymienione w punktach 1–4 należy wykonać zanim rozpoczniemy budowę systemu komputerowego. Punkt piąty – dobór parametrów algorytmu ewolucyjnego –

⁶⁵ *Genetic Algorithm for Neural Networks.*

jest wykonywany w zasadzie metodą prób i błędów w czasie testowania działania systemu dla rozwiązywanego problemu. Schemat na rysunku 23 uwzględnia w ocenie projektowanej sieci zarówno jakość uczenia się, jak i rozmiar sieci.



Rys. 23. Projektowanie sieci neuronowej za pomocą algorytmu ewolucyjnego – schemat ogólny

Z matematycznego punktu widzenia architektura sieci neuronowej jest *grafem skierowanym* $G = [V, E]$. Wierzchołki V grafu G są neuronami, krawędzie E grafu G są połączeniami między neuronami. Waga przypisana krawędzi odpowiada wadze danego połączenia w sieci. Rolę neuronów wejściowych oraz wyjściowych pełnią wierzchołki grafu. Zakłada się istnienie jednego dodatkowego wierzchołka, który nie ma wejść, a na wyjściu ma zawsze sygnał równy 1. Wierzchołek ten pełni rolę biasu. W stosowanych w tej pracy sieciach neuronowych wyróżnione są następujące elementy, które mogą podlegać ewolucji:

1. Liczba neuronów.
2. Istnienie połączeń pomiędzy neuronami.
3. Wagi na połączeniach pomiędzy neuronami.

Pozostałe elementy, takie jak np. funkcja przejścia neuronu, są stałe w czasie ewolucji dla całej sieci neuronowej. Dwa pierwsze spośród wymienionych elementów są reprezentowane w algorytmie genetycznym w postaci bezpośredniej (jako macierz) lub pośredniej (drzewo binarne). Element trzeci jest reprezentowany w algorytmie uczącym w postaci macierzy i nie jest poddawany procesowi ewolucji w głównym algorytmie genetycznym. Wartość na wyjściu pojedynczego neuronu w stosowanej sieci bez sprzężenia zwrotnego jest wyliczana jako tangens hiperboliczny sumy ważonych wejść i biasu.

Sieci bez sprzężenia zwrotnego są *acyklicznymi grafami skierowanymi*. Bardzo często przypisuje się węzłom unikatowe identyfikatory, będące liczbami całkowitymi. Warunkiem koniecznym i wystarczającym istnienia acyklicznego grafu skierowanego G jest istnienie przynajmniej jednej permutacji przypisanych wierzchołkom V grafu

G identyfikatorów takiej, w której żaden wierzchołek o wyższym identyfikatorze nie jest połączony krawędzią z wierzchołkiem o niższym lub równym identyfikatorze.

7.2. KODOWANIE SIECI NEURONOWYCH I STOSOWANE OPERATORY

Wśród różnych metod kodowania struktury sieci neuronowych wyróżnia się dwa główne nurty: kodowanie bezpośrednie (np. Whitley 1995) i pośrednie (np. Whitley 1995, Gruau 1994_w). W kodowaniu bezpośrednim sieć neuronowa jest wprost odwzorowana w chromosomie osobnika. Natomiast w kodowaniu pośrednim konieczna jest procedura dekodowania chromosomu w postać sieci neuronowej. Oba podejścia mają swoje wady i zalety, co będzie widoczne w przedstawionych w dalszej części pracy eksperymentach.

Najbardziej typowa reprezentacja struktury projektowanej sieci neuronowej to zakodowanie każdego połączenia w genotypie. Połączenia określone są przez macierz E , w której element e_{ij} jest zerem, jeśli nie ma połączenia między neuronem i a j , 1 w przeciwnym przypadku. Ma ona wymiar $n \times n$, gdzie n jest liczbą neuronów w sieci (jest to założona, stała liczba). Następnie macierz ta odczytywana jest wierszami i w ten sposób tworzony jest łańcuch bitów będący chromosomem.

Mniej bezpośrednim kodowaniem topologii w genie jest kodowanie jedynie kilku parametrów topologii sieci, które opisują sieci z ograniczonego zbioru architektur. Parametrami tymi mogą być: liczba neuronów w każdej warstwie ukrytej, liczba warstw ukrytych itp. Często w chromosomie są również uwzględniane parametry algorytmu propagacji wstecznej, który jest stosowany jako algorytm uczenia. Algorytm genetyczny (AG) może generować całą topologię sieci (wraz z wagami połączeń i progami), wtedy AG wykorzystywany jest również jako metoda uczenia sieci (Bornholdt, Graudenz 1992, Kwaśnicka, Kilińska 1995). W takich systemach pojawia się często istotny problem: system może wytwarzać sieć, która nie spełnia wymagań narzuconych na topologię. Usuwanie takich sieci z populacji bądź też ich „naprawianie” jest czasochłonne. Jako wartość przystosowania (kryterium oceny jakości sieci) najczęściej brany jest pod uwagę błąd średniokwadratowy, określany na wyjściu sieci na podstawie zbioru trenującego. Jeśli ważna jest szybkość uczenia sieci, wówczas dodawany jest np. czynnik ograniczający liczbę epok. Bardziej złożone ważone funkcje przystosowania mogą zawierać dalsze kryteria, takie jak uogólnianie czy liczbę połączeń w sieci. Bornholdt i Graudenz (Bornholdt, Graudenz 1992) w funkcji przystosowawczej uwzględniali wprost średnią liczbę cykli potrzebną do osiągnięcia przez sieć stanu stabilnego (ewoluowana sieć nie była siecią jednokierunkową).

Otrzymywane w ten sposób sieci testowano na stosunkowo prostych przykładach, np.: funkcja XOR, kopiowanie binarnych wzorców z wejścia na wyjście (Schafer i in. 1992). Rozmiar populacji wynosił 50, współczynnik krzyżowania 0,6, a współczynnik mutacji 0,005. Do reprodukcji były zatrzymywane najlepsze osobniki z poprzedniej populacji. Ponadto parametry sieci zmieniały się w zależności od wykonywanego za-

dania. Dla prostych zadań AG był w stanie dość szybko wygenerować odpowiednie architektury sieci.

Oprócz przykładowego XOR, w (Schafer et al 1992) autorzy wymieniają wiele innych, rzeczywistych zastosowań, między innymi: zbudowanie stosunkowo dużej sieci neuronowej klasyfikującej pasywne sygnały sonarowe. Trenowana sieć składała się z około 500 połączeń. Każda waga była kodowana w postaci liczby rzeczywistej i rekombinacja pojawiała się jedynie pomiędzy wagami. Wykorzystywano małą populację (ok. 50 osobników). Współczynnik mutacji był wyższy niż w wielu tradycyjnych zastosowaniach AG. Podobne podejście zastosowano podczas budowy sieci rozwiązującej zadania nawigacyjne obiektu podwodnego. Jako kolejny tego typu przykład można wymienić zastosowanie AG do określenia topologii sieci dodającej dwubitowe liczby oraz sieci do problemów detekcji sygnałów impulsowych (sieć miała 500 połączeń).

Ciekawe podejście do projektowania sieci neuronowych za pomocą AG przedstawione jest w (Dasgupta, McGregor 1992): Dasgupta i McGregor zastosowali hierarchiczny AG. W podejściu tym chromosom składał się z dwóch segmentów łańcuchów bitów: poziom wyższy reprezentował połączenia pomiędzy N neuronami, poziom niższy – wagi połączeń w sieci. Funkcja przystosowawcza odzwierciedlała sumaryczny błąd średniokwadratowy na wyjściu sieci, „poprawność” jej struktury oraz jej złożoność. W systemie tym poprawne struktury były „nagradzane”: część z wyższego poziomu pozostawała bez zmian (struktura sieci) przez pewną liczbę pokoleń, natomiast algorytm genetyczny działał na części z niższego poziomu (dobór wartości wag). Przeprowadzone eksperymenty dla problemów XOR oraz kodowania 4×4 , przy logistycznej funkcji przejścia i sigmoidalnej na neuronach wyjściowych, z dwupunktowym operatorem *crossover*, dały dobre rezultaty. Dla problemu XOR dobrą sieć uzyskiwano po około 200 pokoleniach, dla kodowania – po około 350. Jednak dla dużych sieci algorytm ten nie był skuteczny, autorzy konkludują, że należy szukać innej reprezentacji sieci.

7.2.1. KODOWANIE BEZPOŚREDNIE

Kodowanie bezpośrednie jest najbardziej intuicyjne. Sieć neuronowa jest pamiętana jako graf skierowany, geny osobnika są bezpośrednio odzwierciedlane na węzły (neurony) sieci i połączenia między nimi. Ten sposób kodowania umożliwia reprezentację dużego zbioru różnych sieci neuronowych, co powoduje, że w podejściu tym kładzie się nacisk na możliwość generowania sieci neuronowych najbardziej odpowiednich dla danego zadania. Z jednej strony daje to szansę na znalezienie najlepszej sieci neuronowej, z drugiej zaś – zwiększa przestrzeń przeszukiwań, co powoduje większe zapotrzebowanie na czas obliczeń i pamięć komputera.

Osobnik: Najprostsze rozwiązanie to zakodowanie sieci neuronowej o N neuronach jako sieć połączeń. W podejściu tym wagi nie są kodowane, sieć jest uczona tradycyjną metodą. Macierz połączeń jest zapisywana wierszami w ciąg zer i jedynek, stanowiący

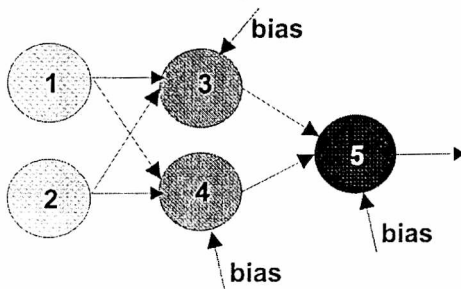
chromosom. Podejście takie zastosowali Miller i Todd (Miller et all 1989). Ten sposób kodowania, oprócz niewątpliwej zalety – prostoty i czytelności, ma kilka ważnych wad, np. dla dużych sieci neuronowych chromosom osobnika jest bardzo duży, dla sieci jednokierunkowych połowa macierzy (poniżej przerywanej linii na rys. 24) jest niewykorzystywana, koduje tylko sieci o stałej liczbie neuronów.

od do	1	2	3	4	5	bias	
1	0	0	0	0	0	0	→ 00000
2	0	0	0	0	0	0	→ 00000
3	1	1	0	0	0	1	→ 110001
4	1	1	0	0	0	1	→ 110001
5	0	0	1	1	0	1	→ 001101

Rys. 24. Kodowanie bezpośrednie sieci neuronowej: macierz połączeń

od do	1	2	3	4	5	bias	
1	0	0	0	0	0	0	→ 00000
2		0	0	0	0	0	→ 0000
3			0	0	1		→ 001
4				0	1		→ 01
5					1		→ 1

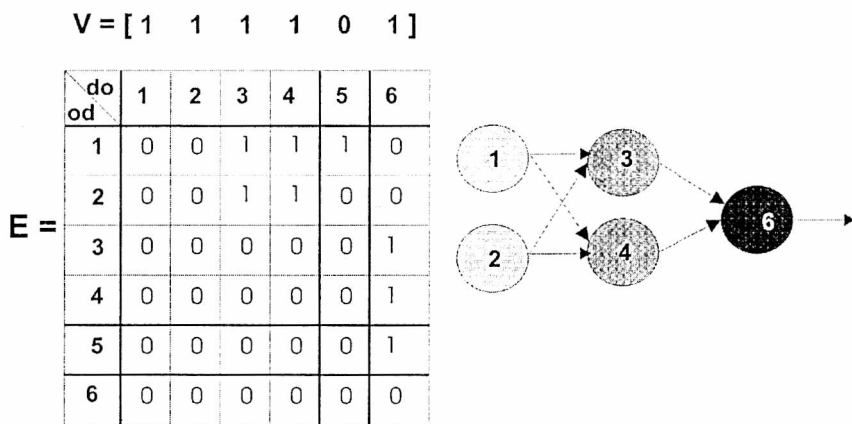
chromosom: 00000 0000 001 01 1 + wagi (kodowane binarnie lub bezpośrednio - liczby rzeczywiste)



Rys. 25. Przykład kodowania bezpośredniego sieci neuronowej: macierz połączeń, chromosom i odkodowana sieć neuronowa

Do powyższego sposobu kodowania możemy dodać do macierzy połączeń wartości wag (np. kodowanie binarnie), dla sieci jednokierunkowych można uwzględnić tylko połowę macierzy, co spowoduje skrócenie długości chromosomu. Takie podejście zastosowane jest m.in. w pracy (Dasgupta, McGregor 1992), rysunek 24. Sama macierz i chromosom oraz odkodowana sieć dla sieci z tego rysunku wyglądałyby jak na rysunku 25.

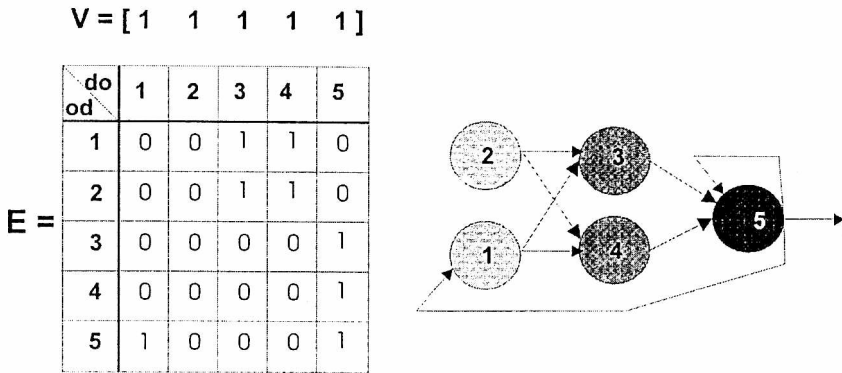
Jak już wspomniano, kodowanie sieci neuronowej za pomocą macierzy połączeń jest bardzo intuicyjne, dlatego też jest dość często spotykane w literaturze. Stosunkowo łatwo można sobie poradzić z różną liczbą neuronów w sieci, wymaga to założenia maksymalnej liczby neuronów. Sieć neuronowa zawierająca maksymalnie n neuronów jest reprezentowana przez macierz E i wektor V . Macierz E koduje informację, czy pomiędzy daną parą neuronów jest połączenie. Wektor V koduje informację, czy dany neuron jest aktywny. Stosując tylko macierz połączeń E moglibyśmy ewoluować sieci o stałej, z góry zadanej liczbie neuronów. Zastosowanie dodatkowego wektora V pozwala ewoluować sieci mniejsze o różnej liczbie neuronów, nie większej od założonego parametru n .



Rys. 26. Przykład kodowania bezpośredniego sieci neuronowej i odkodowana sieć neuronowa

Dla sieci jednokierunkowych wykorzystywana jest tylko połowa macierzy E . Wektor V stanowi maskę na macierz E , informującą, które neurony brać pod uwagę w trakcie dekodowania sieci. Zarówno E jak i V kodują warstwy wejściową, ukrytą (lub ukryte) i wyjściową. Między neuronami wejściowymi nie mogą występować połączenia, podobnie jak między wyjściowymi. Złożoność pamięciowa dla pojedynczej sieci neuronowej wynosi $O(N^2)$. Przykład genotypu i kodowanej w nim sieci pokazany jest na (rys. 26).

Dla kodowania sieci rekurencyjnych wykorzystujemy całą macierz E . Sposób kodowania i operatory genetyczne są te same. Przykład chromosomu i zakodowanej sieci pokazany jest na rysunku 27.



Rys. 27. Przykład kodowania bezpośredniego sieci neuronowej ze sprzężeniem zwrotnym i odkodowana sieć neuronowa

Operatory genetyczne: kodowanie bezpośrednie za pomocą macierzy połączeń nie wymaga skomplikowanych, specjalizowanych operatorów genetycznych. Zaproponowano cztery rodzaje krzyżowania:

1. *Krzyżowanie jednopunktowe* polega na wyborze losowej liczby l z zakresu $[0, N - 1]$ i skopiowaniu pierwszych l wierzchołków i połączeń z pierwszego rodzica, pozostałych z drugiego rodzica.

2. *Krzyżowanie wielopunktowe po wierszach* to losowy wybór każdego elementu wektora V (wierzchołka) i każdego wiersza macierzy E z jednego z dwóch rodziców, prawdopodobieństwo wyboru jest jednakowe.

3. *Krzyżowanie wielopunktowe po kolumnach* jest podobne do poprzedniego, przy czym od rodziców wybierane są całe kolumny macierzy E .

4. *Krzyżowanie wielopunktowe losowe* to wybór pojedynczych elementów wektora V i macierzy E losowo od jednego z rodziców.

Mutacja to dwa proste operatory:

1. *Mutacja macierzy E* dodaje lub usuwa połączenia między neuronami, zamieniając jedynki na zera i odwrotnie,

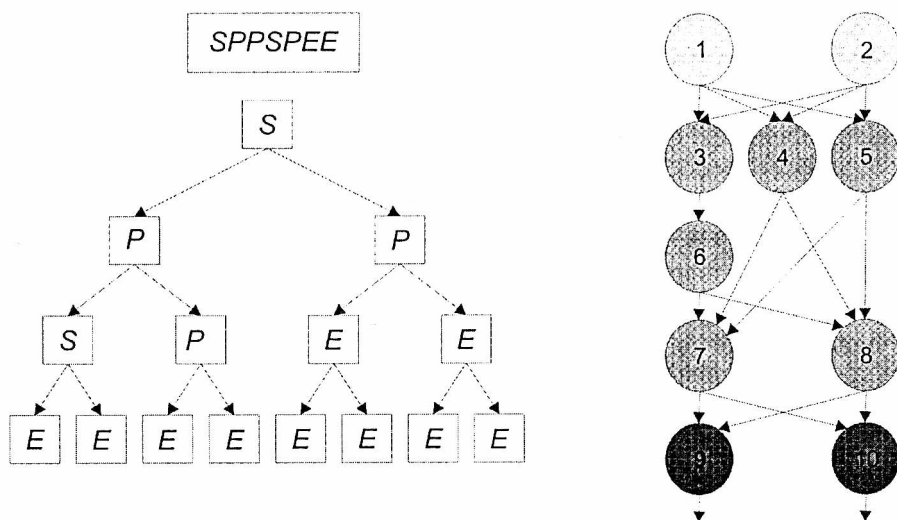
2. *Mutacja wektora V* dodaje lub usuwa neurony w sieci. Wartość zero na pozycji i w wektorze V oznacza, że neuron o numerze i jest wyłączany z sieci. Podczas dekodowania ten neuron nie jest brany pod uwagę.

7.2.2. KODOWANIE POŚREDNIE

Kodowanie pośrednie jest pewnego rodzaju kompromisem pomiędzy złożonością i czasem obliczeń a jakością uzyskiwanych wyników (Lieske 1995). Najbardziej istotną cechą w tym podejściu jest zmniejszenie rozmiaru przestrzeni rozwiązań. Powoduje to, że projektowanie sieci neuronowych przy stosowaniu kodowania pośredniego jest szybsze, ale może prowadzić do rozwiązań suboptymalnych. Kodo-

wanie pośrednie zazwyczaj ma mniejsze wymagania pamięciowe, reprezentacja sieci neuronowej i o N neuronach wymaga $O(N)$ pamięci.

Osobnik: Zastosowane w tej pracy kodowanie pośrednie polega na reprezentacji architektury sieci neuronowych w postaci drzewa binarnego i wymaga stosowania ustalonego zbioru reguł gramatycznych do konstrukcji sieci neuronowej (proces dekodowania osobnika). Chromosom jest ciągiem znaków, reprezentujących drzewo binarne o określonej głębokości. Każdy węzeł drzewa zawiera jeden z dwóch symboli nieterminalnych – $\{S, P\}$ lub symbol terminalny E . Symbol S oznacza podział aktywnego neuronu na dwa neurony połączone szeregowo. Symbol P oznacza podział aktywnego neuronu na dwa, równolegle połączone neurony. Każdy liść w drzewie jest symbolem terminalnym E (patrz rys. 28). Znaczenie poszczególnych symboli pokazuje tabela 2.



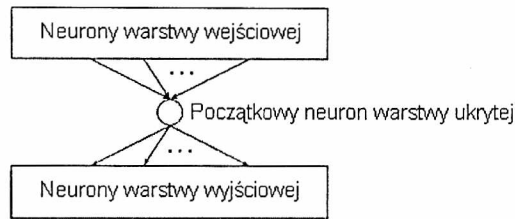
Rys. 28. Przykład kodowania pośredniego sieci neuronowej i odkodowana sieć neuronowa

Tabela 2. Symbole i ich semantyka w kodowaniu pośrednim

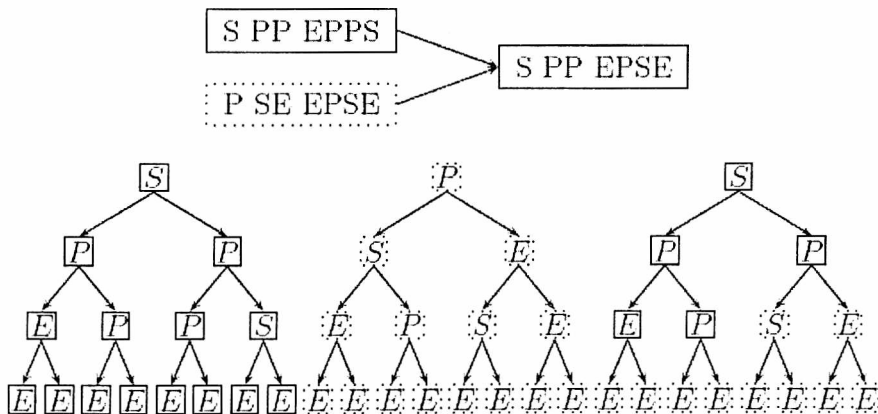
Symbol	Semantyka
S	Sekwencyjny podział aktualnego neuronu, nowo powstały neuron przejmie wszystkie wyjścia neuronu aktualnego, wyjście neuronu aktualnego jest połączone tylko i wyłącznie z wejściem neuronu powstałego, wejście neuronu powstałego jest połączone tylko i wyłącznie z wyjściem neuronu aktualnego.
P	Równoległy podział aktualnego neuronu, aktualny neuron nie podlega żadnym zmianom, powstały z niego neuron jest jego dokładną kopią – ma identyczne połączenia wejściowe oraz wyjściowe
E	Symbol terminalny, aktualny neuron nie podlega żadnym kolejnym podziałom

Wszystkie liście drzewa muszą zawierać symbol terminalny *E*. Jako że informacja zawarta w liściach jest niezmienna i nadmiarowa, nie mają one bezpośredniej reprezentacji w genotypie. Proces dekodowania genotypu uzupełnia niepełne drzewo o wszystkie liście tak, aby każdy z nich zawierał symbol terminalny *E*.

Proces dekodowania genotypu do fenotypu jest, zgodnie z założeniami teoretycznymi, dużo bardziej skomplikowany niż w przypadku kodowania bezpośredniego (Whitley 1995). Składa się on z wielu kroków, z których każdy jest wykonywany dla pojedynczego symbolu w genotypie. Dekodowanie genotypu do fenotypu wymaga zdefiniowania niezmiennego stanu początkowego sieci neuronowej. Stan ten, przedstawiony na rysunku 29, składa się z warstwy wejściowej, warstwy wyjściowej oraz pojedynczego neuronu w warstwie pośredniej. Wszystkie neurony z warstwy wejściowej są połączone z neuronem w warstwie ukrytej. Podobnie neuron w warstwie ukrytej jest połączony ze wszystkimi neuronami warstwy wyjściowej.

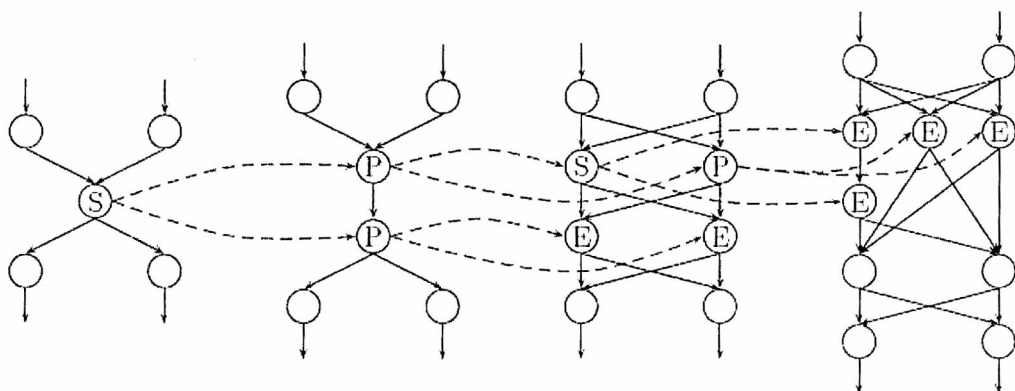


Rys. 29. Stan początkowy sieci w kodowaniu pośrednim



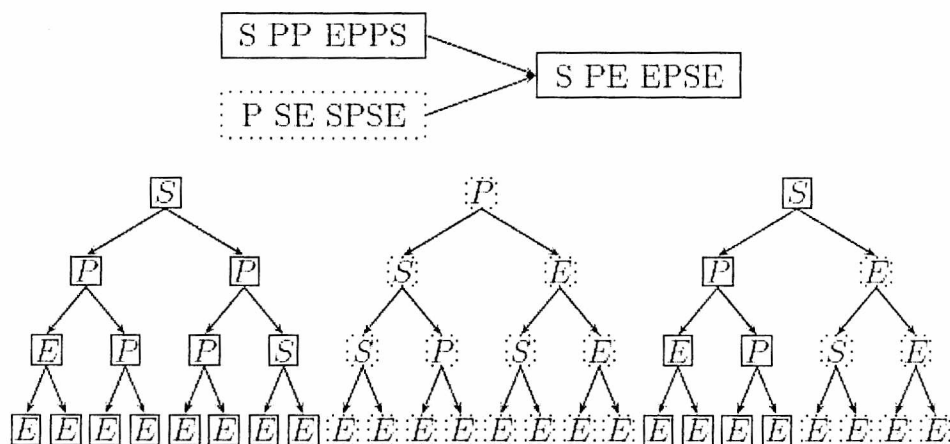
Rys. 30. Przykład krzyżowania przez wymianę liści

Konstrukcja sieci neuronowej następuje w sposób rekurencyjny, rozpoczyna się od neuronu w warstwie ukrytej. Bieżący symbol jest symbolem z korzenia drzewa. W zależności od tego, czy jest tam symbol *S* czy *P*, tworzone są dwa neurony zgodnie z tabelą 2. Dla jednego neuronu symbolem aktualnym staje się symbol znajdujący się w korzeniu lewego poddrzewa, dla drugiego neuronu – symbol znajdujący się w korzeniu prawego poddrzewa.



Rys. 31. Przykład kodowania pośredniego sieci neuronowej i odkodowana sieć neuronowa

Podział ten jest wykonywany rekurencyjnie, aż do napotkania symboli terminalnych (E). Przykładowy genotyp z kodowaniem pośrednim oraz sieć neuronowa, która jest z niego wygenerowana, przedstawione są na rysunku 28. Na rysunku 30 pokazano proces dekodowania genotypu na fenotyp, czyli dekodowanie drzewa na sieć neuronową.



Rys. 32. Przykład krzyżowania przez wymianę poddrzew

Operatory genetyczne. Dla tej reprezentacji należy zaprojektować specjalizowane operatory genetyczne. W niniejszej pracy zaproponowano dwa rodzaje krzyżowania (Paradowski 2004): krzyżowanie liści, jego działanie pokazano na rysunku 31 i krzyżowanie poddrzew, pokazane na rysunku 32.

