UNIVERSITATEA DIN BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

Lucrare de licență
Aplicații ale rețelelor neuro-fuzzy în estimarea costului dezvoltării software

Coordonator științific,
Lect. dr. Șuter Florentina

Student,
Madar Nicușor-Florin

București
2017

Cuprins

1 Introducere

Prin lucrarea de față dorim să prezentăm un studiu asupra rețelelor neuro-fuzzy aplicate peste modelul COCOMO (*Constructive Cost Model*), astfel încât să se poată obține o estimare a dezvoltării unui proiect software plecând de la factorii de cost și dimensiunea sa. Metoda prezentată este centrată pe rețelele ANFIS (*Adaptive neuro-fuzzy inference system*), un tip de rețele neuronale relativ nou, articolul în care au fost definite fiind publicat în 1993.

Fără a intra in detalii deocamdată, rețelele neuro-fuzzy, datorită faptului că sunt alcătuite dintr-o rețea neuronală și un sistem de inferență fuzzy, oferă atât puterea de învățare a rețelelor neuronale clasice, cât și flexibilitatea logicii fuzzy de a lucra cu date incerte [2]. Deoarece, în mod inerent, orice estimare este supusă subiectivității, și nici modelul COCOMO nu oferă o metodă riguroasă pentru alegerea factorilor de cost, este de înțeles că o predicție de cost pleacă de la premisa că datele de intrare sunt *subiective*, i.e. au un oarecare grad de incertitudine. Astfel, problema de față devine un candidat bun pentru o rețea neuro-fuzzy.

Modelul COCOMO datează din 1981 în cartea lui Barry W. Boehm *"Software Engineering Economics"* [1], fiind un model pentru estimarea efortului, costului si gestionării timpului de dezvoltare al proiectelor software. Acesta este o combinație de trei modele cu niveluri diferite de detalii și complexitate:

- Basic: oferă o estimare rapidă; se pretează în stagiile precare ale dezvoltării
- Intermediate: estimările devin mai precise, sunt necesare caracteristici ale proiectului și un stagiu mai matur de dezvoltare
- Advanced: cel mai detaliat dintre toate trei, necesită cea mai multă informație

Lucrarea tratează nivelul *Intermediate* al COCOMO, acesta oferind un echilibru bun între datele de intrare (15 factori de cost și o estimare numerică a dimensiunii proiectului în linii de cod sursă) și acuratețea estimării efortului în *man-months*.

În afara secțiunilor de introducere, lucrarea este împărțită în două părți. În prima parte vom prezenta aspectele teoretice pentru a fundamenta și analiza tehnicile folosite în construirea și antrenarea unei rețele neuro-fuzzy peste modelul COCOMO. În cea de-a doua parte lucrarea va prezenta rezultatele obținute, făcând și o comparație sumară cu rezultatele ce au fost obținute deja în domeniu.

Lucrarea se încheie printr-o comparație cu o rețea neuronală clasică, pentru a expune diferențele față de un model convențional și bine stabilit în acest domeniu pentru problemele de *fitting*.

2 Prezentarea Domeniului

Există o mulțime de articole scrise atât pe tema estimării costurilor de dezvoltare folosind COCOMO, cât și pentru estimarea acestora folosind tehnici neuro-fuzzy. Vom prezenta în continuare, în mod sumar, două articole pentru a oferi lucrării de față un context și o încadrare în domeniu.

Primul articol pe care îl vom prezenta este "COCOMO Estimates Using Neural Networks" [3], o lucrare ce prezintă rezultatele experimentale ale estimării modelului COCOMO folosind o rețea neuronală cu trei straturi. Kaushik et al. prezintă o metodologie de învățare pe setul de date COCOMO disponibil public, evaluând performanța arhitecturii folosind criteriul de evaluare MRE (*Magnitude of Relative Error*). Autorii menționează că urmăresc obținerea unei rețele cu un număr cât mai mic de straturi de perceptroni și de noduri, astfel încât performanțele rețelei să fie cât mai bune, dar complexitatea acesteia, și, implicit, timpul necesar pentru antrenare să fie minimizat.