Mutacja polega na zamianie pojedynczego symbolu w drzewie na inny, losowo wybrany symbol. Efekty mutacji pokazane są w tabeli 3.

Tabela 3. Rezultaty zmiany mutacyjnej w kodowaniu pośrednim

Zamiana symbolu	Efekt zamiany
S na S	Brak zmiany w fenotypie (sieci neuronowej)
S na P	Zmiana podziału sekwencyjnego na równoległy dla mutowanego neuronu
S na E	Brak dalszego podziału neuronu, pominięcie wszystkich podziałów zdefiniowanych w poddrzewie mutowanego węzła
P na S	Zmiana podziału równoległego na sekwencyjny dla mutowanego węzła (neuronu). Brak zmian w fenotypie
P na P	Brak dalszego podziału neuronu, pominięcie wszystkich podziałów zdefiniowanych w poddrzewie mutowanego węzła (neuronu)
P na E	
E na S	Podział sekwencyjny oraz aktywacja wszystkich podziałów neuronu zdefiniowanych w poddrzewie mutowanego neuronu
E na P	Podział równoległy oraz aktywacja wszystkich podziałów neuronu zdefiniowanych w poddrzewie mutowanego neuronu
E na E	Brak zmian w fenotypie

7.3. FUNKCJA PRZYSTOSOWANIA

Funkcja przystosowania to bardzo ważny element działania algorytmów ewolucyjnych, decydujący w dużej mierze o skuteczności i efektywności działania algorytmu. Powinna brać pod uwagę specyfikę rozwiązywanego problemu i wykorzystywać wiedzę dziedzinową o problemie.

W niniejszej pracy całkowita funkcja oceny sieci neuronowej składa się z kombinacji dwóch funkcji przystosowania, oceniających różne cechy uzyskanej sieci. Tymi składowymi finalnej oceny są:

- ocena uczenia f_t (indeks t to skrót od ang. *train*),
- ocena rozmiaru f_s (indeks s to skrót od ang. *size*).

Ocena uczenia opisuje efektywność uczenia się wygenerowanej sieci neuronowej, natomiast ocena rozmiaru jest funkcją liczby neuronów i połączeń wchodzących w skład wygenerowanej sieci neuronowej.

W pracy zdefiniowano trzy funkcje przystosowania, wykorzystujące wymienione elementy w różny sposób.

7.3.1. OCENA UCZENIA SIĘ SIECI

Proces uczenia i testowania generowanej sieci neuronowej jest przeprowadzany m razy, dla każdego z s podziałów zbioru. Wartość funkcji oceny uczenia jest wartością maksymalną ze wszystkich m procesów uczenia. Metoda uczenia ze wsteczną propagacją błędu ma tendencję do utykania w lokalnych minimach, otrzymywany wynik silnie zależy od początkowego zestawu wag, który jest wybierany losowo. Dla $m = 1$ czynnik losowy wpływający na wynik jest zbyt duży. Dla większych wartości m (np. $m = 3$) czas obliczeń wzrasta m -krotnie, lecz wyniki są bardziej wiarygodne. Pozwala

to z większym zaufaniem stwierdzić, czy dana sieć neuronowa jest w stanie nauczyć się postawionego przed nią zadania.

Metoda uczenia za pomocą algorytmu genetycznego znacznie lepiej unika utykania w lokalnych minimach. Dzięki temu losowo wybrany początkowy zestaw wag ma mniejszy wpływ na wynik, niż w przypadku metody uczenia z propagacją wsteczną. Genetyczne uczenie daje dobre wyniki nawet dla $m = 1$.

Sumaryczna wartość błędu sieci jednokierunkowej, działającej na zbiorze testowym liczącym k przykładów, wyraża się wzorem:

$$e = \sum_{i=1}^k e_i = \frac{1}{2} \sum_{i=1}^k \sum_{j=1}^{n_o} (o_j^i - s_j^i)^2 \quad (9)$$

gdzie: k – liczba przykładów w zbiorze testowym, wykorzystanym do oszacowania sieci, n_o – liczba neuronów wyjściowych, o_j^i – wartość generowana na wyjściu j sieci dla k -tego przykładu ze zbioru testowego, s_j^i – wartość, jaka powinna być generowana na j -tym wyjściu sieci dla k -tego przykładu ze zbioru testowego (poprawna).

Sumaryczna wartość błędu sieci ze sprzężeniem zwrotnym (sieci rekurencyjnej) wyraża się wzorem:

$$e = \sum_{t=1}^p e^t = \sum_{t=1}^p \sum_{i=1}^m e_i^t = \frac{1}{2} \sum_{t=1}^p \sum_{i=1}^m \sum_{y=1}^{n_o} (o_y^t - s_y^t)^2 \quad (10)$$

gdzie: m – liczba procesów uczenia–testowania, p – liczba cykli przetwarzania w sieciach rekurencyjnych, o_y^t – uzyskana wartość wyjścia y sieci w czasie t , s_y^t – oczekiwana, prawidłowa wartość wyjścia y sieci w czasie t .

Aby wyznaczyć składnik funkcji oceny informujący o jakości uczenia się sieci f_i , wprowadźmy dodatkowe oznaczenia:

f_i^{ji} – wartość oceny uczenia w i -tym podziale zbioru danych, w j -tym procesie uczenia–testowania,

f_i^i – wartość oceny uczenia w i -tym podziale zbioru danych.

Wartość funkcji oceny f_i jest uśrednionym, maksymalnym błędem sieci uzyskanym we wszystkich podziałach zbioru danych i wyraża się wzorem:

$$f_i = \frac{1}{s} \sum_{s=1}^s f_i^i \quad (11)$$

gdzie: s – liczba stosowanych podziałów zbioru danych na ciąg uczący i testowy, a wartość oceny uczenia w i -tym podziale danych wynosi:

$$f_t^i = \max(f_t^{1i}, f_t^{2i}, \dots, f_t^{mi}) \quad (12)$$

m – liczba procesów uczenia–testowania w jednym podziale danych na ciąg uczący–testujący.

$$f_t^{ji} = \left(\frac{\max(2n_o k - e, 0)}{2n_o k} \right)^r \quad (13)$$

gdzie: e – błąd sumaryczny sieci uzyskany w j -tym procesie uczenia–testowania, n_o – liczba neuronów wyjściowych w sieci, r – jest przyjętym współczynnikiem potęgowania, dobranym eksperymentalnie.

Podane wzory zostały zaproponowane po analizie wyników wielu eksperymentów. Jak łatwo zauważyć, wartość oceny jakości uczenia się sieci jest znormalizowana, co pozwala na bardziej efektywną konstrukcję końcowej funkcji oceny generowanych sieci neuronowych. Podnoszenie f_t^{ji} do potęgi r pozwala na elastyczne ustalanie presji selekcyjnej na poprawnie nauczone sieci. Wzór (13) powoduje, że sieci, które nie są w stanie poprawnie nauczyć się przeznaczonego im zadania otrzymują ocenę bliską zeru, co powoduje, że będą łatwiej eliminowane z populacji. Sieci dobrze realizujące dane zadanie będą mieć tę ocenę bliską jedności.

Zdefiniujemy jeszcze pomocnicze wielkości, pomocne w definiowaniu funkcji oceny generowanych sieci neuronowych:

1. Maksymalny, dopuszczalny błąd sieci, $e^{\max}$.
2. Minimalną (dopuszczalną) ocenę jakości uczenia $f_t^{\min} = (1 - e^{\max})^r$.
3. Warunek *zdania testu uczenia* – mówimy, że sieć *zdała test uczenia*, jeśli dla danej sieci prawdziwa jest nierówność: $f_t > f_t^{\min}$.

7.3.2. OCENA ROZMIARU GENEROWANYCH SIECI

Podobnie jak ocena jakości uczenia sieci f_i ocena rozmiaru sieci f_s też jest znormalizowana. Zdefiniowano dwie wersje funkcji oceny rozmiaru sieci – f_s^1 i f_s^2 . Obie funkcje różnią się tym, że pierwsza ocenia rozmiar sieci niezależnie od jakości działania sieci (czy potrafi się uczyć danego zadania). Natomiast druga funkcja jest liczona tylko dla tej części sieci, które spełniają test uczenia. Inaczej mówiąc, rozmiar sieci jest brany pod uwagę tylko dla takich sieci, które nauczyły się z zadaną dokładnością. Przyjęto następujące oznaczenia: N_Σ – rozmiar ewoluującej populacji, r_i – liczba połączeń w sieci kodowanej przez i -ty osobnik, N_p – liczba osobników (fenotypów), które zdały test uczenia, r_i^p – przeskalowana wartość r_i , x_i – liczba połączeń w i -tym osobniku, który zdał test uczenia, x_i^p – przeskalowana wartość x_i .

Ocena rozmiaru sieci oraz wielkości pomocnicze są zdefiniowane następująco:

$$x_i = 0,1 + \frac{\sum_{i=1}^{N_\Sigma} r_i}{N_\Sigma} - r_i \quad (14)$$

x_i niesie informację, jak dużo połączeń ma dany osobnik (sieć) na tle innych osobników w populacji. Wartość 0,1 w podanym wzorze dodana jest, aby wartość x_i była różna od zera. x_i jest największe dla sieci z najmniejszą liczbą połączeń. Wzór (15) jest podobny, ale bierze pod uwagę tylko te sieci, które przeszły pozytywnie test uczenia.

$$x_i^p = 0,1 + \frac{\sum_{i=1}^{N_p} r_i^p}{N_p} - r_i^p \quad (15)$$

$$f_s^1 = \frac{x_i}{\max(x_1, x_2, \dots, x_{N_\Sigma})} \quad (16)$$

$$f_s^2 = \frac{x_i^p}{\max(x_1^p, x_2^p, \dots, x_{N_p}^p)} \quad (17)$$

Zauważmy, że w tym przypadku, wartości obu funkcji, f_s^1 i f_s^2 są bliższe zeru wtedy, gdy sieć jest jedną z większych sieci w całej populacji (wartość f_s^1) lub odpowiednio, w tej części populacji, która zdała test uczenia (f_s^2). Wartości bliskie jedności oznaczają, że oceniane sieci należą do jednych z mniejszych w populacji lub, w przypadku f_s^2 , wśród sieci, które zdały test uczenia. Wartość f_s^2 jest równa jedynie dla najlepszych sieci w populacji – są to małe sieci, które zdały test uczenia.

7.3.3. TRZY FUNKCJE PRZYSTOSOWANIA O RÓŻNYCH PREFERENCJACH

Pierwsza z funkcji przystosowania kładzie duży nacisk na otrzymanie sieci neuronowych, które są dobrze nauczone przeznaczonego dla niej zadania. W ewolucji można tu wyróżnić dwie fazy: w pierwszej fazie ewolucja stara się wytworzyć sieci neuronowe, zdające test uczenia. Dopiero, gdy ewoluowane sieci mają akceptowalne błędy działania, włączane jest ciśnienie selekcyjne w kierunku zmniejszania rozmiaru sieci, dbając nadal o ich błąd uczenia. Funkcja ta, oznaczona jako f_1 , wyraża się wzorem:

$$f_1 = \begin{cases} f_t & \text{gdy } f_t \leq f_t^{\min} \\ f_t(1 + f_s^1) & \text{gdy } f_t > f_t^{\min} \end{cases} \quad (18)$$

Funkcja przystosowania f_2 , (wzór (19)) jest bardzo podobna do funkcji f_1 . Główną różnicą jest inaczej skierowana presja na minimalizację rozmiaru sieci. Dla sieci, które zdały test uczenia presja ta występuje zawsze. Dzieje się tak, ponieważ oceny rozmiaru określone są tylko dla tych sieci, które zdały test – wykorzystywana jest funkcja f_s^2 zamiast f_s^1 . Wartość oceny rozmiaru, która w funkcji f_1 mogła być równa 0 dla wielu sieci, które zdały test uczenia, w przypadku funkcji f_s^2 może być równa 0 tylko i wyłącznie dla najmniejszej sieci, która zdała test uczenia

$$f_2 = \begin{cases} f_t & \text{gdy } f_t \leq f_t^{\min} \\ f_t(1 + f_s^2) & \text{gdy } f_t > f_t^{\min} \end{cases} \quad (19)$$

Trzecia funkcja przystosowania, f_3 , stara się równoważyć promowanie małych sieci i sieci dobrze wyuczonych, już od samego początku ewolucji. Nie bierze pod uwagę dopuszczalnego, minimalnego błędu; sieci nie są rozróżniane na te, co zdały test uczenia i pozostałe. Wszystkie są oceniane od początku ewolucji, biorąc pod uwagę zarówno błąd działania, jak i rozmiar. Wartość f_3 dana jest wzorem

$$f_3 = f_t(1 + f_s^1) \quad (20)$$

Wszystkie trzy funkcje przystosowania, f_1 , f_2 i f_3 , są przedmiotem badań eksperymentalnych. Przesłanką do zdefiniowania i badań tych trzech funkcji są wcześniejsze prace autorki, prezentowane w pracy (Kwaśnicka, Kilińska 1995). Ówczesne badania wskazywały, że całkiem dobre rezultaty można otrzymać, prowadząc ewolucję w dwóch oddzielnych fazach: ewolucja sieci neuronowych poprawnie realizujących wyznaczone zadania (w tej fazie ocena sieci opiera się na błędzie odpowiedzi), natomiast – po uzyskaniu sieci poprawnie realizujących zadanie, kontynuacja ewolucji mająca na uwadze zmniejszanie rozmiaru sieci, bez pogarszania jej działania. Wyniki te nasuwały pytanie, czy można w jednej funkcji oceny połączyć oba elementy – poprawność działania i rozmiar sieci, w efektywny sposób.

7.4. STOSOWANE ALGORYTMY UCZENIA SIECI

W niniejszej pracy wykorzystano algorytm uczenia z nadzorem, co oznacza, że ciąg uczący musi zawierać wektory w postaci: <wejścia, wyjście>. Wykorzystano dwa algorytmy uczące:

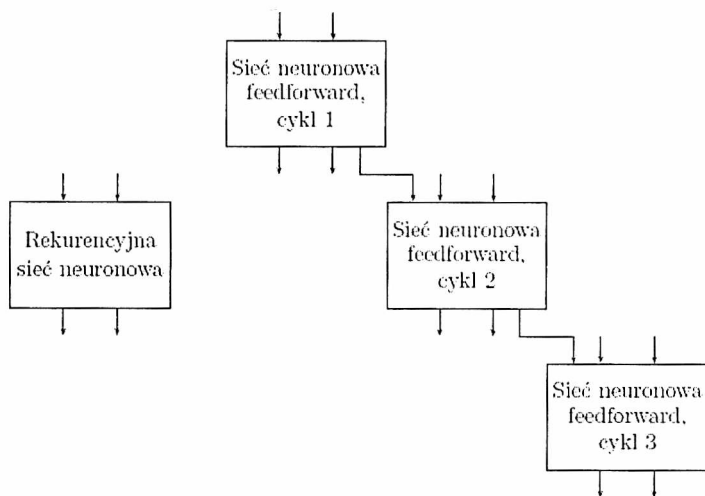
1. Algorytm wstecznej propagacji błędów.

2. Algorytm genetyczny.

Zarówno algorytm wstecznej propagacji błędów, jak i algorytm genetyczny, są dość złożone obliczeniowo, co powoduje, że eksperymenty wymagają dość dużo czasu.

Algorytm **wstecznej propagacji błędów** jest klasyczną metodą uczenia sieci neuronowych bez sprzężenia zwrotnego. Jest też dość znany i opisany w podstawowych podręcznikach, np. (Tadeusiewicz 1993, Żurada i in. 1996). Skupimy się krótko na przedstawieniu sposobu uczenia sieci neuronowych ze sprzężeniem zwrotnym, wykorzystywanych w prezentowanych w niniejszym rozdziale badaniach.

Algorytm wstecznej propagacji błędów jest powszechnie wykorzystywany w uczeniu jednokierunkowych sieci neuronowych. Z uwagi na fakt, że jest to gradientowa metoda optymalizacji, jego wynik silnie zależy od początkowego, zazwyczaj losowego, zestawu wag. Algorytm ten może być zastosowany do pewnego rodzaju sieci neuronowych ze sprzężeniem zwrotnym – rekurencyjnych sieci neuronowych z dyskretnym czasem, realizujących zadania generowania sekwencji. Do takich sieci ograniczono się w prezentowanych tu badaniach. Jest to rozszerzona wersja podstawowego algorytmu, tzw. algorytm wstecznej propagacji błędów przez czas. Algorytm ten opiera się na rozwinięciu sieci rekurencyjnych do sieci bez sprzężeń zwrotnych. Rozwinięcie to polega na wielokrotnym skopiowaniu sieci rekurencyjnej z pominięciem sprzężeń zwrotnych, liczba tych kopii jest równa liczbie cykli przetwarzania wykorzystywanych przez sieć. Idea tego przetwarzania pokazana jest na rysunku 33.



Rys. 33. Zamiana sieci rekurencyjnej na cykl sieci jednokierunkowych

Uczenie genetyczne – osobnik i operatory genetyczne. W proponowanym podejściu, algorytm genetyczny jest wykorzystany do generowania architektury sieci neuronowej, rozwiązującej określone zadanie, opcjonalnie może też być wykorzystywany jako metoda uczenia. Jest to odrębny algorytm genetyczny, z osobnikiem będącym macierzą wag (wzór (21)).

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & \ddots & & \\ \vdots & & \ddots & \\ w_{n1} & \cdots & \cdots & w_{nn} \end{bmatrix} \quad (21)$$

gdzie n jest ustaloną maksymalną liczbą neuronów w sieci, a w_{ij} jest wagą połączenia pomiędzy wyjściem neuronu i a wejściem neuronu j (kodowanie rzeczywiste).

Podczas dekodowania macierzy na fenotyp pomijane są te wagi, dla których nie ma połączenia w rozważanej architekturze sieci.

Różnica pomiędzy uczeniem sieci jednokierunkowych a rekurencyjnych polega na różnym sposobie przetwarzania informacji przez sieć oraz na mechanizmie obliczania błędu sieci, stosowanym do oceny sieci. Ocena osobników wykonywana jest z wykorzystaniem jednej spośród zdefiniowanych funkcji przystosowania, na podstawie testów wyuczonej sieci. Sposób oceny generowanych sieci neuronowych jest jednakowy zarówno dla uczenia metodą wstecznej propagacji błędów, jak i uczenia genetycznego. Pozwala to na porównanie efektywności uczenia obiema metodami.

Operatory genetyczne. Krzyżowanie dla tych osobników nie różni się niczym od krzyżowań stosowanych w ewolucji architektury sieci na macierzy E . Odmienny jest proces mutacji z uwagi na niebinarny charakter macierzy W . Parametrami procesu definiowanymi przez użytkownika programu są prawdopodobieństwo mutacji pojedynczej wagi p_m oraz zasięg mutacji s_m . Parametrem wpływającym na częstość i zakres mutacji jest liczona w systemie liczba pokoleń bez poprawy l . Schemat działania operacji mutacji wag jest następujący:

Dla każdej wagi w_{ij} wykonaj
 wylosuj liczbę całkowitą los z przedziału $[0, l]$,
 wylosuj liczbę losową d z zakresu $\left[-\frac{s_m}{2l}, \frac{s_m}{2l}\right]$,
 jeśli $los \leq l$, to zmień wagę: $w_{ij} = w_{ij} + los$.

Ewolucja architektury sieci neuronowych wraz z uczeniem genetycznym jest zadaniem bardzo czasochłonnym, wymaga bowiem ewolucji wag sieci dla każdego osobnika–architektury w każdym pokoleniu ewolucji architektur sieci. Dlatego też, jako warunek zakończenia zastosowano wystąpienie jednego z dwóch warunków:

- założona maksymalna liczba pokoleń,
- założona liczba pokoleń bez poprawy najlepszego rozwiązania.

Zaproponowane operatory działają poprawnie zarówno dla sieci jednokierunkowych, jak i rekurencyjnych sieci z dyskretnym czasem przetwarzania.

8. BADANIA EKSPERYMENTALNE

Omówiona w poprzednim rozdziale metoda ewolucyjnego generowania sieci neuronowych została przebadana na kilku wybranych przykładach. W eksperymentach wykorzystano specjalnie wykonaną aplikację GANN⁶⁶.

Do systematycznych badań wybrano dwie grupy problemów:

- Sześć problemów testowych wybranych spośród powszechnie stosowanych w badaniach, bez jawnych zależności czasowych, które można rozwiązać, stosując jednokierunkowe sieci neuronowe:

1. XOR,
2. TERMOMETR,
3. PARZYSTOŚĆ,
4. ZOO,
5. IRIS,
6. HOUSING.

- Dwa problemy z jawnymi problemami czasowymi, które są dostosowane do rekurencyjnych sieci neuronowych z dyskretnym czasem.

1. Krótki ciąg arytmetyczny.
2. Długi ciąg arytmetyczny.

8.1. CHARAKTERYSTYKA ZBIORÓW UCZĄCYCH

Z uwagi na długi czas obliczeń starano się ograniczać liczbę zbiorów z danymi uczącymi, zwracając jednocześnie uwagę na ich różnorodność. Krótką charakterystykę zbiorów uczących dla problemów bez jawnej zależności czasowej zawiera tabela 4. Problemy te zawierają niejednorodne atrybuty (przyjmujące wartości dyskretne, nu-

⁶⁶ Program został zaimplementowany w ramach pracy magisterskiej studenta w Instytucie Informatyki Stosowanej, Mariusza Paradowskiego, w języku C++. Student wykonał również część eksperymentów.

meryczne i symboliczne oraz ciągłe), różną liczbę wejść i wyjść sieci, różną liczbę przykładów uczących.

Tabela 4. Charakterystyka problemów bez jawnej zależności czasowej

Parametr	XOR	ZOO	IRIS	TERMOMETR	PARZYSTOŚĆ	HOUSING
Liczba wejść	2	21	4	9	5	13
Liczba wejść binarnych	2	21	0	9	5	1
Liczba wyjść	1	7	3	9	1	1
Liczba wyjść binarnych	1	7	3	9	1	0
Liczba przykładów	8	101	150	10	32	506
Tylko atrybuty binarne	tak	tak	nie	tak	tak	nie

TERMOMETR polega na tym, aby liczby zadane na wejściu w postaci binarnej przedstawić na wyjściu w postaci „słupkowej”. Wejściem jest zakodowana binarnie liczba dziesiętna od 0 do 7. Na wyjściu oczekiwane jest tyle zapalonych neuronów wyjściowych, ile wynosi wartość dziesiętna wejścia, ułożonych kolejno od „lewej do prawej” (symulacja termometru).

PARZYSTOŚĆ – zadana jest liczba w postaci binarnej, sieć musi nauczyć się odpowiadać, czy zadana na wejściu liczba jest parzysta. W pozostałych zbiorach należy, na podstawie różnej liczby cech charakteryzujących obiekty (rośliny, zwierzęta), określić ich przynależność do jednej z wielu klas.

Charakterystyka wybranych problemów z jawną zależnością czasową pokazana jest w tabeli 5. Zadaniem problemu „Krótki ciąg arytmetyczny” jest weryfikacja, czy metoda jest w stanie wygenerować sieć neuronową, która potrafi nauczyć się rekurencyjnej zależności, danej wzorem:

$$\begin{aligned} a_0 &= x, \\ a_t &= a_{t-1} + y, \text{ dla } t > 0. \end{aligned} \quad (22)$$

Zadaniem sieci jest wygenerowanie pierwszych pięciu elementów ciągu.

Zadanie „Długi ciąg arytmetyczny” to wygenerowanie pierwszych dziesięciu elementów tego samego ciągu (wzór (22)).

Tabela 5. Charakterystyka problemów z jawną zależnością czasową

Parametr	Krótki ciąg arytmetyczny	Długi ciąg arytmetyczny
Liczba wejść	2	2
Liczba wyjść	1	1
Liczba cykli przetwarzania	5	10
Sumaryczna liczba wyjść	5	10
Liczba przykładów	15	18

Dane uczące, opisujące dany problem, muszą być przygotowane do tego, aby je wykorzystać do automatycznego projektowania sieci neuronowych.

Pierwszy element to skalowanie danych wejściowych. Każda wartość wejściowa i wyjściowa w całym zbiorze danych jest skalowana do przedziału $[-1; 1]$. Dotyczy to zarówno danych wejściowych, jak i wyjściowych.

Kolejnym elementem jest podział zbioru danych na ciąg uczący i ciąg testowy. Najczęściej stosuje się tu następujące podejścia:

- Dwie trzecie – jedna trzecia (*TwoThirds – OneThird*): dwie trzecie przykładów jest wybierana jako ciąg uczący, jedna trzecia służy do testowania wyuczonej na ciągu uczącym sieci.

- Dziesięciokrotna skrośna walidacja (*10 – Fold Cross*) – cały zbiór danych dzielony jest na dziesięć części, jedna część jest wykorzystywana jako ciąg testowy, pozostałe dziewięć części to ciąg uczący. Badania wykonuje się 10 razy, pozostawiając za każdym razem inny podzbiór jako zbiór testowy.

- Pozostaw jeden na zewnątrz (*Leave One Out*) – badania powtarza się tyle razy, ile jest przykładów w zbiorze, pozostawiając za każdym razem jeden, inny przykład jako jednoelementowy zbiór testowy.

W badaniach zaproponowanej metody stosowano zwykle pierwszy sposób podziału, z uwagi na fakt, iż jest on najmniej czasochłonny i daje zadowalające wyniki w praktycznych zadaniach.

8.2. WYNIKI EKSPERYMENTÓW Z PROBLEMAMI BEZ JAWNYCH ZALEŻNOŚCI CZASOWYCH

W niniejszym rozdziale omówione są tylko wybrane, reprezentatywne wyniki działania zaproponowanej metody dla problemów bez jawnych zależności czasowych. Jak wiadomo, problemy te są odpowiednie dla sieci jednokierunkowych.

Głównym celem przeprowadzonych eksperymentów była ocena skuteczności generowania sieci neuronowych w zależności od stosowanej funkcji przystosowania i szukanie odpowiedzi, jaki faktyczny wpływ na efektywność metody ma sposób kodowania sieci neuronowych w chromosomie.

W programie zaimplementowano możliwość uczenia genetycznego bardziej jako ciekawostkę, niż obiecującą metodę. Wprawdzie w literaturze można znaleźć nieliczne przykłady, kiedy to genetyczne uczenie daje lepsze wyniki niż tradycyjne, metodą wstecznej propagacji błędów, ale po pierwsze są to nieliczne przykłady (np. Montana, Davis 1989), po drugie, wiele przykładów literaturowych i dotychczasowe doświadczenia autorki wskazują na lepszą efektywność tradycyjnego uczenia (Kwaśnicka

1999)⁶⁷. Okazuje się jednak, że w połączeniu z odpowiednią metodą kodowania, uczenie genetyczne może dać zaskakująco dobre wyniki.

Zbudowany system GANN generowania sztucznych sieci neuronowych za pomocą algorytmu genetycznego umożliwia szerokie badania jego właściwości i efektywności dla następujących przypadków:

1. Kodowanie bezpośrednie, ciśnienie selekcyjne ukierunkowane na poprawność sieci neuronowych (bardziej preferowane są sieci lepiej nauczone).

2. Kodowanie bezpośrednie, ciśnienie selekcyjne preferujące mały rozmiar sieci neuronowych.

3. Kodowanie pośrednie, ciśnienie selekcyjne ukierunkowane na poprawność sieci neuronowych.

4. Kodowanie pośrednie, ciśnienie selekcyjne preferujące mały rozmiar sieci neuronowych.

5. Generowanie sieci jednokierunkowych bądź rekurencyjnych, pracujących w dyskretnym czasie.

Dla każdego z podanych przypadków można stosować uczenie tradycyjne bądź uczenie genetyczne.

Po analizie wszystkich wykonanych eksperymentów wydaje się, że najbardziej interesujące wnioski dotyczą takich elementów, jak:

1. Jakie są różnice w działaniu metody przy stosowaniu każdej spośród trzech proponowanych funkcji przystosowania?

2. Jaki wpływ na proces ewolucji topologii sieci neuronowych ma sposób kodowania?

3. Jaka jest efektywność uczenia metodą wstecznej propagacji błędów i uczenia genetycznego?

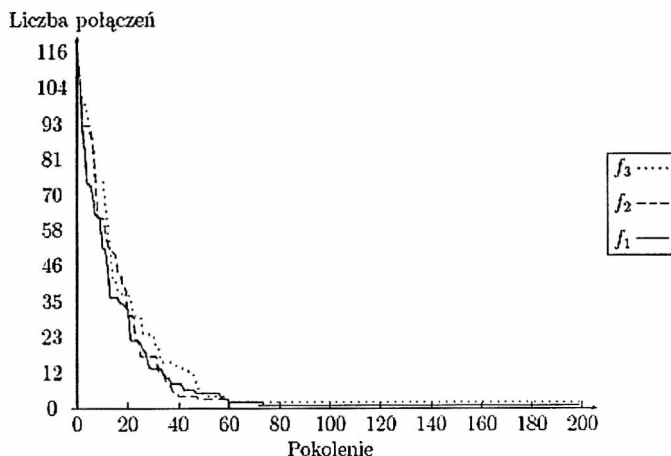
Dla wszystkich, wykorzystanych w eksperymentach problemów, zaproponowana metoda ewolucji sieci neuronowych dobrze sobie radziła. Generowała małe, dobrze nauczone sieci neuronowe. Wyjątek stanowił problem *Termometr*, dla którego nie udało się wyewoluować poprawnie działającej sieci neuronowej. Analiza problemu pozwoliła znaleźć odpowiedź na pytanie, dlaczego ten, w sumie niezbyt trudny problem, okazał się nie do rozwiązania. Okazało się, że cały ciąg uczący zawiera po jednym przykładzie na każdą klasę. Wydzielenie ciągów uczącego i testowego powodowało, że w danych uczących nie było ani jednego przykładu reprezentującego klasę, lub klasy, z ciągu testowego. Sieć nie może nauczyć się rozpoznawać klasy, której się nie uczyła w trakcie uczenia. W eksperymentach, w których nie wydzielano ciągu testowego, sieć neuronowa uczyła się prawidłowo tego zadania, generowane sieci były małych rozmiarów. Niestety, nie ma tu mowy o zdolnościach do uogólniania sieci, raczej o nauczeniu się, być może na pamięć.

⁶⁷ Patrz rozdział 5.2 – niezbyt dobre wyniki różnych prób genetycznego uczenia sieci neuronowych w stosunku do uczenia metodą wstecznej propagacji błędów.

Prezentacja wszystkich wykonanych eksperymentów zajęłaby dużo miejsca, a nie wносиłaby wiele. Dlatego też skupimy się na najważniejszych wnioskach, jakie wynikają z przeprowadzonych eksperymentów.

8.2.1. WPLYW FUNKCJI PRZYSTOSOWANIA NA EFEKTYWNOŚĆ METODY

Przypomnijmy tutaj, że spośród trzech zdefiniowanych i badanych funkcji przystosowania, f_1 promuje na początku ewolucji ocenę uczenia, później włącza ocenę rozmiaru sieci, funkcja f_2 uwzględnia zawsze rozmiar sieci, jeśli spełnia ona test uczenia, a f_3 nie narzuca podziału ewolucji na etapy preferujące jakość działania sieci i minimalizację rozmiaru sieci, bierze oba czynniki pod uwagę od początku ewolucji, dla wszystkich sieci.

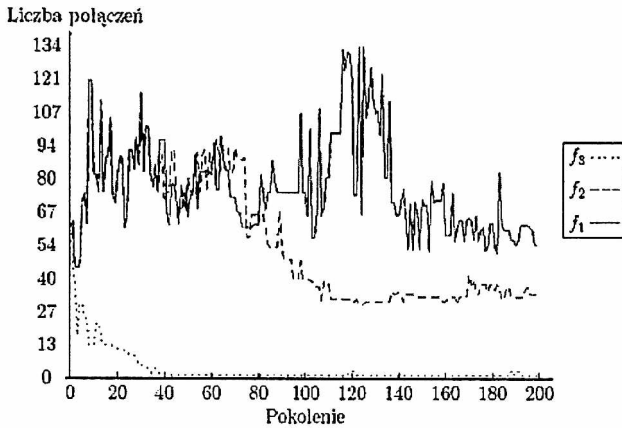


Rys. 34. Porównanie rozmiaru sieci dla trzech funkcji przystosowania dla zadania XOR

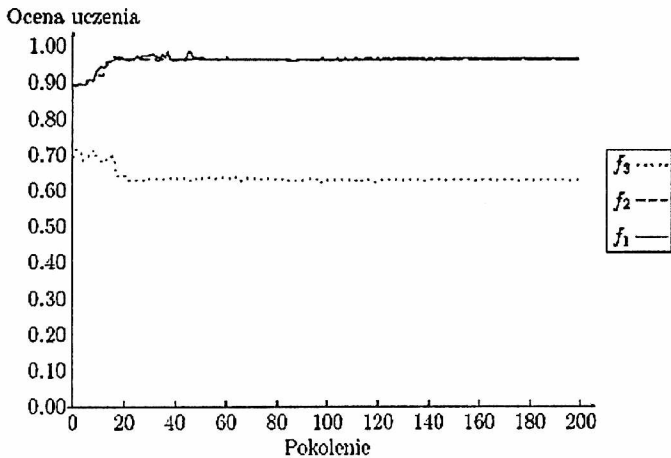
Dla stosunkowo prostych zadań, jakim jest XOR czy *Parzystość* (porównywalne wyniki), nie ma istotnej różnicy w działaniu GANN dla wszystkich funkcji f_1 , f_2 i f_3 . Na rysunku 34 pokazany jest rozmiar sieci dla zadania XOR dla trzech funkcji przystosowania. Zrezygnowano z pokazywania wykresu błędu sieci, ponieważ sieć nauczyła się dokładnie zadania przy każdej z tych funkcji ocenających. W ogólności jednak widoczny jest wyraźny wpływ stosowanej funkcji oceny.

Funkcja f_3 najczęściej prowadzi do ewolucji niezbyt dużych sieci, ale też dających największe błędy. Widoczne jest to przy pozostałych problemach – ZOO, Iris, Housing. Przykładowe wyniki pokazane są na wykresach (rys. 35–42).

Dla danych ZOO, przy funkcjach f_1 i f_2 , GANN przez około 30 pokoleń ewoluował stosunkowo duże sieci neuronowe, ale coraz lepiej uczące się. Po tym czasie sieci zdawały już test uczenia, rozpoczął się proces zmniejszania rozmiaru sieci. Stosując funkcję f_3 , system nie był w stanie wyewoluować poprawnie działającej sieci neuronowej. Widać to wyraźnie na rysunku 36, gdzie ocena uczenia pozostaje na stałym poziomie.



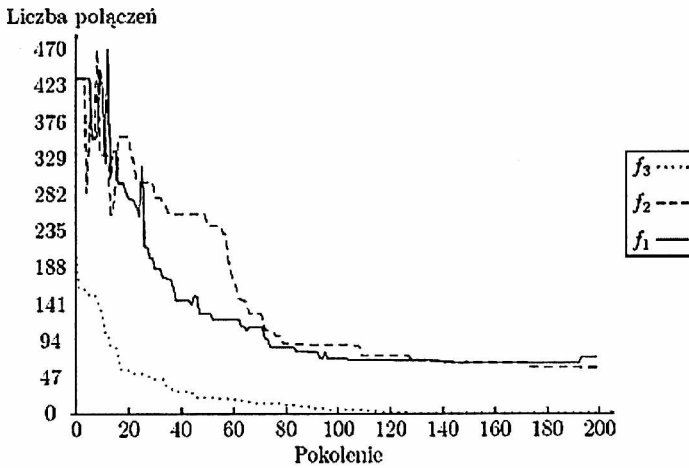
Rys. 35. Porównanie rozmiaru sieci dla trzech funkcji przystosowania, dla zadania ZOO



Rys. 36. Porównanie jakości uczenia sieci dla trzech funkcji przystosowania, dla zadania ZOO

Podobne wyniki uzyskuje się dla problemu IRIS. Funkcja f_3 powoduje powstawanie zbyt małych sieci, aby mogły się dobrze nauczyć danego im zadania. Charakterystyczne są tu wahania rozmiaru sieci przy stosowaniu funkcji f_1 . Wydaje się, że ewolucja nie jest tu w stanie przejść przez etap odnajdowania rozwiązania spełniającego *test uczenia*. Nawet, gdy najlepsze sieci z populacji spełniają ten test, to dzieje się tak tylko dla pojedynczych pokoleń. Zjawisko to należy tłumaczyć wpływem czynnika losowego wprowadzanego przez metody uczenia.

Uzyskane architektury nie są więc dobrymi architektuрами, gdyż ich uczenie daje w większości przypadków sieci niespełniające założonych kryteriów jakości udzielanych odpowiedzi. Poszukiwane, poprawne sieci dawałyby odpowiedzi umożliwiające zdanie testu uczenia w większości przypadków. Nie dochodzi więc do stabilnego uruchomienia kryterium ewolucji związanego z minimalizacją, co jest przyczyną widocznych na wykresie oscylacji rozmiaru najlepszego osobnika.

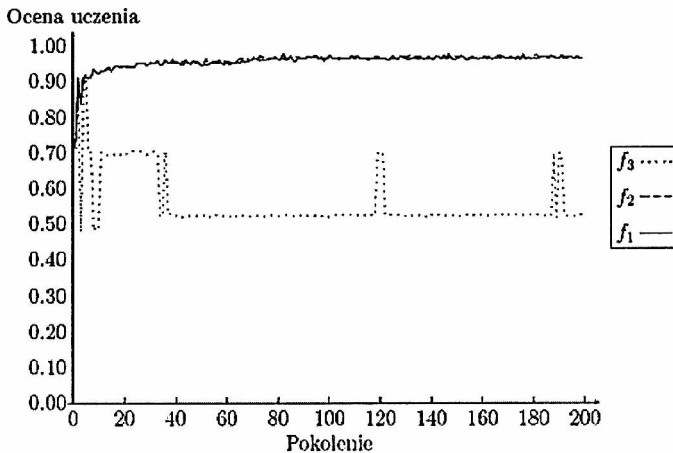


Rys. 37. Porównanie rozmiaru sieci dla trzech funkcji przystosowania, dla zadania IRIS

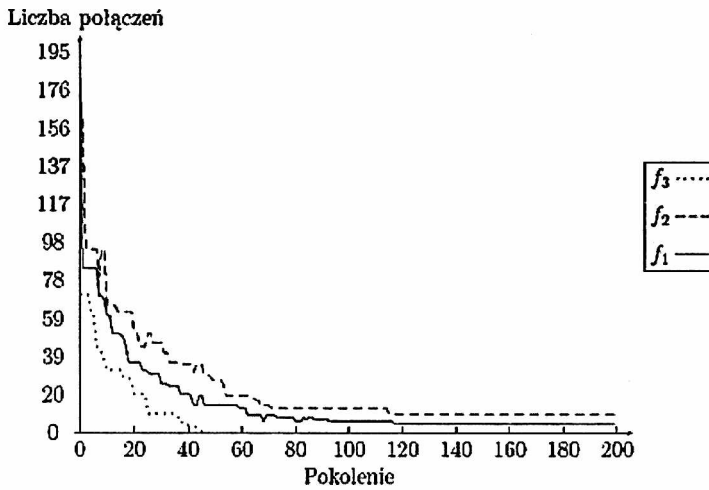
Ewolucja z użyciem funkcji f_2 zachowuje się bardzo podobnie dla początkowych pokoleń, następnie, po uzyskaniu rozwiązań spełniających *test uczenia*, włączone zostaje kryterium odpowiedzialne za minimalizację rozmiaru sieci. Proces ten jest widoczny na wykresie przedstawionym na rysunku 37 i zachodzi w okolicach 70. pokolenia. Prowadzi to do znacznego zmniejszenia rozmiaru sieci neuronowych w porównaniu z sieciami uzyskiwanymi za pomocą funkcji oceny f_1 .

Należy tu podkreślić, iż w części przeprowadzonych badań dla tego problemu uzyskano odwrotne rezultaty – ewolucja z funkcją f_1 przebiegała bardzo efektywnie, natomiast ewolucja z funkcją f_2 utykała w minimum lokalnym.

Podsumowując, efektywność funkcji f_1 i f_2 jest do siebie zbliżona, a przedstawiony przypadek był tylko jednym z kilku uzyskanych scenariuszy. Ilustruje on jednak dość dobrze zachowania badanych funkcji celu f_1 oraz f_2 .

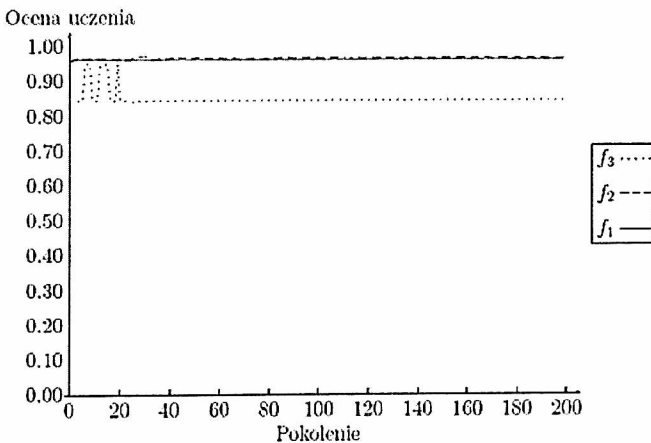


Rys. 38. Porównanie jakości uczenia sieci dla trzech funkcji przystosowania, dla zadania IRIS



Rys. 39. Porównanie rozmiaru sieci dla trzech funkcji przystosowania, dla zadania HOUSING

Wykresy przedstawiające porównanie efektywności przebadanych funkcji oceny dla problemu HOUSING zamieszczone są na rysunku 39 i 40. Ewolucja przeprowadzana dla problemu HOUSING z użyciem funkcji f_1 oraz f_2 nie napotykała problemów, które wystąpiły w przypadku problemu IRIS. Sieci neuronowe, które zdawały test uczenia zostały uzyskane już na samym początku procesu ewolucji, po czym następowała szybka minimalizacja ich rozmiaru. Ewolucja z użyciem funkcji oceny f_3 również dla tego problemu nie dała zadowalających rezultatów. Przyczyną takiego zachowania, podobnie jak dla problemów ZOO oraz IRIS, jest duża liczba przykładów uczących. Brak poprawnych odpowiedzi na część pytań ze zbioru testowego powoduje tylko nieznaczne pogorszenie końcowej oceny sieci. Nie powstaje więc silna presja selekcyjna, wymuszająca przede wszystkim jak najlepsze wyuczenie się przedstawionego problemu.



Rys. 40. Porównanie jakości uczenia sieci dla trzech funkcji przystosowania, dla zadania HOUSING

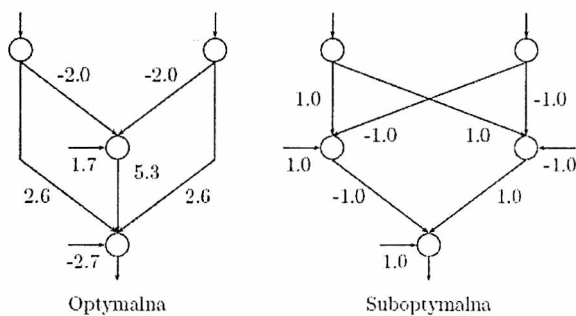
8.2.2. WPŁYW SPOSOBU KODOWANIA NA EFEKTYWNOŚĆ EWOLUCJI SIECI NEURONOWYCH

Wspominaliśmy już, że sposób kodowania powinien mieć duży wpływ na:

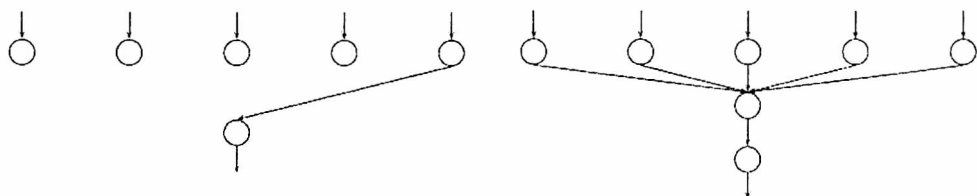
- jakość generowanych rozwiązań z uwagi na możliwość reprezentacji ograniczonej klasy sieci neuronowych,
- szybkość znajdowania rozwiązania z uwagi na różny rozmiar przestrzeni rozwiązań.

Spróbujmy się przyjrzeć temu problemowi na wybranych przykładach. Weźmy pod uwagę rozmiar generowanych sieci neuronowych. Zaczniemy od stosunkowo prostego dla sieci neuronowych problemu XOR. Przykładowe sieci neuronowe, rozwiązujące problem XOR, pokazane są na rysunku 41. Najmniejsza sieć neuronowa (biorąc pod uwagę liczbę połączeń) nie mogła być znaleziona podczas stosowania kodowania pośredniego, ponieważ nie jest ona reprezentowalna w tej przestrzeni rozwiązań.

Podobna sytuacja występuje dla zadania PARZYSTOŚĆ. Wiadomo, że o parzystości liczby decyduje ostatnia cyfra, nie ma więc potrzeby sprawdzać pozostałych cyfr (rys. 42). Jednakże stosując zaproponowany sposób kodowania pośredniego, nie jesteśmy w stanie reprezentować optymalnej sieci neuronowej z uwagi na rozmiar. Stosując kodowanie bezpośrednie, możemy otrzymać sieć neuronową tylko z jednym połączeniem – najmniej znacząca cyfra podawana jest na wejście sieci neuronowej, czyli element decydujący o parzystości liczby.

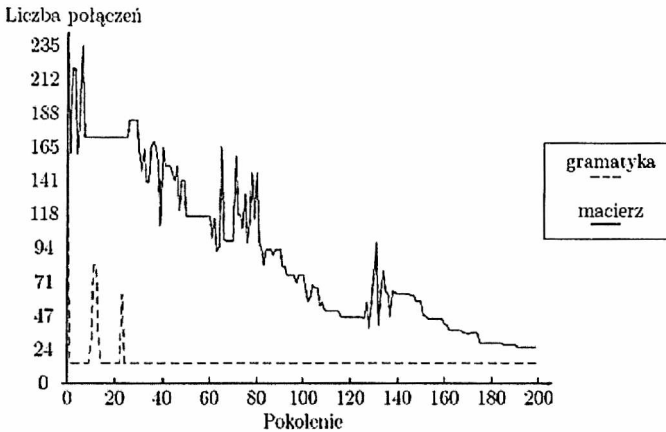


Rys. 41. Sieci neuronowe generowane przez GANN dla zadania XOR przy stosowaniu kodowania bezpośredniego (optymalna) i pośredniego (suboptymalna)



Rys. 42. Sieci neuronowe generowane przez GANN dla zadania PARZYSTOŚĆ przy stosowaniu kodowania bezpośredniego (optymalna) i pośredniego (suboptymalna)

Ewolucja sieci neuronowych kodowanych w sposób pośredni (w postaci gramatyk) powoduje zwykle generowanie mniejszych sieci neuronowych niż przy stosowaniu kodowania macierzowego. Ilustruje to rysunek 43. Porównując liczbę połączeń i ocenę uczenia ewoluowanej sieci neuronowej (rys. 44) widać, że przy niewiele gorszym uczeniu kodowanie pośrednie znacznie szybciej znajduje małe sieci. Jeśli „optymalna” dla danego zadania sieć neuronowa jest obecna w przestrzeni rozwiązań zdefiniowanej przez gramatykę, to znalezienie tej sieci jest stosunkowo łatwe dla procesu ewolucji. Kodowanie bezpośrednie (za pomocą macierzy) pozwala reprezentować znacznie więcej sieci neuronowych, co wymusza na ewolucji przeszukiwanie o wiele większej przestrzeni potencjalnych rozwiązań. Przestrzeń ta jest bardzo zwodnicza, sieci o podobnej architekturze mogą dawać całkiem różne odpowiedzi na te same przykłady, sieci o całkowicie różnych architekturach mogą produkować podobne wyniki; przeszukiwanie takich przestrzeni jest zadaniem trudnym dla algorytmu ewolucyjnego. Dlatego czasami algorytm może utknąć w lokalnym optimum i nie znaleźć prawidłowego rozwiązania.

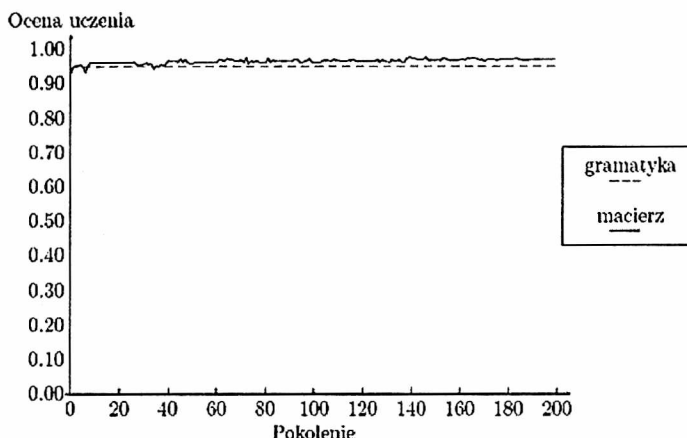


Rys. 43. Rozmiar i ocena uczenia sieci neuronowej generowanej przez GANN dla zadania IRIS przy stosowaniu kodowania bezpośredniego (macierz) i pośredniego (gramatyka)

Warto poświęcić trochę miejsca uczeniu tradycyjnemu i genetycznemu. Tendencja do utkania w minimach lokalnych, cechująca metodę uczenia ze wsteczną propagacją błędów, czyni tę metodę w jej najprostszej postaci niezbyt efektywną podczas procesu ewolucji. Wpływ losowo wybieranego punktu startowego jest zbyt duży, aby można było uzyskać efektywną metodę oceniania sieci. Rozwiązaniem tego problemu jest powtarzanie procesu uczenia wielokrotnie dla różnych punktów startowych.

Algorytm genetyczny zastosowany jako metoda uczenia znacznie lepiej unika utkania w minimach lokalnych. Ewolucja oparta na dużych populacjach bardzo efektywnie przeszukuje przestrzeń rozwiązań i znajduje osobniki będące dobrze wyuczonymi sieciami. Największą wadą tej metody jest jej bardzo duża złożoność czasowa.

Wróćmy do rysunku 41, gdzie mamy sieci neuronowe wyewoluowane dla zadania XOR. Na tym rysunku podane są wagi obydwóch sieci neuronowych, umieszczono je nie bez powodu. Sieć podpisana jako „Optymalna” zawiera mniej neuronów i mniej połączeń.



Rys. 44. Rozmiar i ocena uczenia sieci neuronowej generowanej przez GANN dla zadania IRIS przy stosowaniu kodowania bezpośredniego (macierz) i pośredniego (gramatyka)

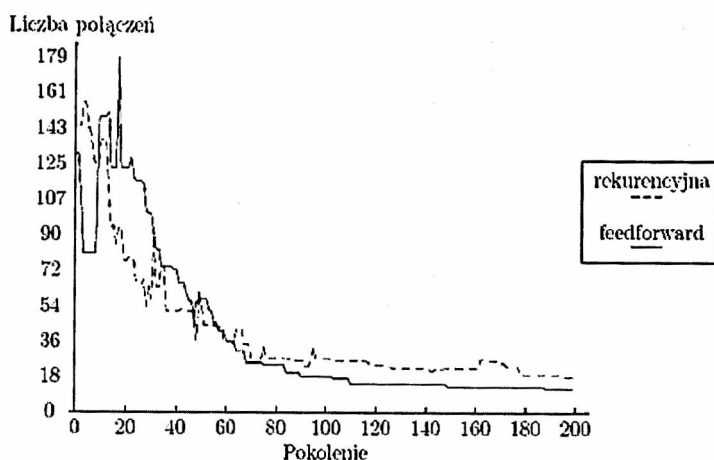
W podręcznikach dotyczących sieci neuronowych, jako rozwiązanie zadania XOR podaje się najczęściej sieć warstwową o dwóch neuronach w warstwie ukrytej, jak ta po prawej stronie rysunku 41. Wyewoluowana sieć neuronowa, zawierająca, poza neuronami wejściowymi i wyjściowym tylko jeden neuron, była zaskoczeniem. Warto podkreślić, że optymalna sieć została otrzymana przy zastosowaniu kodowania bezpośredniego i uczenia genetycznego. Ręczne sprawdzenie pozwala stwierdzić, że ta sieć działa poprawnie. Wykonano próbę nauczania zadania XOR sieci o takiej samej architekturze, stosując metodę wstecznej propagacji błędów. Niestety próba się nie powiodła. Wykonano wiele prób, startując za każdym razem z losowych wag, ale żadna z nich nie zakończyła się powodzeniem. Zmieniono przedział losowania wag tak, aby znane wartości prawidłowych wag mieściły się w tym przedziale. Ale w takim przypadku również nie osiągnięto ani razu powodzenia, mimo wielokrotnego powtarzania eksperymentu. Uruchomienie metody wstecznej propagacji błędów dla początkowego zestawu wag, równych tym z rysunku 41 powodowało, że algorytm nie zmieniał wag, uznając je za optymalne. Prawdopodobnie w otoczeniu tych wartości wag jakość działania sieci gwałtownie się zmienia – optymalizowana funkcja jest bardzo zwodnicza, co powoduje, że algorytm gradientowy, jakim jest metoda wstecznej propagacji błędów, idzie w złym kierunku i nie jest w stanie znaleźć właściwego rozwiązania.

Podany przykład wskazuje jednoznacznie na dużą zaletę uczenia genetycznego – zdolność do ucieczki z pułapki ewolucyjnej. Należy pamiętać jednak, że uczenie genetyczne jest procesem czasochłonnym.

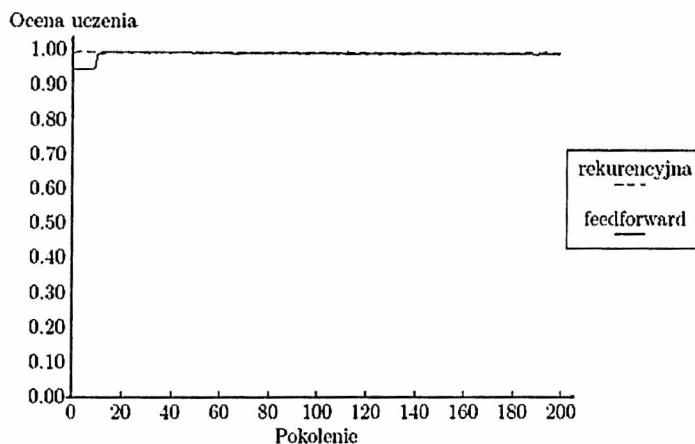
8.2.3. EWOLUCJA JEDNOKIERUNKOWYCH I REKURENCYJNYCH SIECI Z DYSKRETNYM CZASEM, DLA ZADANIA GENEROWANIA SEKWENCJI

W tej części eksperymentów interesowało nas przede wszystkim, czy algorytm ewolucyjny da sobie radę z wyewoluowaniem sieci rekurencyjnej, zdolnej do generowania se-

kwencji o zadanej długości. Ponieważ odpowiednią sieć rekurencyjną można rozwinąć do postaci jednokierunkowej sieci o zwielokrotnionej liczbie wyjść, równej liczbie cykli, więc sprawdzimy, czy system jednakowo dobrze ewoluuje oba typy sieci.



Rys. 45. Rozmiar i ocena uczenia sieci neuronowej generowanej dla zadania generowania sekwencji dla krótkiego ciągu, sieć rekurencyjna i jednokierunkowa

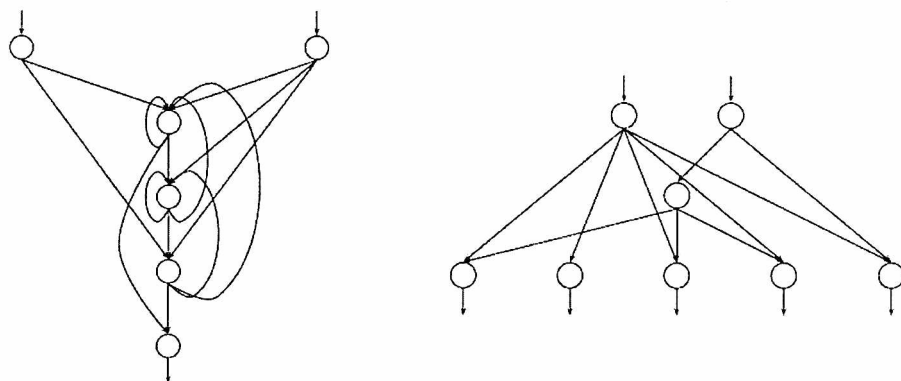


Rys. 46. Rozmiar sieci neuronowej generowanej dla zadania generowania sekwencji dla krótkiego ciągu, sieć rekurencyjna i jednokierunkowa

Krótki ciąg arytmetyczny – 5 elementów danych wyrażeniem (22), nie stanowił trudnego zadania dla systemu GANN. System generował sieci rekurencyjne o małych rozmiarach, dające zbliżone odpowiedzi do oryginalnych wartości ciągu (patrz tab. 6). Generowane sieci jednokierunkowe zazwyczaj były nieco mniejsze, tzn. miały mniej połączeń (rys. 45–47). Dla obydwóch rodzajów sieci, stosunkowo szybko dochodziło do stabilizowania się zarówno rozmiaru sieci, jak i produkowanego błędu.

Tabela 6. Sekwencja generowana przez rekurencyjną sieć neuronową (NN rek.), sieć jednokierunkową (NN BP) i wartości prawidłowe wg założonego ciągu – ciąg krótki, 5 elementów

Wartości parametrów	a_0	a_1	a_2	a_3	a_4
$x = 0, y = 1$ (NN rek.)	0,73	1,14	1,75	3,01	4,45
$x = 0, y = 1$ (NN BP)	0,98	1,78	2,54	3,38	3,85
$x = 0, y = 1$ (ciąg)	1	2	3	4	5
$x = 20, y = -1$ (NN rek.)	19,41	19,03	18,33	17,00	15,69
$x = 20, y = -1$ (NN BP)	19,09	18,38	17,64	16,89	16,13
$x = 20, y = -1$ (ciąg)	19	18	17	16	15
$x = 5, y = 1$ (NN rek.)	5,07	6,17	6,78	9,23	10,28
$x = 5, y = 1$ (NN BP)	3,85	4,86	6,74	8,09	8,87
$x = 5, y = 1$ (ciąg)	6	7	8	9	10
$x = 10, y = 1$ (NN rek.)	10,90	12	12,78	13,70	14,21
$x = 10, y = 1$ (NN BP)	10,46	10,26	12,79	13,87	14,53
$x = 10, y = 1$ (ciąg)	11	12	13	14	15



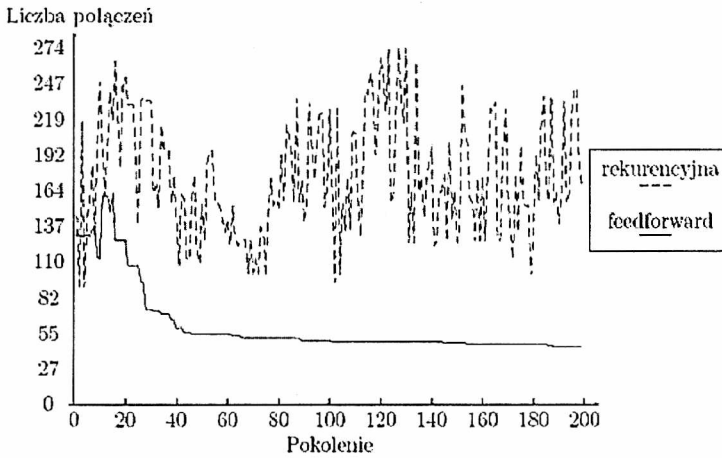
Rys. 47. Przykładowe sieci neuronowe, generowane przez GANN dla zadania generowania sekwencji dla krótkiego ciągu, sieć rekurencyjna i jednokierunkowa

Tabela 7. Sekwencja generowana przez rekurencyjną sieć neuronową (NN rek.), sieć jednokierunkową (NN BP) i wartości prawidłowe według założonego ciągu – 10 pierwszych elementów

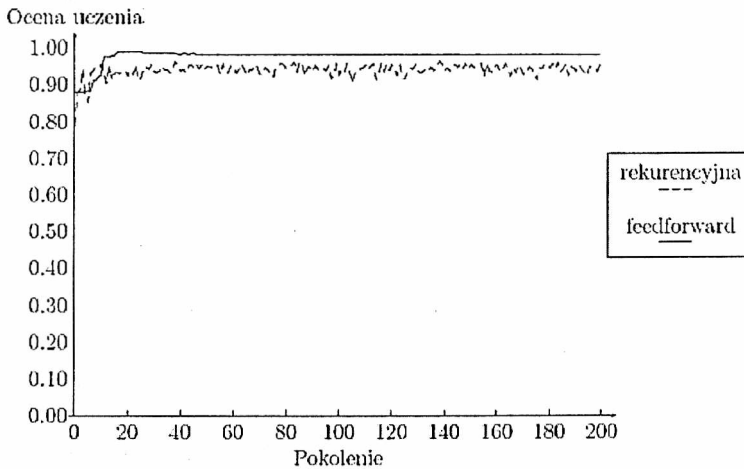
Wartości parametrów	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
$x = 0, y = 1$ (NN rek.)	0,96	1,31	3,46	3,12	4,43	5,57	6,68	7,66	8,69	9,44
$x = 0, y = 1$ (NN BP)	0,98	1,62	2,47	3,07	4,04	4,66	5,52	5,26	6,96	7,01
$xX = 0, y = 1$ (ciąg)	1	2	3	4	5	6	7	8	9	10
$x = 20, y = -1$ (NN rek.)	18,97	18,84	17,81	16,92	15,96	14,92	13,68	12,75	11,92	11,18
$x = 20, y = -1$ (NN BP)	19,13	18,39	17,58	17,10	16,27	14,86	15,33	13,73	12,93	11,50
$x = 20, y = -1$ (ciąg)	19	18	17	16	15	14	13	12	11	10
$x = 5, y = 1$ (NN rek.)	4,59	5,75	6,96	7,95	9,38	10,66	11,25	12,26	13,61	14,72
$x = 5, y = 1$ (NN BP)	3,31	5,04	6,82	7,25	9,09	9,97	10,66	11,42	12,96	14,00
$x = 5, y = 1$ (ciąg)	6	7	8	9	10	11	12	13	14	15
$x = 10, y = 1$ (NN rek.)	10,01	11,20	12,63	13,66	14,26	15,01	16,18	17,18	18,23	18,72
$x = 10, y = 1$ (NN BP)	9,91	11,27	12,64	13,78	14,63	15,29	16,32	16,79	17,79	18,29
$x = 10, y = 1$ (ciąg)	11	12	13	14	15	16	17	18	19	20

Znacznie gorzej system zachowuje się, gdy chcemy wyewoluować rekurencyjną sieć neuronową dla długiego ciągu (10 pierwszych wyrazów tego samego ciągu). Przykładowe ciągi produkowane przez jedną z wygenerowanych sieci rekurencyjnych zawiera tabela 7. Ważne jest to, że dana sieć neuronowa, o zadanej architekturze, nie zawsze jest w stanie wyuczyć się w jednakowym stopniu, czasami dopiero wielokrotne powtarzanie procesu uczenia może dać lepsze efekty. Widoczna jest silna zależność metody uczenia z propagacją wsteczną przez czas od początkowych wartości wag.

Dla zadania „długi ciąg” dużo lepsze efekty dawała ewolucja sieci jednokierunkowych (rys. 48 i 49 oraz tab. 7). Ewoluwane sieci były zminimalizowane i poprawnie realizowały zadanie.



Rys. 48. Rozmiar sieci neuronowej dla zadania generowania sekwencji dla długiego ciągu, sieć rekurencyjna i jednokierunkowa



Rys. 49. Ocena uczenia sieci neuronowej dla zadania generowania sekwencji dla długiego ciągu, sieć rekurencyjna i jednokierunkowa

8.2.4. OMÓWIENIE WYNIKÓW EKSPERYMENTÓW Z PROGRAMEM GANN

Na początek należy powiedzieć kilka zdań na temat powtarzalności wyników. W działaniu algorytmów ewolucyjnych jest to bardzo ważny czynnik. Proponując algorytm ewolucyjny jako narzędzie rozwiązujące daną klasę problemów, interesuje nas, aby rozwiązywał on nasze zadanie w sposób powtarzalny. W ramach przeprowadzonych badań, dla wszystkich konfiguracji, eksperymenty były powtarzane wielokrotnie (10–20), z wyjątkiem tych z jawnymi zależnościami czasowymi – te były powtarzane kilka razy. Wyniki z zastosowaniem poszczególnych funkcji przystosowania były dość powtarzalne. Zawsze udawało się znaleźć sieć realizującą dość dobrze dane zadanie, czasami ewoluowane sieci miały o 1 neuron lub jedno do trzech połączeń więcej.

Porównując stosowane funkcje przystosowania, można jednoznacznie stwierdzić, że f_3 nie jest dobrym rozwiązaniem. Nie wymusza ona podziału ewolucji na etapy, w których na początku położony jest nacisk na ewolucję sieci poprawnie realizujących narzucone zadanie, a dopiero w drugiej fazie minimalizowany jest rozmiar sieci, mając oczywiście na uwadze cały czas jakość jej działania. Założenie, iż najpierw należy uzyskać sieci, które można wyuczyć, a potem minimalizować te sieci w taki sposób, aby nie straciły one swoich właściwości, dało bardzo dobre rezultaty.

Przeprowadzone badania potwierdziły założenia, iż ewolucja oparta na kodowaniu bezpośrednim jest w stanie efektywnie wykorzystać bardzo dużą swobodę modelowania sieci, jaką to kodowanie zapewnia. Istotnym problemem okazał się natomiast fakt, iż znaczny odsetek sieci wygenerowanych przez kodowanie bezpośrednie nie spełnia podstawowych założeń związanych z sensownością uzyskanego rozwiązania. Dlatego, znaczna część wyewoluowanych sieci jest odrzucana jako niezdolne do rozwiązania problemu. Bezpośrednim skutkiem takiego zachowania jest wydłużenie procesu ewolucji.

Ewolucja oparta na kodowaniu pośrednim znacznie szybciej uzyskuje sieci zbliżone do optymalnych. Główną przyczyną takiego zachowania jest znacznie mniejsza przestrzeń poszukiwań niż w przypadku kodowania bezpośredniego. Otrzymywane sieci, zgodnie z tym na co pozwala kodowanie pośrednie, zawsze są sieciami poprawnymi pod względem sensowności ich budowy.

Tendencja do utykania w minimach lokalnych, cechująca metodę uczenia ze wsteczną propagacją błędów, czyni tę metodę, w jej najprostszej postaci, niezbyt efektywną podczas procesu ewolucji. Wpływ losowo wybieranego punktu startowego jest zbyt duży, aby można było uzyskać efektywną metodę oceniania sieci. Dlatego też może być opłacalne stosowanie uczenia genetycznego, mimo znacznego wydłużenia czasu uczenia.

Przeprowadzone, dość ograniczone eksperymenty wskazują, że dla problemów z jawną zależnością rekurencyjną można zbudować poprawnie działające zarówno sieci rekurencyjne, jak i sieci bez sprzężeń zwrotnych. Sieci te miały zbliżony do siebie zarówno rozmiar, jak i możliwości uogólniania. Ewolucja sieci rekurencyjnych jest zadaniem dużo trudniejszym, niż ewolucja sieci bez sprzężeń zwrotnych. Pierwszym powodem jest dużo większa przestrzeń możliwych architektur. Częściowo wiąże się to z koniecznością znalezienia w procesie ewolucji poprawnie działających sprzę-

żeń zwrotnych, gdyż tylko one mogą przyczynić się do efektywnego rozwiązywania postawionych problemów. Czynniki te w bardzo istotny sposób zwiększa rozmiar przestrzeni poszukiwań.

Wnioski wynikające z przedstawionych badań wskazują, że jeśli poszukujemy sieci o możliwie małym rozmiarze oraz dobrze rozwiązującej swoje zadanie, to warto stosować większą przestrzeń poszukiwań i uczenie mające większą zdolność do ucieczki z optimum lokalnego. Oczywiście podejście takie okupione jest wydłużeniem procesu poszukiwania zadowalającej nas sieci.

9. ALGORYTM GENETYCZNY Z EFEKTEM PLEJOTROPOWYM I POLIGENICZNYM JAKO NARZĘDZIE PROJEKTOWANIA NN

W tej części pracy omówione jest wykorzystanie w algorytmach genetycznych takiego modelu osobnika, który jest bardziej „zbliżony do natury”. Taki algorytm genetyczny będzie testowany jako narzędzie projektujące sieci neuronowe dla określonych zadań. W modelu tym rozróżniana jest budowa wewnętrzna osobnika od jego zewnętrznej funkcjonalności. „Przejście” z wewnętrznej budowy osobnika na jego zewnętrzną funkcjonalność jest determinowane przez efekt plejotropowości i poligeniczności. Efekt ten może być traktowany jako operator genetyczny, mówiący nam o wpływie jednej cechy wewnętrznej na wiele cech zewnętrznych oraz zależności jednej cechy zewnętrznej od wielu cech wewnętrznych osobnika. Zależności determinowane przez wymieniony efekt powinny w trakcie działania algorytmu dynamicznie zmieniać się, usiłując przyspieszyć działanie ewolucji. Tak zmodyfikowany algorytm genetyczny mógłby okazać się czasami lepszy od klasycznego algorytmu genetycznego (pod względem liczby pokoleń potrzebnych do znalezienia rozwiązania). Dodanie dodatkowego elementu (efektu plejotropowości i poligeniczności) do działania algorytmu zwiększa jednak kosztowność czasową wykonywanych przez niego obliczeń.

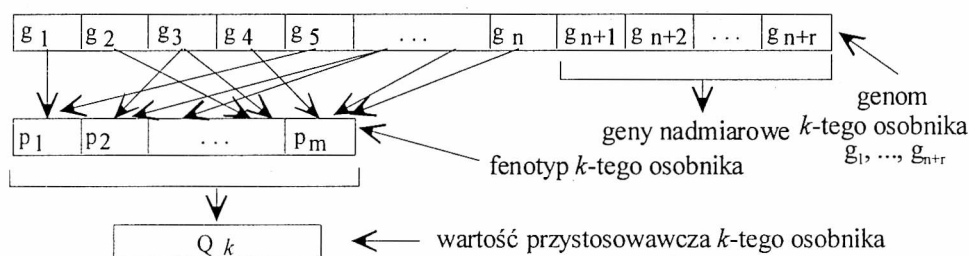
W związku z użyciem w algorytmie genetycznym, związanym z sieciami neuronowymi, efektu plejotropowości i poligeniczności nasuwają się następujące pytania:

- Czy algorytm wykorzystujący efekt plejotropowości i poligeniczności może być ciekawą alternatywą dla klasycznego algorytmu stosowanego w sieciach neuronowych – jak wygląda porównanie ich działania?
- Jaki wpływ na otrzymywane rezultaty ma wykorzystywany efekt plejotropowości i poligeniczności?
- Czy istnieje szansa wykorzystania algorytmu do rozwiązywania skomplikowanych problemów?

Praca jest próbą odpowiedzi na postawione pytania. Aby przedstawić zastosowanie efektów plejotropowych i poligenicznych do projektowania sieci neuronowych, przybliżymy wcześniej algorytm genetyczny z tymi efektami. Jego dokładny opis i właściwości są w pracy (Kwaśnicka 1999).

9.1. K-MODEL A PROJEKTOWANIE SIECI NEURONOWYCH

K-Model (Kwaśnicka 1999), poza innym sposobem dekodowania chromosomu do fenotypu, ma wprowadzone geny nadmiarowe (część genów, mimo że biorą one udział w samym procesie ewolucji, nie są genami fenotypowymi, są one pewnym „balastem”, który przenosi jednak jakąś informację – są to geny nadmiarowe, nieaktywne). Uwzględnia się też w działaniu możliwość wystąpienia operatora makromutacji. Osobnik w tym modelu wygląda jak na rysunku 50.



Rys. 50. Reprezentacja osobnika w K-Modelu

Jak już zostało wspomniane, w osobniku tym jeden gen determinuje wiele fenów – takie zachowanie nazywa się efektem **plejotropowym**. Zjawisko odwrotne, tzn. określanie fenu przez więcej niż jeden gen, nazywa się efektem **poligenicznym**. Jedną z pierwszych prób wykorzystania tych efektów została przeprowadzona przez Altenberga (Altenberg 1994).

Ciekawe efekty może dać odpowiednie zastosowanie efektu plejotropowości i poligeniczności do projektowania sieci neuronowych. Plejotropowość i poligeniczność (nazywana w dalszej części jako efekt **PP**) to element algorytmów genetycznych występujący w odwzorowaniu genotypu na fenotyp (Kwaśnicka 1999). Można zauważyć, że efekt ten może być traktowany jako swoisty sposób kodowania struktury sieci za pomocą gramatyki. W „zwykłym” kodowaniu gramatycznym mamy przejście ze słowa kodowego (słowa w rozumieniu tej gramatyki) na kodowanie bezpośrednie sieci. Kodowanie bezpośrednie pozwala nam dopiero daną sieć przetestować. Zazwyczaj słowo kodowania gramatycznego jest krótsze od słowa kodowania bezpośredniego – jeden symbol kodowy gramatyki jest najczęściej rozwijany na wiele symboli kodowania bezpośredniego.

Zauważmy, że przy efekcie plejotropowości i poligeniczności również mamy podobne przejście – co prawda, jest to przejście z genotypu na fenotyp, ale fenotyp ma być przecież podstawą budowy sieci, może on być więc słowem kodowania bezpośredniego. Wydaje się, że efekt plejotropowości i poligeniczności może być ciekawym sposobem kodowania, ma pewną zaletę, mianowicie: podobnie jak za pomocą gramatyki pozwala, aby chromosom, na którym działa algorytm genetyczny, był krótki (krótszy niż fenotyp). Dodatkowo, może to być gramatyka, która może się zmieniać w trakcie działania algorytmu genetycznego (jeżeli dopuścimy możliwość modyfikacji

tego efektu). Ponieważ można przyjąć, że efekt plejotropowości i poligeniczności zapewnia nam już jakieś złożone kodowanie, wydaje się niecelowe stosowanie innego kodowania fenotypu niż bezpośrednie we właściwym algorytmie genetycznym projektującym sieć neuronową.

9.2. PROPOZYCJA ROZWIĄZANIA

Propozycja rozwiązania obejmuje wprowadzenie podstawowych idei K-Modelu do algorytmu genetycznego projektującego (uczącego) sieć neuronową oraz pewnych modyfikacji do działania samego algorytmu, dotyczących wykonywania dodatkowych operacji.

9.2.1. STRUKTURY GENETYCZNE

Wszystkie podstawowe struktury genetyczne (genotyp, fenotyp) są traktowane jako wektory (tablice jednowymiarowe). Elementami tych tablic mogą być liczby całkowite bądź rzeczywiste.

Genotyp

Geny nie są bitami, lecz liczbami składającymi się z kilkadziesiąt bitów (32 bity, 64 bity) – mogą reprezentować zarówno liczby całkowite, jak i rzeczywiste. Przyjęto, że będą to liczby rzeczywiste.

g_1	g_2	...	g_{N-1}	g_N
-------	-------	-----	-----------	-------

Rys. 51. Genotyp

gdzie: g_i – gen o numerze i , N – rozmiar genotypu.

Z takim przyjęciem reprezentacji kodowania genu wiąże się zmiana funkcjonalności niektórych operatorów genetycznych (działających na genomie) w porównaniu z klasycznym algorytmem genetycznym, w którym osobniki są ciągiem bitów.

Operator mutacji

W klasycznym algorytmie genetycznym operator ten działa na pojedynczych bitach, z określonym prawdopodobieństwem, zmieniając stan bitu na przeciwny. Teraz mamy do czynienia z liczbami rzeczywistymi, uzasadniona wydaje się taka modyfikacja, w której z zadaniem prawdopodobieństwem modyfikujemy gen o pewną wartość, która też może się zmieniać w zależności od jakiegoś parametru.

Operator krzyżowania

Krzyżowanie w klasycznym algorytmie genetycznym wykonywane jest na ciągu bitowym (Michalewicz 1996) rozwiązanie takie można było tu pozostawić. Jednakże bardziej naturalne wydaje się traktowanie genu jako pewnej niepodzielnej całości (całej liczby rzeczywistej), dlatego wprowadzono związane z tym ograniczenie: punkt podziału musi znajdować się zawsze między ciągami bitów będącymi reprezentacją genów. Inaczej mówiąc, całe wartości genów (liczby rzeczywiste) mogą być wymieniane między rodzicami.

Fenotyp

Struktura fenotypu jest analogiczna do struktury genotypu. Tutaj również stosuje się liczby rzeczywiste.

f_1	f_2	...	f_{M-1}	f_M
-------	-------	-----	-----------	-------

Rys. 52. Fenotyp

gdzie: f_i – fen o numerze i , M – rozmiar fenotypu.

Zazwyczaj przyjęcie konkretnego typu i rozmiaru genotypu i fenotypu wymusza pośrednio budowę stosowanej funkcji przejścia między genotypem a fenotypem (np. efekt plejotropowości i poligeniczności – patrz dalej). Funkcja przejścia może zmniejszać lub zwiększać rozmiar fenotypu w porównaniu z genotypem oraz zmieniać typ podstawowej jednostki (np. może „przejsć” z liczb całkowitych na rzeczywiste bądź binarne) – tu jednak takiej sytuacji nie będzie.

Efekt plejotropowości i poligeniczności

Efekt ten, z programistycznego punktu widzenia, można traktować jako przekształcenie liniowe wektora genotypu na wektor fenotypu. Przy takim określeniu genotypu i fenotypu, jak podane zostało wyżej, przekształcenie to może być macierzą ([PP]) spełniającą równanie:

$$[Genotyp]_{I \times N} \times [PP]_{N \times M} = [Fenotyp]_{I \times M} \quad (23)$$

Macierz **PP** (efekt plejotropowości i poligeniczności) może przyjmować dowolne wartości – jednak od tego, jaki typ tych wartości zostanie określony, zależec będzie typ wartości występujących w fenotypie.

3	-5	7	×	1	0	1	0	=	3	2	10	-5
Genotyp									Fenotyp			

Rys. 53. Przykład zastosowania macierzy plejotropowości i poligeniczności

Typ genu i fenu został określony jako rzeczywisty – typ elementu macierzy plejotropowości i poligeniczności może być dowolny – binarny, całkowity, rzeczywisty. Każdy z tych typów ma pewne zalety i wady. Obliczenia dla typu binarnego są najszybsze, najłatwiej jest macierz z takimi elementami modyfikować (np. klasyczna mutacja). Obliczenia dla typu rzeczywistego są najwolniejsze, ale tworzony w rezultacie wektor, może przyjmować wszystkie potencjalne wartości. Z właściwości iloczynu wektorowego wynika, że jeśli wszystkie elementy w kolumnie macierzy **PP** będą miały wartość zerową, to odpowiedni element (cecha) fenotypu również będzie wynosił zero. Biorąc pod uwagę wady i zalety rozważanych postaci macierzy **PP**, zdecydowano się na zero-jedynkową macierz **PP**.

9.2.2. KODOWANIE SIECI

Założono, że projektowana sieć ma narzucone pewne ograniczenia, mianowicie:

- jest to sieć jednokierunkowa, z istniejącymi wszystkimi możliwymi połączeniami w takiej sieci,
- w sieci występuje neuron obciążający (**bias**),
- znamy liczbę neuronów w warstwie wejściowej (**noIn**),
- znamy liczbę neuronów w warstwie wyjściowej (**noOu**),
- sieć ma ustaloną funkcję aktywacji (przyjeliśmy, że będzie to funkcja **sigmoidealna**),
- wartości na wyjściach neuronów przyjmują wartości rzeczywiste ze zbioru $[0, 1]$ (wynika z przyjętej funkcji aktywacji).

Przyjęte kodowanie jest pewną modyfikacją kodowania bezpośredniego. Słowo kodowe jest wektorem, w którym są zapisane liniowo wszystkie istniejące w sieci wagi połączeń. Dla każdej zakodowanej sieci zakłada się znajomość jej liczby neuronów w warstwie ukrytej (**noHi**). Schemat słowa kodowego jest następujący:

$Wagi(b, h_1)$	$Wagi(b, ou)$	$Wagi(in, h_1)$	$Wagi(in, ou)$	$Wagi(h_1, ou)$	$Wagi(h_1, h_1)$
----------------	---------------	-----------------	----------------	-----------------	------------------

Gdzie $Wagi(b, h_1)$ to wagi połączeń od jednostki obciążającej do jednostek ukrytych, o rozmiarze **noHi** i budowie:

$Waga(b, h_1)$	$Waga(b, h_2)$	...	$Waga(b, h_{noHi})$
----------------	----------------	-----	---------------------

$Wagi(b, ou)$ – to wagi połączeń od jednostki obciążającej do jednostek wyjściowych, o rozmiarze **noOu** i budowie:

$Waga(b, ou_1)$	$Waga(b, ou_2)$	...	$Waga(b, ou_{noOu})$
-----------------	-----------------	-----	----------------------

$Wagi(in, h_1)$ to wagi połączeń od jednostek wejściowych do jednostek ukrytych, o rozmiarze **noIn-noHi** i budowie:

$Waga(in_1, hi_1)$	$Waga(in_2, hi_1)$	...	$Waga(in_{noIn}, hi_1)$
$Waga(in_1, hi_2)$	$Waga(in_2, hi_2)$	...	$Waga(in_{noIn}, hi_2)$
...	...	...	...
$Waga(in_1, hi_{noHi})$	$Waga(in_2, hi_{noHi})$	...	$Waga(in_{noIn}, hi_{noHi})$

$Wagi(in, ou)$ to wagi połączeń od jednostek wejściowych do jednostek wyjściowych, o rozmiarze **noIn-noOu** i budowie:

$Waga(in_1, ou_1)$	$Waga(in_2, ou_1)$	...	$Waga(in_{noIn}, ou_1)$
$Waga(in_1, ou_2)$	$Waga(in_2, ou_2)$	...	$Waga(in_{noIn}, ou_2)$
...	...	...	...
$Waga(in_1, ou_{noOu})$	$Waga(in_2, ou_{noOu})$	...	$Waga(in_{noIn}, ou_{noOu})$

$Wagi(hi, ou)$ to wagi połączeń od jednostek ukrytych do jednostek wyjściowych, o rozmiarze **noHi-noOu** i budowie:

$Waga(hi_1, ou_1)$	$Waga(hi_2, ou_1)$	...	$Waga(hi_{noHi}, ou_1)$
$Waga(hi_1, ou_2)$	$Waga(hi_2, ou_2)$	...	$Waga(hi_{noHi}, ou_2)$
...	...	...	...
$Waga(hi_1, ou_{noOu})$	$Waga(hi_2, ou_{noOu})$	...	$Waga(hi_{noHi}, ou_{noOu})$

$Wagi(hi, hi)$ to wagi połączeń od jednostek ukrytych do jednostek ukrytych, o rozmiarze $noHi \cdot (noHi - 1) / 2$ i budowie:

$Waga(hi_1, hi_2)$			
$Waga(hi_1, hi_3)$	$Waga(hi_2, hi_3)$		
...	...	...	
$Waga(hi_1, hi_{noHi})$	$Waga(hi_2, hi_{noHi})$	...	$Waga(hi_{noHi-1}, hi_{noHi})$

Zapis $Waga(a_i, b_j)$ oznacza wagę połączenia wychodzącego z neuronu i warstwy a , dochodzącego do neuronu j warstwy b .

Sumaryczna liczba wag w całej sieci wynosi:

$$\begin{aligned}
 \text{Rozmiar}(Wagi) &= \text{Rozmiar}(Wagi(b, hi)) + \text{Rozmiar}(Wagi(b, ou)) \\
 &\quad + \text{Rozmiar}(Wagi(in, hi)) + \text{Rozmiar}(Wagi(in, ou)) \\
 &\quad + \text{Rozmiar}(Wagi(hi, ou)) + \text{Rozmiar}(Wagi(hi, hi)) \\
 &= noHi + noOu + noIn \cdot noHi + noIn \cdot noOu + noHi \cdot noOu \\
 &\quad + \frac{noHi \cdot (noHi - 1)}{2}
 \end{aligned} \tag{24}$$

Liczba ta przy używaniu efektu plejotropowości i poligeniczności determinuje jeden z rozmiarów macierzy plejotropowości i poligeniczności.

9.2.3. OBLICZANIE PRZYSTOSOWANIA OSOBNIKA

Obliczenie przystosowania danego osobnika wymaga zademonstrowania wszystkich wzorców uczących sieci, zbudowanej na jego podstawie. Wzorce są podawane sieci w „trybie wsadowym”, zatem kolejność ich podawania nie ma znaczenia. Przyjęto, że na przystosowanie osobnika wpływa tylko błąd działania sieci

$$wzorzec_{OU} = [w_1 \quad w_2 \quad \dots \quad w_{N-1} \quad w_N] \quad (25)$$

gdzie: $wzorzec_{OU}$ – wzorec wyjściowy, w_i – i -ta jednostka wzorca wyjściowego ($w_i \in \{0,1\}$).

Każdy neuron wyjściowy może przyjmować wartości ze zbioru $[0, 1]$, jest ich tyle samo ile jednostek ma wzorec wyjściowy

$$sieć_{OU} = [ou_1 \quad ou_2 \quad \dots \quad ou_{N-1} \quad ou_N] \quad (26)$$

gdzie: $sieć_{OU}$ – wyjście sieci, ou_i – i -ty neuron wyjściowy ($ou_i \in [0, 1]$).

Dla każdej takiej pary (wzorca wyjściowego i wyjścia sieci) liczony jest błąd:

$$\delta_k = \frac{\sum_{i=1}^N (ou_i - w_i)^2}{N} \quad (27)$$

gdzie: δ_k – błąd dla k -tego wzorca.

Przystosowanie osobnika jest liczone na podstawie wszystkich podawanych wzorców uczących na podstawie wzoru:

$$\Delta = \frac{\sum_{k=1}^L (1 - \delta_k)^2}{L} \quad (28)$$

gdzie: δ_k – błąd dla k -tego wzorca, L – liczba wzorców uczących, Δ – przystosowanie osobnika.

Algorytm genetyczny ma maksymalizować tę wartość – co jest jednoznaczne z minimalizacją cząstkowych (dla każdego wzorca uczącego) błędów sieci. Należy zauważyć, że przystosowanie jest wielkością rzeczywistą z przedziału $[0, 1]$.

Jeżeli przystosowanie osiągnie określoną wartość, można przyjąć, że rozwiązanie zostało znalezione. W badaniach prowadzonych w tej pracy dano możliwość użycia innego określenia nauczania sieci (warunku końca nauczania).

Jeżeli mamy wzorec uczący i wyjście sieci (zdefiniowane jak w (25) i w (26)), to można wprowadzić inną miarę oceny przystosowania:

$$\varphi_k = \sum_{i=1}^N \xi (ou_i - w_i) \quad (29)$$

gdzie:

$$\xi(x) = \begin{cases} 0, & \text{gdy } |x| < 0,5 \\ 1, & \text{gdy } |x| \geq 0,5 \end{cases}$$

φ_k – liczba błędów dla k -tego wzorca.

Dla wszystkich wzorców uczących będzie ta miara wyglądała:

$$\Gamma = \sum_{k=1}^L \varphi_k \quad (30)$$

gdzie: φ_k – błąd dla k -tego wzorca, L – liczba wzorców uczących, Γ – miara przystosowania osobnika.

Sieć może być uznana za nauczoną, jeżeli Γ będzie wynosiła dokładnie 0.

9.3. ALGORYTM EWOLUCYJNY Z MODYFIKACJAMI

Podstawowy algorytm działania **AG** w omawianym tutaj przypadku projektowania (uczenia) sieci jest podobny do typowego algorytmu z dostosowaniem do stosowanych tu struktur – schemat algorytmu pokazany jest na rysunku 54. Algorytm wymaga na wstępie stworzenia odpowiedniej macierzy plejotropowości i poligeniczności. Należy więc odgórnie nadać elementom macierzy odpowiednie wartości, przez co określić wpływ genów na fenoty. Może się to okazać, przy złym podaniu elementów macierzy, „katastrofalne” – sieć może nie być w stanie się nauczyć (np. wszystkie wejścia albo wyjścia są „odcięte” od innych neuronów). Jest to niewątpliwa wada takiego podejścia.

Modyfikacje algorytmu

Wymienioną wadę można usunąć, wprowadzając poszukiwanie możliwie dobrej macierzy plejotropowości i poligeniczności podczas działania algorytmu genetycznego. Dokonać tego można, próbując pewnych modyfikacji macierzy. Dostajemy zmieniony schemat działania, pokazany na rysunku 55.

Modyfikacje macierzy (poszukiwanie takiej, która umożliwiłaby nauczanie sieci) są oparte przede wszystkim na spostrzeżeniu, że zmiany dokonywane na jednej kolumnie macierzy wpływają na wartość jednej wagi połączenia w sieci (zgodnie ze wzorem mnożenia macierzy – patrz wzór (23)). Wyzerowanie zatem jednej kolumny w macierzy spowoduje likwidację jednego połączenia w sieci, gdy kolumna nie jest wyzerowana, odpowiednie połączenie powinno istnieć.

Operacje na kolumnach są zasadniczo traktowane jako operacje atomowe. Wprowadzenie pojęcia operacji atomowej wynika z tego, iż przy modyfikacjach macierzy zastosowano mechanizm zatwierdzania zmian. Mechanizm ten polega na wprowadzeniu danej zmiany „w życie” dopiero po sprawdzeniu, czy zmiana nie przynosi pogorszenia działania sieci. Sprawdzenie takie bazuje na ponownym obliczeniu przystosowania osobnika wykorzystującego tę macierz (przejście z genotypu na sieć neuronową, sprawdzenie wszystkich wzorców uczących) i porównaniu z przystosowaniem, jakie osobnik miał przed modyfikacją macierzy. Jest to, jak widać złożona operacja – dopuszczenie więc możliwości modyfikacji elementów mniejszych niż kolumna (czy wiersz) macierzy, takich jak pojedynczy element macierzy, może okazać się zbyt złożone obliczeniowo (i co za tym idzie – czasochłonne).

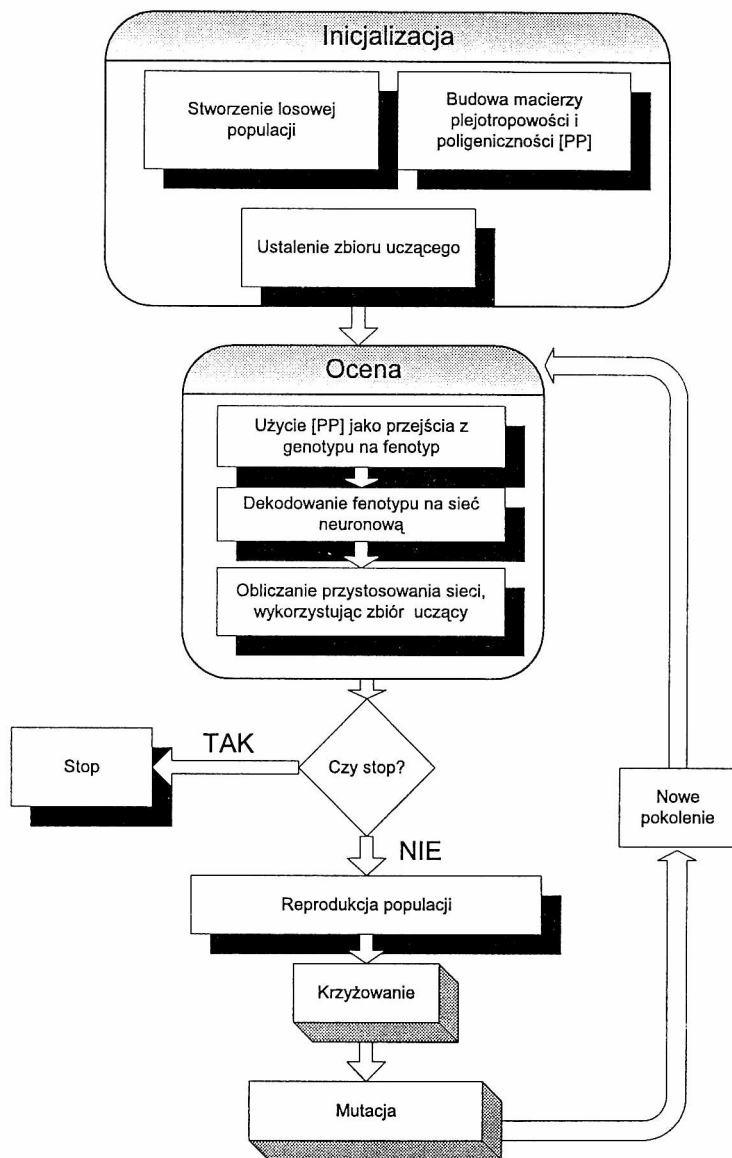
Czasami może się zdarzyć, że modyfikacja kilku kolumn daje polepszenie działania sieci, jednak modyfikacja każdej z tych kolumn dokonana oddzielnie zostanie odrzucona, bo takie pojedyncze modyfikacje nie poprawiają działania sieci. Nie będzie zatem możliwe doprowadzenie do polepszenia działania sieci. Aby temu zapobiec, wprowadzono dodatkowe operacje, modyfikujące wiele kolumn (wierszy), dopiero po wszystkich modyfikacjach zatwierdzające zmiany. Do operacji takich należy, np. modyfikacja zbioru kolumn, które są odpowiedzialne za wszystkie połączenia wejściowe (wyjściowe) do (z) danego neuronu – za jej pomocą można dodawać, usuwać, modyfikować neurony w większym stopniu. Zauważmy, że taka modyfikacja może wyzerować wszystkie wagi połączeń dochodzących do danego neuronu, co jest równoważne z jego usunięciem. Podobnie jest z dodawaniem neuronów do sieci.

Prezentowany algorytm, przez wprowadzenie wymienionych operatorów modyfikujących macierz pleiotropowości i poligeniczności, ma szansę nie mieć wad typowego algorytmu uczącego sieć neuronową – może służyć do projektowania sieci, gdyż ma zdolność szybkiego usuwania i dodawania neuronów do sieci. Stosowane są następujące „instrukcje ulepszające” macierz (po takiej instrukcji, i tylko takiej, musi nastąpić zatwierdzenie zmiany):

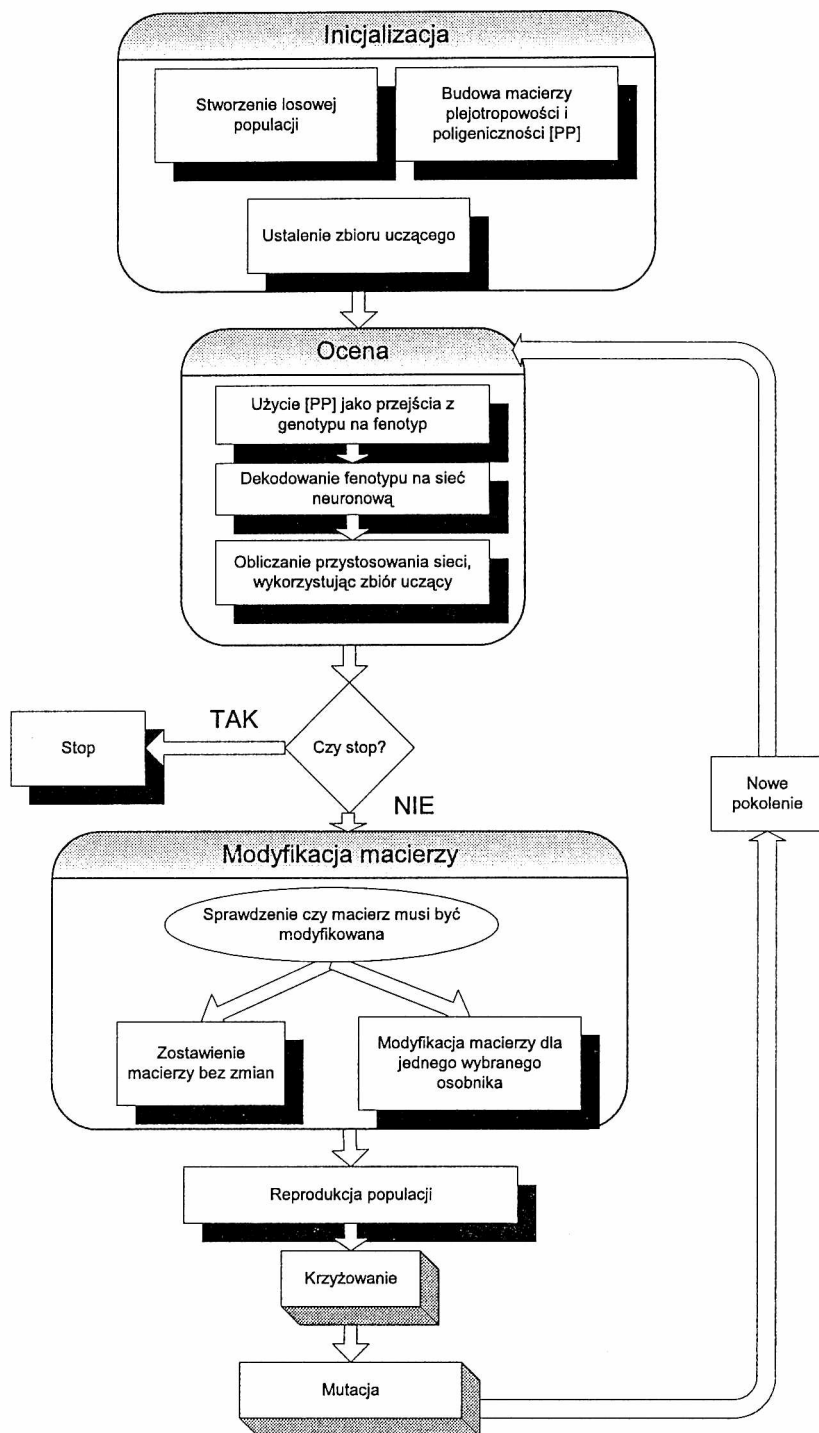
- Usunięcie, dodanie, modyfikacja połączeń do danego neuronu z warstwy wejściowej. Najczęściej tylko część neuronów potrzebuje połączenia z warstwą wejściową – np. niektóre neurony ukryte. Czasami takie połączenia mogą wprowadzać za duży „szum informacyjny” dla neuronu – może potrwać stosunkowo długo, zanim właściwy algorytm genetyczny zlikwiduje takie połączenia (przez wyzerowanie odpowiednich wag). Co jakiś czas jest sprawdzane, czy nie można takich połączeń zlikwidować. Likwidowanie wiąże się z ryzykiem, że zlikwidujemy za dużo takich połączeń (może nie być w ogóle przepływu informacji z wejścia do sieci) – potrzebna jest funkcja dodająca takie połączenia. Modyfikacja połączenia (tak w tej instrukcji, jak i w następnych) jest traktowana jako losowa zmiana wartości wagi połączenia (wag połączeń).

- Usunięcie, dodanie, modyfikacja połączeń od danego neuronu do warstwy wyjściowej. Sytuacja podobna do poprzedniej. Część neuronów nie musi być bezpośrednio połączona z wyjściem (np. rzadko kiedy istnieje potrzeba łączenia neuronów

z warstwy wejściowej bezpośrednio do warstwy wyjściowej). Z drugiej strony wyjście zazwyczaj powinno być połączone z jakimiś neuronami z warstwy ukrytej.



Rys. 54. Projektowanie NN za pomocą AG z wykorzystaniem [PP]
– podstawowy algorytm



Rys. 55. Projektowanie NN za pomocą AG z wykorzystaniem [PP] – modyfikacja

- Usunięcie, dodanie, modyfikacja neuronu warstwy ukrytej. Operacje dokonywane są na wszystkich kolumnach, będących odpowiednikami wag połączeń, które dochodzą do danego neuronu ukrytego. Co – jak już wspomniano – może usuwać, dodawać neurony.

- Usunięcie, dodanie, modyfikacja połączenia między dwoma neuronami. Operacja wykonywana jest na konkretnej kolumnie macierzy – specyfika jej działania była już opisana.

- Modyfikacje wierszy macierzy. Są one jednoznaczne ze zmianą natężania efektu pleiotropowego dla danego genu – dokonuje się w ten sposób najczęściej nieznacznej modyfikacji wszystkich wag.

- Modyfikacje losowo wybranego zbioru elementów macierzy. Jest to jedyna operacja, która nie bazuje na modyfikacji kolumny (wiersza) macierzy. Moc wybranego losowo zbioru elementów jest jednak porównywalna z liczbą elementów znajdujących się w kolumnie (wierszu) macierzy, więc nie ma tutaj zbyt dużych strat wydajnościowych, a pozwalamy na bardziej elastyczną modyfikację macierzy.

W algorytmie zastosowano pewne dodatkowe wskaźniki, sterujące wielkością mutacji, modyfikacją tablicy pleiotropowości i poligeniczności. Do wskaźników tych należą:

- Wskaźnik stabilizacji populacji – na podstawie średniego przystosowania oceniamy zmiany w populacji z pokolenia na pokolenie, im większa stabilizacja, tym większa powinna być mutacja.

- Wskaźnik podobieństwa genotypów w pokoleniu – dla każdego pokolenia populacji obliczane jest podobieństwo genotypów wszystkich osobników. Jeśli genotypy są zbyt podobne do siebie, należy je zróżnicować – mutacja.

- Wskaźnik podobieństwa rezultatów (dawanych na wyjściu sieci) – dla wszystkich osobników w pokoleniu sprawdzane jest podobieństwo rezultatów dawanych na wyjściu sieci. Jeżeli podobieństwo to jest zbyt duże, a podobieństwo genotypów jest bardzo małe – należy je zróżnicować.

Ciąg wykonywanych jedna po drugiej modyfikacji macierzy został nazwany „dostrajaniem macierzy”. Dostrajanie to, ze względu na warunek jego wystąpienia podczas działania algorytmu genetycznego, można podzielić na:

- startowe dostrajanie macierzy,
- zadane dostrajanie macierzy (należy do dynamicznych modyfikacji macierzy),
- „samoistne” dostrajanie macierzy (należy do dynamicznych modyfikacji macierzy).

Startowe dostrajanie macierzy (wstępny dobór macierzy)

Modyfikacje te pełnią funkcję doboru warunków startowych dla działania algorytmu genetycznego. Z populacji, która nie była jeszcze poddana działaniu algorytmu genetycznego, zostaje wybranych w losowy sposób kilka osobników. Dla każdego osobnika następuje próba takich modyfikacji macierzy **PP**, aby osobnik miał jak największe przystosowanie. Ostatecznie wybierana jest macierz dająca potencjalnie najlepsze przystosowanie.

Zadane dostrajanie macierzy

Modyfikacje następują podczas trwania ewolucji algorytmu genetycznego. Mogą one wystąpić dopiero po jednym wygenerowanym pokoleniu (algorytm genetyczny powinien zacząć już działać). Pokolenia, w których następuje optymalizacja dane są wzorem:

$$\text{aktualizacja}_{t+1} = \text{aktualizacja}_t \cdot \alpha + \beta \quad (31)$$

gdzie: aktualizacja_t – pokolenie, w którym następuje t -ta modyfikacja macierzy, α , β – współczynniki.

Współczynniki α , β , jak i pierwsze pokolenie, w którym następowało dostrajanie macierzy, były ustalane eksperymentalnie. Celem dostrajania było przyspieszenie działania algorytmu genetycznego. Dostrajanie było wykonywane dla aktualnie najlepszego osobnika w pokoleniu.

„Samoistna” optymalizacja macierzy

Celem tej optymalizacji jest usunięcie pewnej nieprawidłowości występującej czasami podczas działania algorytmu genetycznego. Nieprawidłowością tą jest małe zróżnicowanie fenotypu (co implikuje małe zróżnicowanie otrzymywanych przez sieć rezultatów) podczas dużego zróżnicowania genotypu. Sytuacja taka może świadczyć o złej budowie macierzy plejotropowości i poligeniczności. Następuje próba zróżnicowania fenotypów poprzez modyfikację macierzy i bez pogarszania działania sieci.

9.4. PRZEPROWADZONE EKSPERYMENTY

Wykorzystanie algorytmów genetycznych przy sieciach neuronowych przetestowane zostało na dwóch przykładach:

- XOR,
- Autoenkoder.

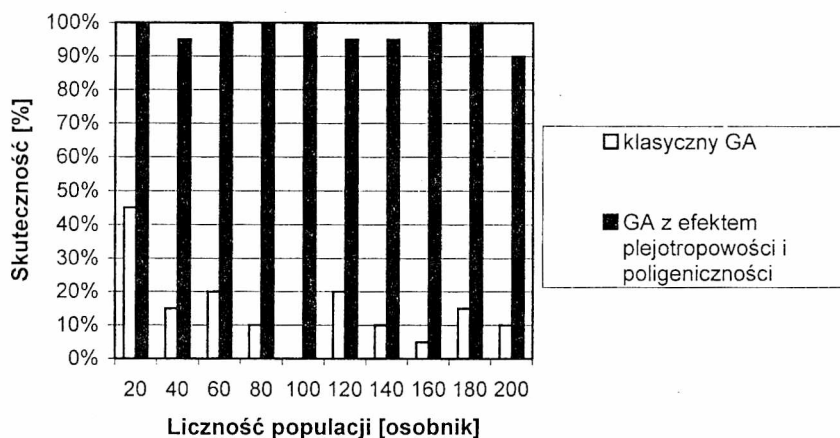
Warunki początkowe testów były zadawane w zewnętrznym pliku konfiguracyjnym. Plik ten umożliwia sformułowanie problemu, jaki ma być rozwiązywany, ustalenie początkowego zbioru wartości genów, warunków zakończenia testów, liczności populacji, rozmiaru genotypu, maksymalnej mutacji, warunków modyfikacji macierzy plejotropowości i poligeniczności, warunku dokonywania dodatkowego zróżnicowania osobników w populacji, współczynników związanych z optymalizacją macierzy. W pliku tym zapisane były również wzorce uczące, wykorzystywane później w trakcie uczenia sieci.

Działanie algorytmu genetycznego z efektem plejotropowości i poligeniczności było porównywane z klasycznym algorytmem genetycznym projektującym sieć neu-

ronową. Dla obydwóch algorytmów zawsze ustalana była ta sama maksymalna liczba neuronów ukrytych w sieci. W przeprowadzonych testach skoncentrowano się na badaniu wpływu macierzy plejotropowości i poligeniczności na efektywność działania algorytmu, czyli sprawdzaniu, jak często i jak szybko odpowiednie rozwiązania problemów są znajdowane. Niemal każdy test starał się pokazać jakiś inny aspekt wpływu macierzy plejotropowości i poligeniczności na działanie algorytmu. Skupimy się tutaj na pokazaniu tylko najciekawszych i jednocześnie reprezentatywnych wyników. Testowane problemy zawierają porównania skuteczności działania algorytmów (jak często są znajdowane rozwiązania) oraz liczb pokoleń potrzebnych na znalezienie rozwiązań. Czas przeprowadzanych testów, był w przypadku obu algorytmów (klasycznego i zmodyfikowanego) porównywalny.

XOR

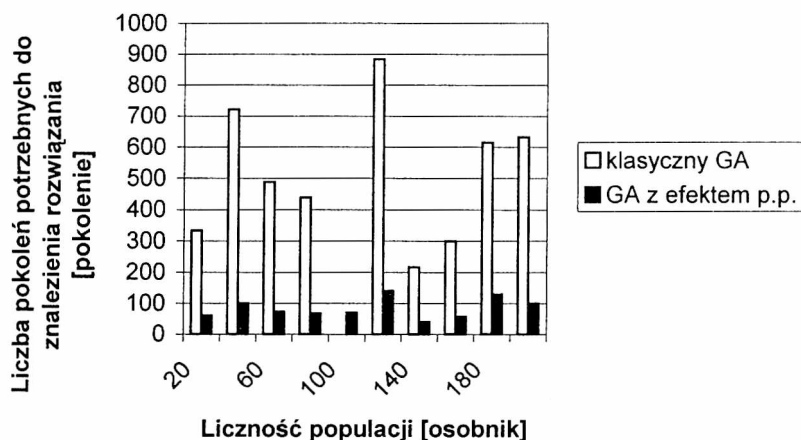
Celem testu było sprawdzenie, czy algorytm daje sobie radę z najbardziej znanym problemem dotyczącym sieci neuronowych. Zależność między wejściem a wyjściem nie jest złożona (jak już wiemy, jeden neuron ukryty wystarcza do zasymulowania tej zależności). Podczas wykonywania testu liczba pokoleń algorytmu genetycznego została ograniczona do 1000, tzn., jeżeli rozwiązanie nie zostało znalezione w tylu pokoleniach, przyjmowano, że nie zostało w ogóle znalezione. Dla każdej z zadanej liczności populacji (od 20 do 200, zmienianym z krokiem równym 20) test był powtarzany 20 razy. Wyniki podsumowuje rysunek 56.



Rys. 56. XOR – porównanie skuteczności działania algorytmów

Testy przeprowadzono dla różnych ograniczeń maksymalnej liczby neuronów, jakie mogła mieć projektowana sieć. Przy ograniczeniu liczby do jednego neuronu, klasyczny algorytm genetyczny rzadko umiał znaleźć rozwiązanie (raz, dwa razy na dwadzieścia powtórzeń), podczas gdy zmodyfikowany algorytm dawał dobre wyniki (podobne do tych z większą liczbą neuronów ukrytych). Dopiero przy liczbie trzech neuronów ukrytych algorytm klasyczny zaczął lepiej działać. Algorytm genetyczny

wykorzystujący efekt plejotropowości i poligeniczności miał rozmiar genotypu osobnika ustalony na 60 (rozmiar chromosomu w klasycznym **AG** zależy od liczby połączeń w sieci).



Rys. 57. XOR – liczba pokoleń potrzebna do znalezienia rozwiązania

Porównując działanie dwóch algorytmów, pierwszą zauważalną różnicą była różnica w ich skuteczności działania, czyli jak często rozwiązania były znajdowane.

Algorytm klasyczny rzadko znajdował rozwiązania, podczas gdy zmodyfikowany algorytm miał zawsze niemal stuprocentową skuteczność (jedna próba, czasami dwie na dwadzieścia bywały nieudane). Taka skuteczność działania wskazuje na algorytm wykorzystujący efekt plejotropowości i poligeniczności jako ten lepszy i pewniejszy. Jak widać na rysunkach 56 i 57, klasyczny **AG** lepiej sobie radził w małych populacjach niż większych. Wydaje się, że bardziej odgrywał tu rolę dryf genetyczny, niż „ukierunkowana” ewolucja⁶⁸.

Wyniki dotyczące skuteczności obydwóch algorytmów nie są reprezentatywne dla klasycznego **AG**, ponieważ średnia była liczona tylko dla testów zakończonych sukcesem, a tych nie było w tym przypadku za dużo – widoczna jest duża rozbieżność między testami przeprowadzonymi dla poszczególnych licznosci populacji. Jedynym wnioskiem, jaki można tutaj wysunąć jest to, że algorytm ten jest zauważalnie gorszy od algorytmu z efektem plejotropowości i poligeniczności. Efekt **PP** ma korzystny wpływ na działanie algorytmu.

⁶⁸ Ewolucja małych populacji ma nieco inne właściwości, istotną rolę odgrywa dryf genetyczny, co umożliwia np. zejście populacji po zboczu przystosowawczym i przejście na inny, możliwe, że wyższy szczyt przystosowawczy. Niestety taka ewolucja może być w stanie „odkryć” wysoki szczyt przystosowawczy, ale nie ma dobrych właściwości eksploatacyjnych – nie potrafi dostroić rozwiązania. Więcej na ten temat w (Kwaśnicka 1999).

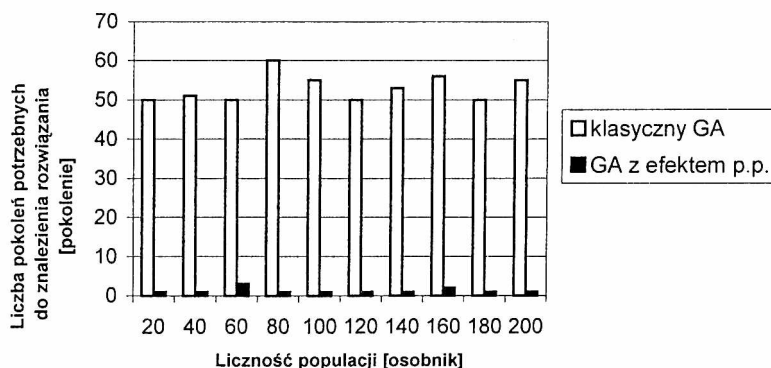
Autoenkoder

Test ten ma następującą postać:

Test	Wejście				Wyjście			
1								
2								
3								
4								

Celem testu było sprawdzenie, czy algorytm daje sobie radę z trochę bardziej skomplikowanym problemem, gdzie jest większa liczba wejść i wyjść. W teście tym wymagane są dwa neurony ukryte. Sprawdzano tutaj możliwość wygenerowania, na neuronach ukrytych, zapisu binarnego liczb od 0 do 3 (w teście jest zawsze zapalony jeden neuron wejściowy – jego pozycja w podawanym wzorcu mówi, jaka to jest wartość dziesiętna). Wyjście ma narzucone podobne ograniczenia. Sieć ma zatem zakodować w dowolny sposób liczbę na dwóch bitach, a następnie ją rozkodować do początkowej postaci. Test był przeprowadzany przy założeniach:

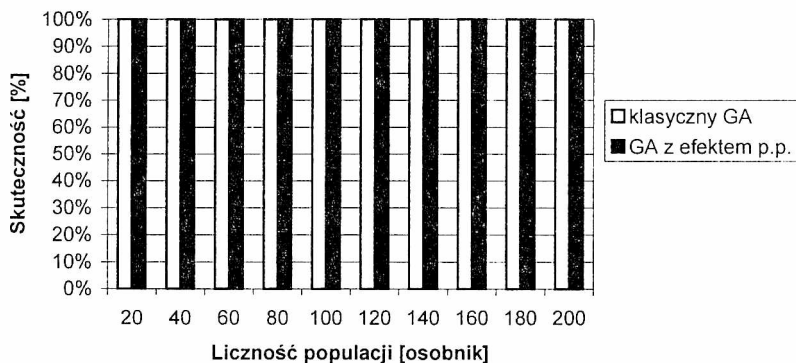
- jeżeli liczba pokoleń do znalezienia rozwiązania była większa od 1000, przyjmowano, że algorytm nie znalazł rozwiązania,
- genotyp (dla algorytmu wykorzystującego efekt plejotropowości i poligeniczności) składał się z 60 genów,
- maksymalna liczba neuronów ukrytych wynosiła 2 (stąd rozmiar genotypu dla klasycznego GA wynosił 39 geny),
- najlepszy osobnik przechodził z pokolenia na pokolenie,
- macierz plejotropowości i poligeniczności (dla algorytmu modyfikującego tę macierz podczas działania) była modyfikowana co 5 pokoleń od pokolenia pierwszego,



Rys. 58. Autoenkoder – porównanie skuteczności działania algorytmów

- testy były przeprowadzane dla populacji o licznosciach: od 20 do 200 co 20 osobników (tzn. 20, 40, 60, 80, ..., 180, 200),
- dla każdej licznosci test był przeprowadzany 20 razy.

Test ten również nie jest skomplikowany. Z otrzymanych rezultatów wynika, że jest prostszy niż test poprzedni. Zarówno w przypadku jednego, jak i drugiego algorytmu, mamy stuprocentową skuteczność – rozwiązanie jest zawsze znajdowane. Nie ma większego znaczenia wielkość populacji, ani wielkość chromosomu w przypadku zmodyfikowanego algorytmu (rys. 58).



Rys. 59. Autoenkoder – liczba pokoleń potrzebna do znalezienia rozwiązania

Średnia liczba pokoleń potrzebnych do znalezienia rozwiązania pozwala nam zaobserwować pozytywny wpływ efektu plejotropowości i poligeniczności na otrzymane rezultaty. W 90% przypadków algorytm genetyczny wykorzystujący ten efekt, już w punkcie startu znajdował poprawne rozwiązanie (rys. 58). Klasyczny algorytm genetyczny średnio potrzebował około 50–60 pokoleń, aby takie rozwiązanie znaleźć.

Znajdowanie rozwiązania już w momencie startu działania algorytmu, spowodowane jest wstępną optymalizacją macierzy plejotropowości i poligeniczności (optymalizacja warunków startowych). Przy rozwiązywaniu poprzedniego problemu, nie można było dostrzec, jak duży wpływ ma taka optymalizacja. Tutaj większość dokonywanych testów kończyła się jeszcze przed działaniem właściwego algorytmu genetycznego (nie był używany jeszcze żaden operator genetyczny) – musiała zatem działać optymalizacja macierzy.

9.5. OMÓWIENIE WYNIKÓW

W prezentowanych eksperymentach próbowano wstępnie zbadać wpływ efektu plejotropowości i poligeniczności na działanie algorytmu genetycznego, zastosowane-

go do projektowania sieci neuronowych. Chodziło o sprawdzenie, czy algorytm wykorzystujący ten efekt może być ciekawą alternatywą dla klasycznego algorytmu genetycznego.

Jak pokazały przeprowadzone badania, algorytm z odpowiednim użyciem tego efektu może dawać interesujące rezultaty. Dla każdego testowanego problemu, algorytm wykorzystujący efekt plejotropowości i poligeniczności miał zawsze najlepszą skuteczność w znajdowaniu rozwiązań – nie spadała ona poniżej 85%. Liczba pokoleń, potrzebnych do znalezienia rozwiązania, też zazwyczaj była najmniejsza właśnie w przypadku tego algorytmu. Wyniki badań wskazują na to, że zarówno skuteczność, jak i liczba pokoleń, potrzebnych na znalezienie rozwiązania, w zasadzie nie zależą zbyt mocno od liczebności populacji (nie były jednak testowane sytuacje skrajne). Algorytm działa podobnie dla wszystkich testowanych licznosci. Niestety, proces przejścia z genotypu na fenotyp dla osobnika – to jedno mnożenie macierzy plejotropowości i poligeniczności przez macierz genotypu. Taka operacja jest czasochłonna i zauważalnie spowalnia działanie algorytmu.

Klasyczny algorytm genetyczny nie miał tak dobrej skuteczności. Algorytm ten jest bardzo czuły na licznosc populacji. Wraz z jej wzrostem skuteczność algorytmu rosła oraz spadała natomiast liczba potrzebnych do znalezienia pokoleń. Przy małych populacjach algorytm rzadko kiedy znajdował rozwiązanie. Jedyną zaletą klasycznego algorytmu, w porównaniu ze zmodyfikowanym algorytmem, jest szybki czas ewolucji jednego pokolenia. Dla tak samo licznych populacji klasyczny algorytm, gdy znajdował rozwiązanie, mimo większej liczby potrzebnych na to pokoleń, zazwyczaj robił to w czasie krótszym niż zmodyfikowany algorytm. Jednak to, aby klasyczny algorytm znajdował tak samo często rozwiązania (skuteczność) jak algorytm zmodyfikowany, można sobie zapewnić dopiero przy dużych populacjach. Wtedy jednak różnica w czasie nie jest już tak znaczna i zmodyfikowany algorytm może okazać się szybszy (jeśli znajduje rozwiązanie w mniejszej liczbie pokoleń). Jak już wspomniano, klasyczny algorytm wykazuje czulość na licznosc populacji, podczas gdy zmodyfikowany algorytm działa zawsze podobnie. Często zmodyfikowany algorytm już dla populacji o licznosci 20 osobników miał lepszą skuteczność i potrzebował zauważalnie mniej pokoleń do znalezienia rozwiązania niż klasyczny algorytm przy populacji o licznosci 200 osobników. Czas ewolucji pokolenia był wtedy porównywalny dla obu algorytmów, czyli algorytm wykorzystujący efekt plejotropowości i poligeniczności wydaje się być lepszą alternatywą, jeżeli chodzi o szukanie odpowiednich sieci neuronowych.

Innym zagadnieniem jest to, jak mocno wpływa macierz plejotropowości i poligeniczności i jej modyfikacja na efektywność działania algorytmu. Przeprowadzone testy pozwalają na wyciągnięcie pewnych wniosków. Wstępna modyfikacja macierzy plejotropowości i poligeniczności pozwala na optymalizację warunków startowych, co pozwala już w pierwszym pokoleniu znaleźć rozwiązania (widać to w przypadku „auto-inkodera”). Osobniki już w pierwszym pokoleniu są dobrze przystosowane, o wiele lepiej niż w przypadku klasycznego algorytmu.

Dodatkową zaletą wykorzystywania przez algorytm genetyczny efektu pleiotropowości i poligeniczności jest zazwyczaj mniejsza „zajętość” pamięci. Widoczne jest to w dużych populacjach, wtedy, gdy rozmiar genotypu jest mniejszy niż liczność populacji. Wynika to z reprezentacji macierzy w pamięci (zajętość pamięciowa macierzy determinowanej przez genotyp, o rozmiarze równym np. 20, jest taka sama jak zajętość pamięci 20 genotypów w przypadku klasycznego algorytmu). Przeprowadzone badania sugerują, że wielkość genotypu nie ma większego znaczenia (najtrudniejsze testy były wykonywane przy najmniejszym rozpatrywanym rozmiarze genotypu i dawały dobre rezultaty) i zazwyczaj jest ona mniejsza od liczności populacji.

Algorytm prawdopodobnie może być użyty do rozwiązywania skomplikowanych problemów. Przeprowadzone badania pozwalają przypuszczać, że będzie on miał lepsze rezultaty niż klasyczny algorytm genetyczny. Przy takich problemach ważne mogą okazać się optymalizacje wykorzystywanego efektu pleiotropowego i poligenicznego – optymalizacja warunków startowych, jak i optymalizacje dynamiczne, dokonywane podczas ewolucji osobników.

Możliwe ulepszenia, dalsze kierunki prac

Działanie algorytmu wykorzystującego efekt pleiotropowości i poligeniczności można próbować ulepszyć. Ulepszenia te związane mogą być z optymalizacjami macierzy pleiotropowości i poligeniczności. To macierz ta nadaje kierunek rozwoju osobnikom z populacji, na której działa algorytm genetyczny. Odpowiednio modyfikując tę macierz, można działanie algorytmu znacząco przyspieszyć – już teraz widać, że modyfikacje powodują przyspieszenie działania algorytmu i szybsze znajdowanie rozwiązania.

Dotychczasowe modyfikacje macierzy działają „na oślep”, wykonywane są w losowych miejscach macierzy. Niestety, nie wiadomo jak tę macierz odpowiednio modyfikować, tak aby metoda modyfikacji była uniwersalna dla różnych rozwiązywanych problemów.

Drugim problemem jest ustalenie początkowej macierzy – optymalizującej warunki startu. Kwestie, jakie trzeba tu rozwiązać, są związane z wyborem odpowiedniego osobnika (grupy osobników), dla którego macierz jest optymalizowana, sposobu optymalizacji macierzy (nie może tu być próby modyfikacji każdego elementu osobno, bo przy dużych macierzach jest to zbyt czasochłonne, porównywalne z całym procesem ewolucji populacji).

W pracy tej nie przeprowadzono kompleksowych badań dotyczących właściwości uogólniania wiedzy przez stworzone sieci. Jednak wstępne badania, jakie przeprowadzono (wyniki nie zostały w tej pracy zademonstrowane) dla „dodawania liczb dwubitowych” i „rozpoznawania cyfr” mogą świadczyć o lepszym uogólnianiu wiedzy przez sieci generowane z użyciem algorytmu genetycznego wykorzystującego efekt pleiotropowości i poligeniczności niż klasycznego algorytmu genetycznego. Opisywane optymalizacje macierzy zdają się działać i usuwać niepotrzebne połączenia oraz

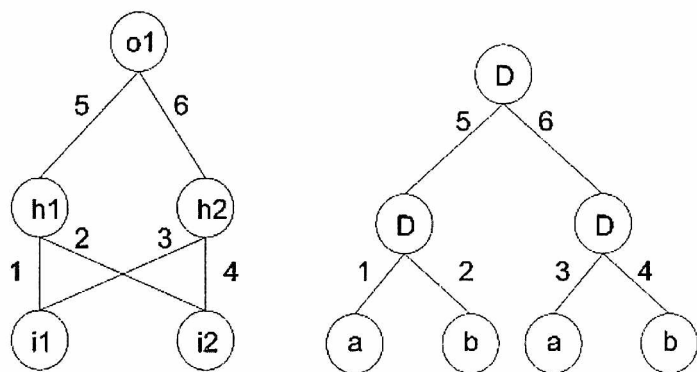
neurony. Jednak nie zostało to zbyt dokładnie zweryfikowane, wymaga jeszcze dalszych, bardziej wszechstronnych badań.

Znalezienie właściwego sposobu modyfikacji macierzy plejotropowości i poligeniczności – który daje dobre rezultaty nie zajmując przy tym zbyt wiele czasu, nawet dla skomplikowanych zadań – może być podstawą dalszych prac.

10. PROGRAMOWANIE GENOWE (GEP) W PROJEKTOWANIU SSN⁶⁹

Zdaniem Ferreiry (Ferreira 2002) GEP jest wygodnym narzędziem do projektowania sieci neuronowych. GEP pozwala na zakodowanie sieci neuronowej o różnym rozmiarze i kształcie w postaci liniowego chromosomu o stałej długości. Sieć zakodowana jest w strukturze składającej się z głowy i ogona (patrz rozdz. 3.5). Głowa zawiera funkcje aktywacji dla neuronów ukrytych i wyjściowych oraz symbole terminalne, które reprezentują wejściowe neurony. Ogon zawiera tylko symbole terminalne. Poza głową i ogonem geny zawierają dodatkowe dziedziny: D_w i D_t , kodujące odpowiednio wagi i progi. Strukturalnie, D_w o długości d_w równej hn (n jest maksymalną liczbą argumentów funkcji) i D_t o długości d_t równej h , położone są bezpośrednio za ogonem. Strukturę reprezentującą sieć neuronową autorka nazywa NN-genem (NN-gene).

Dla każdego NN-genu na początku ustalane są wagi i progi, w trakcie ewolucji ulegają one modyfikacjom. Zaproponowano też specjalizowane operatory genetyczne w celu zapewnienia odpowiedniego zróżnicowania wag i progów.



Rys. 60. Reprezentacja osobnika w GEP (na podstawie (Ferreira 2002))

⁶⁹ Cały ten rozdział jest napisany na podstawie pracy (Ferreira 2002).

Rozważmy sieć neuronową o dwóch wejściach (i_1 i i_2), dwóch ukrytych neuronach (h_1 i h_2), z jednym neuronem wyjściowym (o_1), z progami równymi jeden – dla uproszczenia można je pominąć. Taka sieć może być też reprezentowana w postaci drzewa (rys. 60).

Wejścia sieci, i_1 i i_2 są oznaczone odpowiednio jako a i b , symbol D oznacza funkcję sumowania przemnożonych przez odpowiednie wagi dwóch poddrzew i policzenia aktywacji. Drzewo takie można przedstawić w postaci struktury liniowej (jaśniejsze cyfry kodują wagi):

0123456789012
DDDabab654321

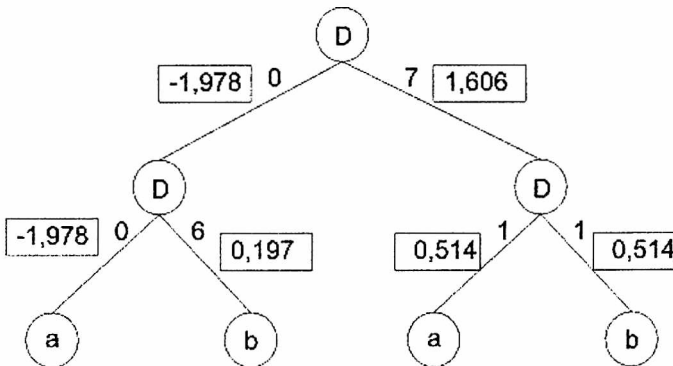
Wartości wag są pamiętane w macierzy i są odczytywane w miarę potrzeby. Sieć neuronowa dla problemu XOR, przy założeniu $h=3$ $d_w=6$ może być zakodowana w następujący sposób:

0123456789012
DDDabab701160

Założmy dalej następującą postać Dw (elementy w Dw numerowane są od zera):

{-1,978, 0,514, -0,465, 1,22, -1,686, -1,797, 0,197, 1,606, 0, 1,753}

Konwersja tego chromosomu daje drzewo pokazane na rysunku 6 (w prostokątach wpisane są wartości wag połączeń).



Rys. 61. Reprezentacja drzewiasta sieci neuronowej dla zadania XOR (na podstawie (Ferreira 2002))

10.1. SPECJALIZOWANE OPERATORY PRZESZUKUJĄCE

Zaproponowane dla „tradycyjnego” GEP operatory mogą być łatwo dostosowane do zadania projektowania sieci neuronowych. Specjalne operatory zaproponowano do

działania na zestawie wag (D_w) i progów (D_t) w celu rozprzestrzeniania się wag i progów w populacji – transpozycja. Bezpośrednią mutację wprowadzono w celu wprowadzania nowych wartości wag i progów.

Transpozycja specyficzna dla dziedzin D_w i D_t

Operator ten wybiera losowo: chromosom, gen z odpowiadającymi mu D_w lub D_t , pierwszą pozycję transpozycji, długość obszaru transpozycji i docelowe miejsce (wewnątrz D_w lub D_t); następnie przesuwa wybrany fragment do miejsca docelowego.

Przykład

Mamy chromosom ($h = 4$, obszary D_w i D_t rozróżnione są podwójnymi pionowymi kreskami, **T** koduje funkcję o trzech argumentach, **Q** – funkcję o czterech argumentach):

```
01234567890123456 | | 7890123456789012 | | 3456
DTQaababaabbaabba | | 0571745736284668 | | 2867
```

Załóżmy, że do transpozycji wybrano fragment 84668, a jako miejsce wstawienia wylosowano liczbę 4. Efekt takiej transpozycji jest następujący:

```
01234567890123456 | | 7890123456789012 | | 3456
DTQaababaabbaabba | | 0571846687457362 | | 2867
```

Efektem działania tego operatora są różne rozmieszczenia wag w sieci, co oczywiście daje inaczej działającą sieć.

Dwupunktowa, wewnątrzgenowa rekombinacja (*Intragenic two-point recombination*)

Jest to typowa rekombinacja dwupunktowa z tą różnicą, że punkty rekombinacji są wybrane w obrębie jednego genu. Działa na dwóch rodzicielskich osobnikach. Załóżmy dwa rodzicielskie osobniki, każdy z nich ma ten sam zestaw wag:

```
0123456789012345601234567890123456
TTaababaab14393255QDbabbabb96369304
Qaabbbabb97872192QDbabbaaa81327963
```

Załóżmy dalej, niech do rekombinacji wybrane będą gen 0 i punkty 1 (pomiędzy pozycją 0 a 1) oraz 12 (pomiędzy pozycją 11 a 12). Otrzymamy potomstwo:

```
0123456789012345601234567890123456
Taaabbbabb97893255QDbabbabb96369304
QTaababaab14372192QDbabbaaa81327963
```

Dlatego samego zestawu wag u obydwóch osobników, wskutek rekombinacji różna będzie ich ekspresja.

Bezpośrednia mutacja wagi lub progów

Jest to losowo wybrana waga zastępowana przez losowo wybraną wartość. Zdaniem autorki, lepsze efekty otrzymuje się, jeśli ten operator jest wyłączony.

10.2. MOŻLIWOŚCI ZAPROPONOWANEGO PODEJŚCIA

Do badań autorka podejścia (Ferreira 2002) wykorzystała dwa problemy logiczne:

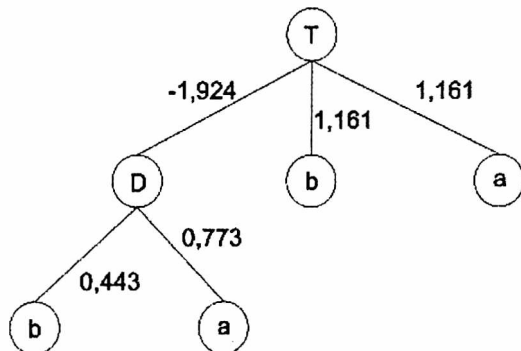
- problem XOR,
- sześciobitowy multiplexer.

Omówimy pokrótce otrzymane wyniki. W obu przypadkach założono zbiór dziesięciu wag z przedziału $[-2, 2]$. W ocenie generowanych sieci uwzględniano błąd produkowany przez sieć dla wszystkich wzorców reprezentujących zadanie. Najlepszy osobnik był zachowywany w trakcie ewolucji (elitaryzm).

Problem XOR

Zastosowano trzy funkcje: D – z dwoma argumentami, T – z trzema argumentami i Q z czterema argumentami. Ustalono $h = 4$, co pozwala reprezentować dużą przestrzeń potencjalnych rozwiązań (długość chromosomu wynosi 33). W tzw. *wersji kompaktowej* autorka użyła chromosom o długości 17 przy $h = 2$. Są dwa symbole terminalne oznaczone jako a i b . Macierz wag składała się z dziesięciu elementów. Jako ocenę osobnika zastosowano sumaryczny procentowy błąd dla wszystkich (czterech) wektorów wejściowych. Ponieważ należy minimalizować taki błąd, więc przystosowaniem jest ten błąd odjęty od pewnej stałej. Selekcja dokonywana była metodą ruletki.

Algorytm znalazł wiele rozwiązań, zazwyczaj są to dość duże sieci, ale wśród licznych rozwiązań były też sieci o małych rozmiarach, np. ta z rysunku 62. Można zatem stwierdzić, że w tym podejściu algorytm również jest w stanie znaleźć optymalną sieć w sensie jakości jej działania i rozmiaru. Ta sieć została znaleziona przez *kompaktową* wersję programu, tzn. z głową o długości 2. Różnice w działaniu algorytmów dla dwóch długości głowy: $h = 4$ i $h = 2$ są zgodne z oczekiwaniami. Większa przestrzeń poszukiwań powoduje, że sprawność algorytmu, mierzona procentem prób zakończonych sukcesem w stu uruchomieniach po 50 pokoleń ewolucji każde, jest większa niż dla krótszych chromosomów, z głową równą 2–77% wobec odpowiednio 30%, ale znajdowane sieci są dużych rozmiarów.



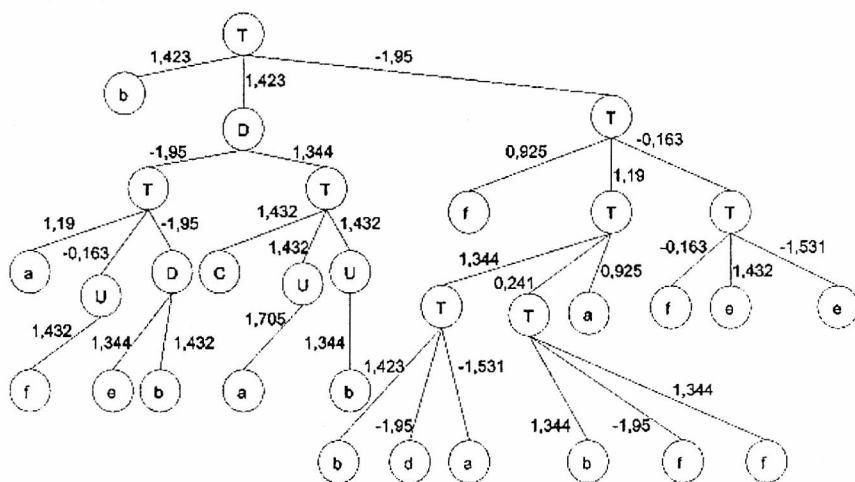
Rys. 62. NN dla zadania XOR (na podstawie (Ferreira 2002))

Sześciobitowy multiplekser

Zadanie polega na odkodowaniu dwubitowego binarnego adresu (00, 01, 10, 11) i zwróceniu wartości rejestrów danych (d_0, d_1, d_2, d_3). Multiplekser jest funkcją czterech aktywności: dwóch (a_0 i a_1) określających adres i czterech (d_0 do d_3) określających odpowiedź. Są możliwe $2^6 = 64$ kombinacje – tyle jest wektorów uczących. Jako przystosowanie zastosowany jest wzór

$$f_i = \begin{cases} n & \text{dla } n \geq \frac{1}{2} C_i \\ 1 & \text{w przeciwnym razie} \end{cases} \quad (32)$$

gdzie: n jest liczbą wektorów uczących z poprawną odpowiedzią, C_i to liczba wektorów uczących (równa 64).



Rys. 63. Najlepsza sieć neuronowa znaleziona dla zadania 6-multiplekser (na podstawie (Ferreira 2002))

Zrezygnowano z czteroargumentowej funkcji Q, założono tylko jedno-, dwu- i trójargumentowe funkcje U, D, T, oraz symbole terminalne $\{a, b, c, d, e, f\}$ reprezentujące odpowiednio $\{a_0, a_1, d_0, d_1, d_2, d_3\}$. Testowano podejście jednogenowe i czterogenowe, w którym składowe sieci są łączone węzłem logicznym OR. Ewoluwano populację liczącą 30 osobników przez 2000 pokoleń. Próby powtarzano 100 razy. Współczynnik sukcesu był stosunkowo niski, dla ewolucji osobników jednogenowych wynosił 4%, a dla osobników czterogenowych – 6%. Najlepsza znaleziona sieć pokazana jest na rysunku 63.

GEP jest stosunkowo nowym podejściem. Jak twierdzi sama autorka podejścia, multiplekser jest najbardziej złożonym zadaniem, jakie rozwiązano, wykorzystując GEP. Zamieszczony (Ferreira 2002) wykres zmian przystosowania w ewolucji zakończonej sukcesem pokazuje, że średnie przystosowanie w populacji waha się, niewiele wzrastając w trakcie 2000 pokoleń, od około 32 do 35, nie przekraczając w trakcie ewolucji wartości 42. Najlepszy osobnik (z przystosowaniem 43 w zerowym pokoleniu) osiąga wartość 64 (pełna poprawność) w okolicach 1100 pokolenia.

11. PODSUMOWANIE

Pierwszy rozdział tej pracy kończył się krótką notką o niektórych próbach modelowania mózgu. Podsumowanie jednej z wcześniejszych książek autorki zaczynało się od fragmentu:

„Zabawa w naśladowanie natury trwa. Czy tylko zabawa? Czy tak właśnie można nazwać projekt pod kierunkiem Hugo de Garis’a, realizowany przez międzynarodowy zespół ludzi (w połowie 1998 roku zaangażowanych było 20 naukowców z 9 krajów) w *Advanced Telecommunications Research*, w japońskim mieście Kioto (Buller, 1998). Projekt „CAM-Brain”, którego celem jest opracowanie sztucznego mózgu, istnieje już od kilku lat. Po ośmiu latach realizacji powinien powstać sztuczny mózg, składający się z miliarda neuronów, o inteligencji małego kota (*Robokoneko* to japońska nazwa małego kociaka-robota – ma być on gotowy i publicznie pokazywany pod koniec 1999 roku). Jest to pierwszy etap projektu mającego za główny cel budowę sztucznego mózgu o inteligencji dorównującej człowiekowi. CAM to nazwa pochodząca od ang. *Cellular Automata Machine* – maszynowa implementacja automatu komórkowego. Projektowany mózg to elektroniczny automat komórkowy. Mocno upraszczając, automat komórkowy to specyficzna symulacja komputerowa: definiuje się reguły dotyczące jak ma zachować się komórka, w zależności od jej aktualnego stanu i stanów sąsiednich komórek. Układ komórek zmienia się dynamicznie, ponieważ komórki kolejno pobierają odpowiednie dla nich reguły i zmieniają się zgodnie z nimi. Wydaje się proste, ale tylko zasada działania jest prosta, natomiast jak zbudować dobrze funkcjonujący *CAM-Brain*?” (Kwaśnicka 1999, s. 156).

Dziś już wiemy to, co wtedy można było tylko przewidywać, że projekt „CAM-Brain” już nie istnieje. Sztuczny mózg – jeszcze nie istnieje. Czy wzorując się na naturze, można wyewoluować sztuczny mózg? Zajrzyjmy na stronę Artificial Development (AD) – <http://www.ad.com/default.asp>. Może jeszcze w chwili, gdy Czytelnik ma w ręku tę książkę, istnieje i strona, i firma, i projekt. Z całą pewnością istnieją w momencie pisania tych słów. Celem Artificial Development⁷⁰ jest wykonanie realistycznej symulacji mózgu w celu wytwarzania nowych produktów kognitywistyki ob-

⁷⁰ „Our goal is to achieve a realistic whole-brain simulation for the purpose of creating new cognitive computational products” (cytat ze strony <http://www.ad.com/default.asp>). Artificial Development powstała w lipcu 2003 roku w celu zbudowania pełnego symulatora ludzkiej kory mózgowej. Jej założycielem był Marcos Guillen.

liczeniowej⁷¹. AD zamierza wykorzystać najnowsze osiągnięcia badań dotyczących systemu nerwowego (*neuroscience*). Realizowany w AD CCortex™ model jest biorealistyczną siecią neuronową symulującą ludzką korę i systemy peryferyjne. W zamierzeniu prace AD umożliwią powstanie zaawansowanych funkcjonalnie systemów obliczeniowych, a rozwinięte w ramach projektu technologie będą miały szeroki wpływ na badania, począwszy od badań medycznych po zaawansowane systemy autonomiczne. Docelowo CCortex to $20 \cdot 10^9$ neuronów i $20 \cdot 10^{12}$ połączeń, co pozwoli na złożoność konkurencyjną w stosunku do mózgów ssaków. Będzie to jednocześnie największa i najbardziej biologicznie realistyczna sieć neuronowa, jaka kiedykolwiek została zbudowana. System ma być dziesięć tysięcy razy większy niż wcześniejsze podejścia, które zakładały zamodelowanie głównych charakterystyk ludzkiej inteligencji.

Zastosowanie algorytmów ewolucyjnych do projektowania sztucznych sieci neuronowych w celu rozwiązania danego zadania, np. klasyfikacji danych medycznych, jest dalekie od modelowania mózgu. Niemniej jest to jedno z możliwych podejść do bardzo trudnego zadania, jakim jest znalezienie sieci neuronowej, dobrze realizującej przeznaczone jej zadanie.

Jak to już wcześniej wspomniano, celem tej pracy było przekazanie własnych doświadczeń na temat różnych, ewolucyjnych podejść do projektowania sieci neuronowych. Należy nadmienić, że nie wszystkie eksperymenty zostały tu omówione. Doświadczenia z wcześniejszych prac, np. (Kwaśnicka, Kilińska 1995, Kwaśnicka, Szerszon 1997), w których ewoluowano również sieci rekurencyjne – neurony wewnętrzne mogły być dowolnie połączone między sobą, zostały wykorzystane podczas opracowywania systemu GANN, pozwalającego na bardziej wszechstronne badania. Wydaje się, że w podsumowaniu powinny być postawione dwa pytania:

1. Jakie zalecenia wynikają z przeprowadzonych eksperymentów?
2. Jakie dalsze badania wyłaniają się z dotychczasowych doświadczeń?

Spróbujemy krótko odpowiedzieć na te dwa pytania, w kolejności. Po pierwsze, warto podkreślić, że badania jednoznacznie wykazują znaną, choć czasami jakby zapomnianą, właściwość algorytmów genetycznych: efektywność algorytmu genetycznego silnie zależy od przyjętego sposobu kodowania i zdefiniowanej miary przystosowania. Mamy zwykle świadomość roli, jaką w algorytmach ewolucyjnych odgrywają operatory genetyczne i ich częstość, oraz inne parametry algorytmu, jak np. rozmiar ewoluującej populacji. Potrafimy spędzić dużo czasu, próbując dostroić parametry algoryt-

⁷¹ „Mianem Nauk Kognitywnych określa się badania nad zespołem zagadnień, których wspólnym mianownikiem są procesy poznawcze człowieka i ich modele. Różne aspekty tych zagadnień leżą w zakresie badań neurofizjologów, psychologów, językoznawców, logików, antropologów i informatyków, a lista ta daleka jest od pełności. ... Nauki kognitywne obejmują psychologię poznawczą, neurofizjologię, fizykę umysłu, lingwistykę kognitywną, logikę, informatykę, filozofię umysłu i kilka innych dyscyplin. Wspólne zainteresowanie dotyczy funkcjonowania mózgu oraz systemów naśladujących funkcje mózgu: zbierających i przetwarzających informacje oraz reagujących na nie.” W. Duch, <http://www.uni.torun.pl/~cogsci/coto.htm>

mu, opracować specjalizowane operatory genetyczne itp. Tak, to wszystko jest ważne. Ale decyzja o sposobie kodowania osobnika wpływa na wielkość przestrzeni poszukiwań – przestrzeni potencjalnych rozwiązań. Może być ona bardzo duża, w skrajnej sytuacji może reprezentować wszystkie, możliwe rozwiązania. Teoretycznie jest to bardzo pozytywne, algorytm ewolucyjny, pracując z taką przestrzenią, potencjalnie może znaleźć globalne optimum – najlepsze możliwe rozwiązanie. Widać to w eksperymentach pokazanych krótko w ósmym rozdziale: najmniejsze sieci dla XOR i zadania parzystości liczby (rys. 41, 42). Warto podkreślić, że użycie słowa *teoretycznie* w zdaniu powyżej nie było przypadkowe. Taka przestrzeń jest bardzo duża, jeśli przy tym będzie wielomodalna i bardzo zwodnicza, pełna gwałtownych zmian, nieciągła, to znalezienie rozwiązania globalnego będzie bardzo trudne, wymagające bardzo długiej ewolucji, wyniki będą niepowtarzalne, co będzie oznaczało, że duży wpływ na uzyskane rozwiązanie będzie mieć czynnik losowy. Zaprojektowanie innego sposobu kodowania, ograniczającego przestrzeń rozwiązań (np. kodowanie sieci w postaci gramatyki) spowoduje, że akceptowalne rozwiązanie może być osiągnięte znacznie szybciej. Jednakże najlepsze rozwiązanie może być niereprezentowalne w przestrzeni poszukiwań, nie będzie zatem mogło być znalezione. Należy o tym pamiętać podczas określania sposobu kodowania. Pierwsze zalecenie wynikające z dotychczasowych doświadczeń możemy sformułować następująco:

Jeśli jesteśmy w stanie część ‘nieobiecującej’ przestrzeni poszukiwań potencjalnych rozwiązań wyeliminować poprzez odpowiednie zdefiniowanie sposobu kodowania, to zadanie dla algorytmu ewolucyjnego będzie łatwiejsze. Należy przy tym uważać, aby ‘obiecujące’ rozwiązania były reprezentowalne w przyjętym kodowaniu.

Rozważmy teraz rolę funkcji przystosowania. Definiując ją, ukierunkowujemy algorytm genetyczny. Kładąc na przykład nacisk na otrzymywanie małych sieci neuronowych przez silne zwiększanie przystosowania małych sieci, możemy doprowadzić, że algorytm nie będzie w stanie wyewoluować sieci, która potrafi się dobrze nauczyć danego jej zadania. Z drugiej strony, nie zwracając uwagi na rozmiar sieci, algorytm zazwyczaj bardzo szybko generuje sieć neuronową, która poprawnie uczy się na danych uczących, ale jest za duża, uczy się „na pamięć” i nie potrafi uogólniać. Efekt ten uwidocznił się już w pierwszych pracach, gdzie ucieczką od tego problemu było przedłużanie ewolucji wystarczająco długo, promując sieć mniejszą spośród jednakowo nauczonych. W systemie ANN (Kwaśnicka, Kilińska 1995) starałyśmy się równoważyć rozmiar i błąd sieci w jej ocenie. Testowane trzy warianty funkcji przystosowania pozwalają zdefiniować kolejne zalecenie:

W zadaniu projektowania sieci neuronowych przystosowanie sieci powinno być zróżnicowane w trakcie ewolucji: w pierwszej fazie powinno kłaść nacisk na jakość uczenia sieci, a następnie włączyć rozmiar sieci. Należy zwłaszcza dokładnie rozważyć rozwiązywany problem, dobrze zdefiniować cechy „dobrego rozwiązania” i starać się tak zdefiniować funkcję przystosowania, aby w ocenie rozwiązania równoważyła wszystkie zdefiniowane cechy.

Trzecie zalecenie wynika z porównania różnych podejść ewolucyjnych: algorytm genetyczny, podejście z plejotropowością i poligenicznością oraz GEP. Badania wła-

sne autorki, jak i cytowane badania z GEP wydają się nie zachęcać do stosowania bardziej złożonych algorytmów ewolucyjnych. I to nie z przyczyny, że nie radzą sobie dobrze z tym zadaniem. Jak mogliśmy zauważyć, zastosowanie K-modelu daje dobre wyniki, algorytm jest skuteczny, zwłaszcza dla trudniejszych zadań, ale wymaga ustalania więcej parametrów. Nie jest tak przyjazny dla użytkownika, wymaga większej wiedzy o samym algorytmie. Podejście GEP – uważane przez Ferreirę (Ferreira 2002) za dobre, wydaje się jednak zbyt mało skuteczne – skuteczność na poziomie 30–77% dla XOR i 4–6% dla multipleksera wydaje się być wysoce niezadowolająca. Warto nadmienić, że w ocenie sieci autorka brała pod uwagę błąd działania sieci, pomijając ich rozmiar. Maksymalny rozmiar wynikał z przyjętych parametrów – długości głowy i wymiarowości funkcji. Po dotychczasowych doświadczeniach wydaje się, że można zdefiniować kolejne, następujące zalecenie:

Zaawansowane modele ewolucyjne można wykorzystywać dla złożonych zadań, starając się dołożyć wysiłku w celu zrozumienia wpływu poszczególnych parametrów na efektywność i skuteczność stosowanego podejścia. Dla prostszych zadań lub jeśli zadanie realizuje mało doświadczona osoba, lepiej jest zastosować podejście zbliżone do klasycznego algorytmu genetycznego.

Spróbujmy przejść do drugiego pytania – w jakich kierunkach (i czy) powinny być kontynuowane badania? Zgodnie z powszechnie znaną zasadą, że im więcej wiemy, tym więcej wiemy, ile nie wiemy, możemy określić kierunki dalszych badań mających na celu doprowadzenie do powstania w pełni automatycznej, skutecznej metody projektowania sieci neuronowych.

Pierwszy kierunek, jaki się narzuca, to próba **połączenia zalet dwóch podejść do kodowania sieci neuronowych: bezpośredniego i pośredniego**. Można to sobie wyobrazić, że ewoluujemy sieci z kodowaniem pośrednim, kiedy mamy już wystarczająco dobrą sieć, to dekodujemy ją do sposobu bezpośredniego i kontynuujemy ewolucję w przestrzeni sieci zakodowanych bezpośrednio. Oczywiście, nie jest pewne, czy podejście to da dobre efekty. Pamiętajmy, że przestrzeń „jakości” sieci nad ich topologiami jest bardzo zwodnicza. Oznacza to, że nie wiadomo, czy w otoczeniu „dobrej” sieci, którą otrzymaliśmy z ewolucji z kodowaniem pośrednim, będzie się znajdować optymalna lub bliska optymalnej (dla danego zadania) sieć. Ponieważ analitycznie nie jesteśmy w stanie nic wykazać, do weryfikacji zaproponowanego podejścia pozostają tylko badania eksperymentalne.

Drugi kierunek wynika z doświadczenia z siecią XOR. Jak było wspomniane przy omawianiu wyników, algorytm wstecznej propagacji błędu nie był w stanie wyuczyć małej sieci, sieć ta była znaleziona w całości przez algorytm genetyczny. Ale uczenie genetyczne zajmuje sporo czasu. Wcześniejsze doświadczenia wskazywały, że algorytm genetyczny niezbyt dobrze sobie radzi z dostrojeniem wag, a dobre efekty dawało uczenie genetyczne z dostrajaniem metodą wstecznej propagacji błędu. Wydaje się, że ten aspekt wymaga dalszych badań. Wiadomo, że metoda wstecznej propagacji błędu, jak każda metoda gradientowa, znajduje optimum lokalne. Algorytm genetyczny może opuścić optimum lokalne, ale przy tak dużej przestrzeni, jaką jest przestrzeń

wag dla dużej sieci neuronowej, zadanie to jest tak trudne, że uczenie genetyczne staje się mało efektywne. Dla dużych sieci mamy więc sytuację patową: **propagacja błędów** w **optimum lokalnym**, a **algorytm genetyczny jest mało efektywny**. **Może zatem wykorzystać inną, niepopulacyjną metaheurystykę do szukania optymalnego zestawu wag?** Może warto spróbować zastosować symulowane wyżarzanie?

Projektanci sieci neuronowych wiedzą, że sieć neuronowa nie może być ani za duża (uczenie się na pamięć), ani za mała (nie nauczy się dobrze). No i musi być odpowiednio dobrze nauczona – nie może być przeuczona. **Projektując sieć neuronową, mamy dwa kryteria: błąd i rozmiar. Może warto sprawdzić, czy szukanie zbioru optymalnych sieci w sensie Pareto dałoby zadowalające wyniki?**

Kolejne zadanie to projektowanie sieci innych rodzajów, np. *spiking neurons*. Rozpoznanie problemów specyficznych do projektowania innych rodzajów i architektur sieci oraz opracowanie metody ich automatycznego projektowania to duże wyzwanie. Pojawiają się nieliczne prace na ten temat, ale zadowalających rezultatów nie ma.

Mimo iż prace nad ewolucyjnymi metodami automatycznego projektowania sieci neuronowych są prowadzone od wielu lat, to nadal nie mamy w pełni automatycznej, uniwersalnej metody projektowania sieci neuronowych (Abraham 2002, Angeline i in. 1994, Fieldsend, Singh 2005, Garcia-Pedrajas i in. 2005, Ritchie i in. 2007). Niemniej prowadzone prace pozwalają nam lepiej zrozumieć występujące tu problemy, a większa moc komputerów pozwala na wykonywanie badań dla coraz trudniejszych zadań. Wszystko to zbliża nas do opracowania, jeśli nie uniwersalnej i w pełni automatycznej, to pomocnej metody projektowania sieci neuronowych.

LITERATURA

- ABRAHAM A., 2002, *Optimization of evolutionary neural networks using hybrid learning algorithms*. IJCNN'02, Proceedings of the 2002 International Joint Conference on Neural Networks, Vol. 3, IEEE CNF, s. 2797–2802.
- ACHARYA R., KUMAR A., BHAT P.S., LIM C.M., IYENGAR S.S., KANNATHAL N., KRISHNAN S.M., 2004, *Classification of cardiac abnormalities using heart rate signals*. Medical & Biological Engineering & Computing, Vol. 42, s. 288–293, 2004.
- ALMUALLIM H., DIETTERICH T.G., 1991, *Learning with many irrelevant features*. In 9th National Conference on Artificial Intelligence, s. 547–552. MIT Press, 1991. <http://citeseer.nj.nec.com/almuallim91learning.html>.
- ALTENBERG L., 1994, *Evolving better representation through selective genome growth*, Proceedings of the IEEE 1994 World Congress on Computational Intelligence, s. 182–187, 1994.
- ANGELINE, P.J.; SAUNDERS, G.M.; POLLACK, J.B., 1994, *An evolutionary algorithm that constructs recurrent neural networks*. IEEE Trans. on Neural Networks, Vol. 5, Issue 1, s. 54–65, 1994.
- ARABAS J., 2001, *Wykłady z algorytmów ewolucyjnych*. Wydawnictwa Naukowo-Techniczne, Warszawa 2001.
- BALA J., DEJONG K., HUANG J., VAFAIE H., WECHSLER H., *Visual routine for eye detection using hybrid genetic architectures*. In Bolle, R., Dickmanns, E., eds.: Proceedings of the International Conference on Pattern Recognition Vol. 3, Vienna, Austria, IEEE, s. 606–610, 1996. <http://chagall.gmu.edu/publication/ICPR96fr.ps>
- BALAKRISHNAN K., HONAVAR V., 1995, *Evolutionary Design of Neural Architectures: A Preliminary Taxonomy and Guide to Literature*. Technical Report CS TR 95–01, Department of Computer Science, Iowa State University, Ames, Iowa, 1995.
- BALAKRISHNAN K., HONAVAR V., *Properties of genetic representations of neural architectures*. In Proceedings of the World Congress on Neural Networks. s. 807–813, 1995.
- BOERS E., KUIPER H., HAPPEL B., SPRINKHUIZEN-KUYPER I., *Biological metaphors and the design of modular artificial neural networks*. Master's thesis, Department of Computer Science and Experimental and Theoretical Psychology, Leiden University, the Netherlands, 1992.
- BOGUSZEWSKI P., 2003, Wywiady Sygnałów Dnia. Audycja w Programie 1 Polskiego Radia. Światowy Tydzień Mózgu, Rozmawiali: Krzysztof Grzesiowski i Henryk Szrubarz 10.03.2003, <http://www.radio.com.pl/jedynka/sygnały/default.asp?ivID=1193>, maj 2003.
- BONISSONE P., CHEN YO-TO, GOEBEL K., KHEDKAR P.S., *Hybrid Soft Computing Systems: Industrial and Commercial Applications*. (Invited Paper). Proceedings of the IEEE, Vol. 87, No. 9, September 1999, s. 1641–1666.
- BORNHOLDT S., GRAUDENZ D., *General Asymmetric Neural Networks and Structure Desing by Genetic Algorithms*. Neural Networks. Vol. 5, s. 327–334, 1992.
- BULLER A., *Sztuczny mózg to już nie fantazje*, Prószyński i S-ka, Warszawa 1998.
- CANGELOSI A., PARISI D., NOLFI S.: *Cell division and migration in a „genotype” for neural networks*. Network: Computational Neural Systems 5, s. 497–515, 1994.
- CARUANA R., FREITAG D., *Greedy attribute selection*. In Proc. of the 11th International Machine Learning Conference, pages, New Brunswick, NJ, Morgan Kaufmann, s. 28–36, 1994. <http://citeseer.nj.nec.com/caruana94greedy.html>
- CHWIAŁKOWSKA E., *Sztuczna Inteligencja w Systemach Ekspertkich*. NIKOM, Warszawa 1991.
- CICHOSZ P., *Systemy uczące się*. WNT, Warszawa 2000.

- CINICIO S., *Projektowanie sieci neuronowych za pomocą algorytmu genetycznego*. Praca magisterska, Wydział Informatyki i Zarządzania, Politechnika Wrocławska, Wrocław 1998.
- COETZEE F.M., LAWRENCE S., GILES C.L., *Bayesian classification and feature selection from finite data sets*. In Proceedings UAI2000, Stanford, CA, July 2000. UAI. Preprint dostępny (maj 2007): <http://citeseer.nj.nec.com/coetzee00bayesian.html>.
- CRAMER, N. L., *A Representation for the Adaptive Generation of Simple Sequential Programs*. In J. J. Grefenstette, ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, s. 183–187, Erlbaum 1985.
- CRICK F., *Zdumiewająca hipoteza czyli nauka w poszukiwaniu duszy*. Prószyński i S-ka, Warszawa 1997.
- CURRAN K., XUELONG LI, MC CAUGHLEY N., *The Use of Neural Networks in Real-time Face Detection*. Journal of Computer Sciences, Vol. 1, No. 1, s. 47–62, 2005.
- DASGUPTA D., MCGREGOR D.R., *Designing Application-Specific Neural Networks using the Structured Genetic Algorithms*. International Workshop on Combinations of Genetic Algorithms and Neural networks, COGAN-92, IEEE Computer Society Press, 1992.
- DASH M., LIU H., *Feature selection for classification*. International Journal of Intelligent Data Analysis. 1(1997), s. 131–156, 1997, <http://www.public.asu.edu/~huanliu/papers/ida97.ps>.
- DAWKINS R., *Rzeka genów*. Wydawnictwo CIS, Oficyna Wydawnicza MOST, Warszawa 1995.
- DONG M., KOTHARI R., *Feature Subset Selection Using a New Definition of Classifiability*, Computer Science Department, Wayne State University, IBM – India Research Lab., Block I, Indian Institute of Technology, 2003. <http://citeseer.nj.nec.com/559168.html>
- DUCH W., *Jak działa mózg i skąd się bierze umysł?* Praca udostępniona: <http://www.fizyka.umk.pl/~duch>. 2006.
- DUCH W., *Sztuczny mózg? Sztuczna inteligencja?* Praca udostępniona: <http://www.fizyka.umk.pl/~duch/ref/PL/00ai-mozg>. 2006.
- DUCH W., KORBICZ J., RUTKOWSKI L., TADEUSIEWICZ R., *Biocybernetyka i Inżynieria Biomedyczna 2000*. T. 6: *Sieci neuronowe*. Akademicka Oficyna Wydawnicza EXIT, Warszawa 2000.
- DUCH W., GRUDZIŃSKI K., *Sieci Neuronowe i Uczenie Maszynowe: próba integracji*. Katedra metod naukowych, Uniwersytet Mikołaja Kopernika, 2000, <http://www.phys.uni.torun.pl/publications/kmk/00SBM.pdf>.
- DZIK J., *Płeć opłaca się mieć*. Instytut Paleobiologii PAN w Warszawie. W rozmowie z Olgą Woźniak. www.przekroj.pl/index.php?option=com_content&task=view&id=21&Itemid=44, 6/2006.
- EGGENBERGER P., *Creation of Neural Networks Based on Developmental and Evolutionary Principles*. Proc. of the International Conference on ANNs (ICANN 97), Lausanne, Switzerland, October 1997.
- FERREIRA C., *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*. 2002. <http://www.gene-expression-programming.com/gep/Books/>.
- FIELDSEND, J.E.; SINGH, S., *Pareto evolutionary neural networks*. IEEE Trans. on Neural Networks, Vol. 16, Issue 2, s. 338–354, 2005.
- FOGEL D.B., *Evolving Artificial Intelligence*. A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Engineering Sciences, University of California, San Diego 1992.
- FOGEL D.B., *Evolutionary Computation Toward a New Philosophy of Machine Intelligence*. IEEE Press, The Institute of Electrical and Electronics Engineers, Inc., New York 1995.
- FOGEL L.J., *Artificial Intelligence Through Simulated Evolution*. John Wiley, Chichester, UK 1996.
- FORSYTH R., *Expert Systems. Principles and case studies*. Chapman and Hall, Ltd, 1984.
- FREEMAN J.A., SKAPURA D.M., *Neural Networks, Algorithms, Application and Programming Techniques*. Addison-Wesley Publishing Company 1992.
- GALAR R., *Handicapped individua in evolutionary processes*. Biological Cybernetics, Vol. 51, s. 1–9, 1985.
- GALAR R., *Evolutionary Search with Soft Selection*. Biological Cybernetics, Vol. 60, s. 357–364. 1990.
- GALAR R., *Miękka selekcja w losowej adaptacji globalnej w R^n Próba biocybernetycznego ujęcia rozwoju*, Prace Naukowe Instytutu Cybernetyki Technicznej Politechniki Wrocławskiej Nr 84, Wrocław 1990.

- GALAR R., *Simulation of Local Evolutionary Dynamics of small Populations*. Biological Cybernetics, Vol. 65, s. 37–45, 1991.
- GALAR R., *The rate of development of evolutionary processes*. Proceedings of VI International Congress of Cybernetics and systems, Paris, France 1984.
- GALAR R., *Soft Selection in Global Optimization*. XVII Krajowa Konferencja Teoria Obwodów i Sieci Elektronicznych, Wrocław–Polanica Zdrój 1994.
- GALAR R., KWAŚNICKA H., KWAŚNICKI W., *Technological substitution forecasting with a model based on biological analogy*. Technological Forecasting and Social Change, Vol. 23, No. 1, s. 41–58. 1983.
- GARCIA-PEDRAJAS, N.; HERVAS-MARTINEZ, C.; ORTIZ-BOYER, D., *Cooperative coevolution of artificial neural network ensembles for pattern classification*. IEEE Transaction on Evolutionary Computation. Vol. 9, Issue 3, s. 271–302, 2005.
- GLENN F., MANGASARIAN O.L., *A Feature Selection Newton Method for Support Vector Machine Classification*, Computer Sciences Department, UW, Madison, 2002, <http://zSzzSzfzftp.cs.wisc.edu/zSzpubzSzdmizSztech-reportszSz02-03.pdf/a-feature-selection-newton.pdf>.
- GOLDBERG D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc. 1989.
- GOONATILAKE S., KHEBEL S. (Ed), *Intelligent Hybrid Systems*, John Wiley & Sons, 1995.
- GRUAU F., 1994_w, *Cellular Encoding of Neural Networks*. Ecole Normal Supérieure de Lyon, Institut IMAG, Research Report No. 92–91.
- GRUAU F., *Neural Network Synthesis using Cellular Encoding and Genetic Algorithm*. PhD Thesis, L'Ecole Normale Supérieure De Lyon, January 1994.
- GRUAU F., *Neural Network Synthesis Using Cellular Encoding And The Genetic Algorithm*. Université Claude Bernard, Lyon 1994.
- GUPTA G., YADAV R.N., KALRA P.K., JOHN J., *Modeling with Recurrent Neural Networks using Compensatory Neuron Model*. Neural Information Processing – Letters and Reviews, Vol. 6, No.3, 2005.
- HARP S.A., SAMAD T., GUHA A., *Towards the genetic synthesis of Neural Networks*. Proceeding of the third ICGA, Morgan Kaufmann, Inc., San Mateo 1989.
- HEARST M.A., HIRSH H., *AI's greatest trends and controversies*. IEEE Intelligent Systems. January/February 2000, s. 8–17.
- HINTON G.E., *Connectionist Learning Procedures*. Artificial Intelligence, Vol. 40, s. 185–234, 1989.
- HODGKIN, A.L., *The Ionic Basis of Electrical Activity in Nerve and Muscle*. Cambridge Philosophical Society: Biological Reviews, Vol. 26, s. 339–401, 1951.
- HODGKIN A.L., HUXLEY A.F., *Ions through the Membrane of the Giant Axon of Loligo*. Journal of Physiology, Vol. 116, s. 473–496, 1952.
- HOFFMAN A., *Wokół ewolucji*. Biblioteka Myśli Współczesnej, PIW, Warszawa 1983.
- HORNE B.G., HUSH D.R., *Bounds on the complexity of recurrent neural network implementations of finite state machines*. Neural Networks, 9, s. 243–252, 1996.
- HORNIK K., *Approximation Capabilities of Multilayer Feedforward Networks*. Neural Networks, Vol. 4, s. 251–257, 1991.
- HUK M., *Modelowanie zjawiska kierunkowania uwagi przy pomocy sztucznych sieci neuronowych w zadaniu klasyfikacji*. Praca doktorska – Instytut Informatyki Stosowanej, Politechnika Wrocławska 2006.
- ITTI L., KOCH C., *Computational Modeling of Visual Attention*. Nature Reviews Neuroscience, Vol. 2, No. 3, s. 194–203, 2001.
- JĄŁOCHOWSKI K., *Sztuczny mózg z nanorurek*. <http://www.radio.com.pl/nauka/default.asp>. 2006.
- JANKOWSKI N., *Ontogeniczne sieci neuronowe w zastosowaniu do klasyfikacji danych medycznych*. Praca doktorska pod kier. W. Duchy. www.fizyka.umk.pl/publications/kmk/99phd-nj.pdf.
- JANKOWSKI N., *Approximation and Classification in Medicine with IncNet Neural Networks W: Machine Learning and Applications*. Workshop on Machine Learning in Medical Applications, Chania, Greece, s. 53–58, 1999.

- JOHN G.H., KOVAHI R., PFLEGER K., *Irrelevant Features and the Subset Selection Problem*, Computer Science Dept., Stanford University, 1994, <http://citeseer.nj.nec.com/john94irrelevant.html>
- KOLLER D., SAHAMI M., 2006, *Toward Optimal Feature Selection*. Computer Science Dept., Stanford University, 1996, <http://citeseer.nj.nec.com/koller96toward.html>.
- KOMORI M., ABE H., YAMAGUTHI T., *A New Feature Selection Method Based on Dynamic Incremental Extension of Seed Features*. Faculty of Information, Shizuoka University, 2002, http://lpt.unim-b.si/jckbse2002/application/Apl_Login/upcamera/62-jckbse.pdf.
- KON M., PLASKOTA L., *Neural networks, radial basis functions, and complexity*. Statistical Physics Proceedings. Białowieża 1997, s. 322–335.
- KON M., A., PLASKOTA L., *Complexity of Predictive Neural Networks*. Int. Conference on Complex Systems, May 2000.
- KON M., PLASKOTA L., *Information complexity of neural networks*. Neural Networks, Vol. 13, Issue 3, 2000.
- KORBICZ J., OBUCHOWICZ A., UCIŃSKI D., *Sztuczne sieci neuronowe – podstawy i zastosowania*. Akademicka Oficyna Wydawnicza PLJ, Warszawa 1994.
- KOVAHI R., JOHN G.H., *Wrappers for Features Subset Selection*, Data Mining and Visualization Silicon Graphics. Inc., Global Business Intelligent Solutions IMB Almaden Research Center, 1997, <http://citeseer.nj.nec.com/13663.html>.
- KOZA J.R., *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Report No. STAN-CS-90-1314, Stanford University 1990.
- KOZA J.R., *Future Work and Practical Applications of Genetic Programming*, Version 3, June 25, 1996, for „Handbook of Evolutionary Computation”, 1996. Dostępny sieci <http://www-cs-faculty.stanford.edu/~koza>.
- KRAWIEC K., SŁOWINSKI R., SZCZESNIAK I., *Pedagogical Method for Extraction of Symbolic Knowledge from Neural Networks*. Lecture Notes in Computer Science, Vol. 1424, s. 436, 1998.
- KROLL J., *Introduction to Neural Networks and Computational Complexity*, 2003.
- KURZYŃSKI M., *Multistage Bayes approach versus neural network to the computer-aided myocardial infarction diagnosis*. (Hutten H., Krosł P. eds.) In: Advancement of medicine and health care through technology – the challenge to biomedical engineering in Europe, 2nd European Medical & Biological Engineering Conference. EMBEC'02, Vol. 3, s. 762–763, 2002.
- KUSCU I., TRONTON Ch., *Design of Artificial Neural Networks using Genetic Algorithms: review and prospect*. Cognitive and Computing Sciences, 1994.
- KWAŚNICKA H., MICHALAK K., *Correlation-based Feature Selection Strategy in Neural Classification*. ISDA 2006 Proceedings, IEEE Computer Society, Los Alamitos, Washington, Tokyo. Eds. Yuehui Ch., Abraham A. October 2006. Vol. I, s. 741–746.
- KWAŚNICKA H., MICHALAK K., *Correlation-based feature selection strategy in classification problems*. Int. J. Appl. Math. Comput. Sci. Technical University of Zielona Góra., 2006, Vol. 16, No. 4, s. 503–511.
- KWAŚNICKA H., *Neural Network Design with Genetic Algorithm*, Computer Science. Proceedings of the Conference, Ed. by Ivo Vondrak, Jan Stefan, Dep. of Computer Science VSB – Technical University, Ostrava, Czech Republic, s. 386–393, 1995.
- KWAŚNICKA H., *Evolutionary Approach to Artificial Neural Network Design – Good Way or Blind Alley*. MENDEL'97 III International Mendel Conference on Genetic, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets. Technical University of Brno, Faculty of Mechanical Engineering, Czech Republic, June 1997, s. 342–347.
- KWAŚNICKA H., *Evolutionary Approach to Artificial Neural Network Design – Good Way or Blind Alley*. MENDEL'97 III International Mendel Conference on Genetic, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets. Technical University of Brno, Faculty of Mechanical Engineering, Czech Republic, June 1997, s. 342–347.
- KWAŚNICKA H., *Obliczenia ewolucyjne w sztucznej inteligencji*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 1999.

- KWAŚNICKA H., *Sztuczna inteligencja – meandry przeszłości i kierunki dalszego rozwoju*, IV Krajowa Konferencja Naukowa Sztuczna Inteligencja'2000, SzI-15'2000, Materiały Konferencyjne, s. 15–22. Siedlce–Warszawa, 27–28 września 2000.
- KWAŚNICKA H., *Sztuczna inteligencja. Rozwój, Perspektywy, Systemy ekspertowe*. Wyższa Szkoła Zarządzania i Finansów, Wrocław 2005.
- KWAŚNICKA H., KILIŃSKA M., *Application of genetic algorithm to neural networks design*. Proceedings of the 12th Int. Conference on Systems Science, Ed. Z. Bubnicki, Wrocław, Poland 12–15 September 1995, Vol. 1. System Theory. Control Theory, Oficyna Wyd. PWR., s. 164–168, 1995.
- KWAŚNICKA H., MARKOWSKA-KACZMAR U. (red.), *Sieci neuronowe w zastosowaniach*. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2005.
- KWAŚNICKA H.: MICÓR A., *Krótkoterminowe prognozowanie pogody z wykorzystaniem sieci neuronowych*, III Krajowa Konferencja Inżynieria Wiedzy i Systemy Ekspertowe, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 1997, s. 393–400.
- KWAŚNICKA H., ORSKI P., *Genetic Algorithm as an Attributes Selection Tool for Learning Algorithms*. W: Intelligent Information Systems 2004, New Trends in Intelligent Information Processing and Web Mining, Proceedings of the International IIS: IIP WM'04 Conference, Springer, Advances in Soft Computing, s. 449–453, 2004.
- KWAŚNICKA H., PARADOWSKI M., *Selection Pressure and an Efficiency Of Neural Network Architecture Evolving*. W: Seventh International Conference on Artificial Intelligence and Soft Computing, June 7–11, 2004. Lecture Notes in Computer Science 3070, Springer, 2004. s. 444–449, 2004.
- KWAŚNICKA H., PARADOWSKI M., *Efficiency Aspects of Neural Network Architecture Evolution Using Direct and Indirect Encoding*. ICANNGA 2005. Lecture Notes in Computer Science 3070, Springer, 2005, s. 444–449, 2005.
- KWAŚNICKA H., PRZYGODA W., *WOCC – hybrydowy system ekspertowy określający typy charakterologiczne osób*, III Krajowa Konferencja Inżynieria Wiedzy i Systemy Ekspertowe, s. 358–364, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 1997.
- KWAŚNICKA H., SZERSZON P., *NetGen – Evolutionary Algorithms in Designing Artificial Neural Networks*. Proceedings of the Third Conference Neural Networks and Their Applications and Summer School on Neural Networks Applications to Signal Processing, Kule, Polskie Towarzystwo SN, 1997, s. 671–676.
- KWAŚNICKA H., SZPUNAR-HUK E., *Genetic Programming in Data Modelling*, w: Genetic Systems Programming. Theory and Experiences, Springer, Series: Studies in Computational Intelligence, Vol. 13 Nedjah N., Abraham A. Mourelle L. de Macedo (ed.), ISBN: 3-540-29849-5, rozdz. 5, s. 105–130, 2006.
- KWAŚNICKA H., WOŹNIAK P., *EMOT – Evolutionary Approach to 3D Computer Animation*. Proceedings of the International Multiconference on Computer Science and Information Technology, Vol. 1 (CD ISSN 1896–7094) XXII Autumn Meeting of Polish Information-Processing Society. November 6–10, 2006. Wisła, Poland 2006.
- KWOK, T., YEUNG, D., *Experimental Analysis of Input Weight Freezing in Constructive Neural Networks*. Proc. IEEE Int. Conf. on Neural Networks, 1993, s. 511–516.
- LANGLEY P., *Selection of Relevant Features in Machine Learning*, Pat Langley, Institute for the Study of Learning and Expertise, 1994, <http://citeseer.nj.nec.com/langley94selection.html>.
- LIESKE B., *Automatyczna optymalizacja sztucznych sieci neuronowych za pomocą algorytmów genetycznych*. Praca magisterska, Wydział Informatyki i Zarządzania, Politechnika Wrocławska, 1995.
- LIU H., MOTODA H., DASH M., *A Monotonic Measure for Optimal Feature Selection*. Dept. of Info. Sys. and Comp. Science, National Univ. of Singapore, Division of Intelligent System Science, Osaka University, Bioinformatics Center, National Univ. of Singapore, 1998, <http://citeseer.nj.nec.com/liu98monotonic.html>.
- MARCISZEWSKI W., *Sztuczna inteligencja*. Znak, 1998.
- MARKOWSKA-KACZMAR U., *Ekstrakcja reguł z sieci neuronowych. Podejście ewolucyjne*. Wrocław, Oficyna Wyd. PWR. 2006.

- MARKOWSKA-KACZMAR U., KWAŚNICKA H. (red.), *Sieci neuronowe w zastosowaniach*. Oficyna Wyd. PWr. Wrocław. 2005.
- MCCULLOCH W.S., PITTS W.H., *An Logical Calculus of the Ideas Immanent in Nervous Activity*. Bulletin of Mathematics and Biophysics, Vol. 5, s. 115, 1943.
- MCKENNA S.J., RICKETTS I.M., CAIRNS A.Y., HUSSEIN K.A., *Cascade-Correlation Neural Networks for the Classification of Cervical Cells*. IEE Colloquium on Neural Networks for Image Processing Applications, s. 5/1–5/4, 1992.
- MEDSKER L.R., *Hybrid Intelligent Systems*. Kluwer Academic Publishers, Boston–Dordrecht–London, 1995.
- MEL B. W., KOCH C., *Sigma-pi learning: on radial basis functions and cortical associative learning*. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, Vol. 2, s. 474–81. Morgan Kaufmann, 1990.
- MICHALEWICZ Z., *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. WNT, Warszawa 1996.
- MILLER G.F., TODD P.M., HEDGE S.U., *Designing Neural Networks using Genetic Algorithms*. Proceeding of the third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Inc., San Mateo, California 1989.
- MITCHELL M., *Introduction to Genetic Algorithms*. MIT Press, 1996.
- MITCHELL T.M., *Machine Learning*. The McGraw-Hill Companies, Inc., 1997.
- MOHRAZ K., PROTZE P., *FlexNet A Flexible Neural Network Construction Algorithm*. Proceedings of the 4th European Symposium on Artificial Neural Networks (ESANN '96), s. 111–116, Brussels 1996.
- MONTANA J.D., DAVIS L., *Training Feedforward Neural Networks Using Genetic Algorithms*. BBN Systems and Technologies Corp., Cambridge 1989.
- MURRAY D., *Tuning Neural Networks with Genetic Algorithms*. AI Expert 1994.
- NADEL L., STEIN, D. (Ed.), *Lectures in Complex Systems*. Lecture volume V, Santa Fe Institute, Addison-Wesley Publishing Company, 1993.
- NERI F., ZINI F., *Learning Disjunctive Concepts with Distributed Genetic Algorithms*. In Proc. of AIIA congress, Volume on „Apprendimento Automatico”, Parma, Italy, 1994. http://www.dai.ed.ac.uk/groups/evalg/eag_local_copies_of_papers.body. Html.
- NOWOSTAWSKI M., KWAŚNICKA H., *The Search Engine for Information Systems Based on Parallel Genetic Algorithm*. Proceedings of the 2nd International Conference on Parallel Processing & Applied Mathematics, s. 442–451, PCZ IMI, Częstochowa 1997.
- OLSHAUSEN B., ANDERSON C., VAN ESSEN D., *A Neurobiological Model of Visual Attention and Invariant Pattern Recognition based on Dynamic Routing of Information*. J. Neurosci., Vol. 13, s. 4700–4719, 1993.
- ORPONEN P., *Neural networks and complexity theory*. In Proc. 17th Symposium on Mathematical Foundations of Computer Science, Springer-Verlag Lecture Notes in Computer Science, Vol. 629 (1992), s. 50–61.
- ORPONEN P., *Computational complexity of neural networks*. A survey. Nordic Journal of Computing 1 (1994), s. 94–110.
- OSSOWSKI S., *Sieci neuronowe w ujęciu algorytmicznym*, WNT, Warszawa 1997.
- PARADOWSKI M., *Ewolucja topologii sieci neuronowych z wykorzystaniem algorytmów genetycznych*. Praca magisterska, Politechnika Wrocławska 2004.
- PEARLMUTTER B.A., *Gradient Calculations for Dynamic Recurrent Neural Networks*. A Survey. Draft of July 20, 1995 for IEEE Transactions on Neural Networks.
- PEKALSKA E., HAROL A., LAI C. DUIN R.P.W., *Pairwise selection of features and prototypes*. Computer Recognition Systems (Proc. of 4th Int. Conf. on Computer Recognition Systems CORES'05), 2005.
- PELC T., *Model formalny sieci neuronowej do reprezentacji semantyki zdań języka naturalnego na potrzeby automatycznego programowania*. Praca doktorska, Wydawnictwo Politechniki Wrocławskiej, Wrocław 1998.

- PENROSE R., *Nowy umysł cesarza. O komputerach, umyśle i prawach fizyki*. PWN, Warszawa, 1996.
- PERCHELT L., Proben1 – A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical Report 21/94.
- PORTO V. W., FOGEL D. B., FOGEL L. J., *Alternative Neural Network Training Methods*. IEEE EXPERT, June 1995, s. 16–21, 1995.
- PRUSINKIEWICZ P., LINDENMAYER A., *The Algorithmic Beauty of Plants*. Springer-Verlag, New York 1990.
- RADCLIFFE N.J., SURRY P.D., *Cooperation through Hierarchical Competition in Genetic Data Mining*. Edinburgh Parallel Computing Centre, Scotland. www.dai.ed.ac.uk/groups/evalg/eag_local_copies_of_papers/body.html.
- RAMAN B., IOERGER T.R., *Enhancing Learning using Feature and Example selection*. Department of Computer Science, Texas A&M University, 2003, <http://citeseer.nj.nec.com/raman03enhancing.html>.
- REILLY R.G., SHARKEY N.E., *Connectionist Approaches to Natural Language Processing*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1992.
- RITCHIE, MARYLYN D.; MOTSINGER, ALISON A.; BUSH, WILLIAM S.; COFFEY, CHRISTOPHER S.; MOORE, JASON H., *Genetic programming neural networks: A powerful bioinformatics tool for human genetics*. Applied Soft Computing Journal, Vol. 7, Issue 1, s. 471–479, 2007.
- ROSENBLATT F., *The perceptron: a probabilistic model for information storage and organization in the brain*. Psychological Review, Vol. 65, s. 386–408, 1958.
- RUSSEL S.J., NORVIG P., *Artificial Intelligence. A Modern Approach*. Prentice Hall, Inc., 1995.
- RUTKOWSKA M., PILIŃSKI L., RUTKOWSKI L., *Sieci neuronowe, algorytmy genetyczne, systemy rozmyte*. WN PWN, Warszawa 1997.
- SCHAFER J. D., WHITLEY D., ESHELMAN L. J., *Combinations of Genetic Algorithms and Neural Networks: A Survey of the State* International Workshop on CGA&NN, Baltimore, Maryland, 1992.
- SCHALKOFF R.J., *Artificial Intelligence: An Engineering Approach*. McGraw-Hill Publishing Comp., 1990.
- SCHMITT M., *On the Complexity of Computing and Learning with Multiplicative Neural Networks*, Neural Computation, Vol. 14, No. 2, s. 241–301, 2002.
- SCOT A., *Schody do umysłu. Nowa kontrowersyjna wiedza o świadomości*. WNT, Warszawa 1999.
- SEARLE J.R., *Umysł, mózg i nauka*. Wydawnictwo Naukowe PWN, Warszawa, 1995, skrót w: Świat Nauki, Nr 1, 1999.
- SHIN Y., GHOSH J., *Approximation of Multivariate Functions using Ridge Polynomial Networks*. Proc. IJCNN, Baltimore, s. 380–385, 1992.
- SIMA J., ORPONEN P., *Computational Taxonomy and Survey of Neural Network Models*. Neural Computation, 2001.
- SPRATLING M.W., HAYES G., *Learning Synaptic Clusters for Nonlinear Dendritic Processing*. Neural Processing Letters, Vol. 11, No. 1, s. 17–27, 2000.
- SOUCEK B. (ed.), *Neural and Intelligent Systems Integration: Fifth and Sixth Generation Integrated Reasoning Information Systems*. Series: Sixth Generation Computer Technologies, John Wiley & Sons, Inc. NY, Toronto, Singapore 1991.
- SUN R., *Hybrid connectionist-symbolic models: A report from the IJCAI95 workshop on connectionist-symbolic integration*. Artificial Intelligence Magazine, 1996.
- SUN Z., BEBIS G., YUAN X., LOUIS S.J., *Genetic Feature Subset Selection for Gender Classification: A Comparison Study*. Computer Vision Lab. Department of Computer Science, University of Nevada, Genetic Adaptive Systems Lab. Department of computer science, University of Nevada, 2002, <http://www.cs.unr.edu/~bebis/genderWACV02.pdf>.
- SUSAC D., *Crystal Balls 3: Coming in 30 Years to a Theatre as Near as You Think – Superhuman Cyborgs and Androids*, March 1, 1998. <http://ai.about.com/compute/ai/library/weekly/aa030198a.htm>.
- SZYMBORSKI K., *Ewolucja inteligencji*. <http://www.racjonalista.pl/kk.php/s,3663>. 2006.
- TADEUSIEWICZ R., *Sieci neuronowe*, Akademicka Oficyna Wydawnicza RM, 1993.
- TADEUSIEWICZ R., *Do zbudowania sztucznego mózgu droga daleka*. Nauka w Polsce, PAP. <http://www.naukawpolsce.pap.pl/nauka>. 30. 09. 2005.

- TADEUSIEWICZ R., *Czy możliwe jest zbudowanie sztucznego mózgu?*
http://www.if-pan.krakow.pl/ptp/tydzmozgu05_pliki/200305.htm. 2005.
- TREADGOLD N.K., GEDEON T.D., *Extending and Benchmarking the CasPer Algorithm*. A1'97, Perth Australia, 1997, s. 398–406.
- TREADGOLD N.K., GEDEON T.D., *Extending CasPer: A Regression Survey*. Int. Conf on Neural Information Processing, 1997, s. 310–313.
- TREADGOLD N.K., GEDEON T.D., *Exploring Architecture Variations in Constructive Cascade Networks*. Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The IEEE International Joint Conference on Vol. 1, s. 343–348, 1998.
- TSOTSOS J.K., CULHANE S., CUTZU F., *From foundational principles to a hierarchical selection circuit for attention*. In: Visual Attention and Cortical Circuits, Braun J, Koch C, Davis J. (eds.), Cambridge, MA, MIT Press, s. 285–306, 2001.
- VAARIO J., *Modeling Adaptive Self-Organization*, in Brooks and Maes. (eds.) Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems. s. 313–318, 1994.
- VAFAlE H., DE JONG K., *Genetic Algorithms as a Tool for Feature Selection in Machine Learning*. Center for Artificial Intelligence George Mason University, 1992, <http://cs.gmu.edu/research/gag/papers/TAI92.ps>
- VAFAlE H., DE JONG K., *Robust Feature Selection Algorithm*. Center for Artificial Intelligence George Mason University, 1993,
http://www.chem.purdue.edu/bendor/groupmem/Dongmao/List_of_reference/featureSelection/vafalie93robust.pdf
- VAFAlE H., DE JONG K., *Improving A Rule Induction System Using Genetic Algorithms*, George Mason University, 1994, <http://citeseer.nj.nec.com/vafalie94improving.html>
- VAFAlE H., DE JONG K., *Genetic Algorithms as a Tool for Restructuring Feature Space Representations*. Center for Artificial Intelligence George Mason University, <http://cs.gmu.edu/research/gag/papers/TAI95.ps>
- VAFAlE H., DE JONG K., *Genetic Algorithms as a Tool for Restructuring Feature Space Representations*. Computer Science Department, George Mason University, 1995. www.cs.gmu.edu:80/research/gag/pubs.html
- VON NEUMANN J., *The Computer and the brain*. New Haven, Yale University Press, 1958.
- WALTZ D.L., *Artificial Intelligence: Realizing the Ultimate Promises of Computing*. AI Magazine 18(3): s. 49–52. <http://www.cs.washington.edu/homes/lazowska/cra/ai.html>.
- WEISS G. (ed.), *Multiagent Systems*. A Modern Approach to Distributed Artificial Intelligence. The MIT Press, 1999.
- WERMTER S., SUN R., *Introduction*, in: *Hybrid Neural Systems*, Springer, Heidelberg, New York 2000.
- WHITLEY D., *Genetic Algorithms and Neural Networks*, Generic Algorithms in Engineering and Computer Science, 1995.
- WILLIAMS R.J., PENG J., *An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories*. Neural Computations 2, s. 490–501, 1990.
- WINTER C., JEFFERY C.S., DRILLING J.S., *Automatic Classification of Subdwarf Spectra using a Neural Network*. Astronomical Society of the Pacific Conference Series, Vol. 291, s. 375–378, Kluwer Academic Publishers 2004.
- YAO X., *Evolving Artificial Neural Networks*. School of Computer Science, The University of Birmingham, 1999.
- ŻURADA J., BARSKI M., JĘDRUCH W., *Sztuczne sieci neuronowe*, PWN, Warszawa 1996.

Źródła dostępne tylko w postaci dokumentów umieszczonych w internecie, dostępne na przełomie roku 2006 i 2007 – strony naukowców i laboratoriów naukowych

(Ailab 1997w) <ftp://ftp.ifi.unizh.ch/pub/institute/ailab/techreports/97.21.ps.gz>.
 (CasCor_w) Cascade Correlation Neural Networks. <http://www.dtreng.com/mlfn.htm>.

- (Earthlink_w) http://home.earthlink.net/~paulcarr/cool_tech/JHUECfinalreport.html.
- (Vaario_w) http://mi-2.mech.kobe-u.ac.jp/~jari/Publications/vaario_alifeIV.ps.gz.
- (Gruau 1994_w) <http://www.cwi.nl/~gruau/gruau/PhD94-01-E.ps.Z>.
- (Gruau 1992_w) <http://www.cwi.nl/~gruau/gruau/RR92-21.ps.Z>.
- (Duch_w) <http://www.phys.uni.torun.pl/~duch/Wyklady/kog-m/mozg-caly.jpg>
- (Brooks_w) BROOKS R.A., BREAZEL C., MARJANOVIĆ M., SCASSELLATI B., WILLIAMSON M.: The Cog Project: Building a Humanoid Robot, <http://www.ai.mit.edu/projects/cog>.
- (Evidence_w) Evidence of brain evolution. strona [www](http://www.brainmuseum.org/evolution/evidence.html): <http://www.brainmuseum.org/evolution/evidence.html>.
- (Far_w) FAR B.H.: Department of Information and Computer Sc., Saitama Univ., <http://www.cit.ics.saitama-u.ac.jp/~far/Lectures/KE2/01kaime/sld010.htm>.
- (Holloway_w) HOLLOWAY R.: Evolution of the Human Brain. <http://www.massey.ac.nz/~alock/hbook/brain.htm>.
- <http://citd.scar.utoronto.ca/ANT3032/Henderson/Table.htm>.
- (Kasperski_wa) KASPERSKI M.: Mózg. [kognitywistyka.net](http://www.kognitywistyka.net) Wortal wiedzy o umyśle. <http://www.kognitywistyka.net/mozg/mozg.htm>.
- (Kasperski_wb) KASPERSKI M.: *Rozwój osobniczy mózgu – wstęp. Mózg – budowa*. 2005. Notatki na stronie: http://www.kognitywistyka.net/mozg/historia_badan.htm.
- KROLL J., *Introduction to Neural Networks and Computational Complexity*. 2003. <http://www.eecs.tufts.edu/~jkroll/neuralcomp.html>.





Algorytmy ewolucyjne, sieci neuronowe i sztuczne systemy immunologiczne to przykłady efektywnego naśladowania natury. Każde z tych podejść ma swoje mocne i słabe strony, ma swoich zwolenników i przeciwników.

Książka dotyczy dwóch metod inspirowanych naturą uproszczonego modelu ewolucji biologicznej i mózgu, czyli algorytmów ewolucyjnych i sztucznych sieci neuronowych. Pokazano, jak można wykorzystywać algorytmy ewolucyjne dla wsparcia projektowania sieci neuronowej dla określonego zadania. Zawarte w książce przemyslenia, wyniki badań i wnioski są efektem wielu lat pracy. Ponieważ problem jest bardzo złożony i trudny, autorka uznała, że warto podzielić się swoim doświadczeniem.

W pracy skupiono się głównie na projektowaniu jednokierunkowych sieci neuronowych, choć część eksperymentów dotyczyła sieci ze sprzężeniem zwrotnym. Doświadczenie wskazuje, że ogólne wnioski wynikające z badań nad genetycznym projektowaniem jednokierunkowych sieci neuronowych są prawdziwe również w odniesieniu do genetycznego projektowania innych rodzajów sieci.



Wydawnictwa Politechniki Wrocławskiej
są do nabycia w księgarni
„Tech”
plac Grunwaldzki 13, 50-377 Wrocław
budynek D-1 PWr., tel. 071 320 29 35
Prowadzimy sprzedaż wysyłkową

ISBN 978-83-7493-353-7