Având ponderile $W_i, i = \overline{1,17}$ inițializate cu 1, rata de învățare $\alpha = 0.001$ și biasul $b = 1$, algoritmul propus este prezentat în șapte pași:

1. Execută pașii 2-7 până când condiția de oprire este falsă.
2. Execută pașii 3-6 pentru fiecare pereche de antrenare.
3. Stratul de intrare primește semnalul de intrare și îl trimite către stratul ascuns prin aplicarea funcțiilor identitate de activare pentru toate nodurile de intrare $v_i, i = \overline{1,17}$.
4. Fiecare nod ascuns își însumează semnalele de intrare ponderate pentru a calcula ieșirea rețelei după formula:

$$y_{inj} = b_j + \sum_{i=1}^{17} v_i W_{ij} \tag{2.1}$$

   Funcția de activare a perceptronului $f(y_{in})$ este aplicată pentru fiecare nod, unde

$$f(y_{in}) = \begin{cases} 1 & \text{dacă } y_{in} > \theta \\ 0 & \text{dacă } -\theta \leq y_{in} \leq \theta \\ -1 & \text{dacă } y_{in} < -\theta \end{cases} \tag{2.2}$$

   este funcția de activare, cu θ fiind o valoare de prag.
5. Calculează ieșirile (i.e. efortul) pe stratul de ieșire folosind aceeași procedură ca în pasul 4 considerând toate ponderile pentru $j = \overline{1,5}$ ca fiind 1.
6. Compară efortul real cu cel estimat; dacă diferența este în limitele acceptate atunci rezultatul este acceptat, altfel ponderile sunt actualizate după:

$$W_i(nou) = W_i(vechi) + \alpha \cdot intrare(i) \tag{2.3}$$

7. Verifică dacă rezultatele se supun condiției de operire i.e. dacă nu există nicio schimbare în ponderi atunci oprește procesul de antrenare, altfel repornește de la pasul 2.

Folosind metoda prezentată, autorii încheie articolul ajungând la concluzia că o rețea neuronală oferă o acuratețe acceptabilă pentru predicție asupra datelor din setul de date COCOMO '81., încercând să îl modeleze folosind o rețea cu un număr de noduri cât mai mic, pentru a păstra performanța.

Al doilea articol pe care îl vom prezenta este "Software Cost Estimation using Adaptive Neuro Fuzzy Inference System" [4], o lucrare ce folosește un model de rețele neuro-fuzzy pentru a prezenta rezultate experimentale ale estimării modelului COCOMO. Autorii prezintâ o metodă ce implică normalizarea datelor de intrare înainte ca acestea să fie prezentate rețelei, ei sugerând normalizarea min-max. Normalizarea min-max este definită ca:

$$x' = \frac{x - min_x}{max_x - min_x}(max\_nou_x - min\_nou_x) + min\_nou_x \tag{2.4}$$

Astfel, ei obțin un set de date încadrat strict în intervalul $[-1, 1]$. Metodologia lor de lucru a fost de a crea o rețea ANFIS folosind doar trei caracteristici pentru intrare, și având o singură ieșire, estimarea de efort. Justificarea autorilor de a folosi doar trei intrări nu este clară, ei enunțând că numărul maxim de parametri pentru o rețea ANFIS trebuie sa fie 3. În continuare, ei consideră că una dintre cele mai importante caracteristici este numărul de linii de cod, astfel încât le rămân de determinat ultimele două intrări pentru rețea, folosind o abordare prin încercări repetate până când au obținut cea mai mică eroare de antrenare peste setul de date restrâns la cele trei caracteristici. Autorii concluzionează această parte prin găsirea celor trei caracteristici care oferă cea mai mică eroare de antrenare: **TURN, VIRT** și **LOC**.

Arhitectura rețelei propuse este de a folosi câte treizeci de funcții de apartenență pentru fiecare nod de intrare, treizeci de clustere și treizeci de funcții de apartenență pentru ieșire folosind algoritmul de clustering FCM.

Lucrarea este concluzionată de către rezultate, care au o acuratețe bună, în ciuda faptului că s-au folosit doar trei caracteristici din totalul de șasesprezece pe care le oferă setul de date per proiect. Astfel, noi am considerat că rețelele neuro-fuzzy sunt o direcție bună de cercetare pentru a obține estimări cat mai bune în ceea ce privește estimarea costului de dezvoltare software.

3 Mediul de programare

Pentru implementarea modelelor predictive ce vor fi prezentate în această lucrare, mediul de dezvoltare este Matlab.

Matlab este un limbaj de programare interpretat multi-paradigmă, centrat în jurul manipulării matricelor, cu tipuri de date slabe (variabilele sunt convertite implicit) și inferate (variabilele pot avea valori atribuite fara ca tipul lor să fie declarat). În plus, cei de la MathWorks oferă o mulțime de biblioteci (denumite *toolboxuri*) pentru diverse aplicații în matematică numerică și simbolică, procesare de semnale și imagini, sisteme de control etc. De asemenea, există suport pentru exportarea codului în limbaje de programare tradiționale precum C, Java sau Python, dar și pentru apelarea de funcții scrise în aceste limbaje folosind un *wrapper* care permite tipurilor de date din MATLAB sâ fie pasate și primite în mod dinamic. Prin urmare, am considerat că Matlab oferă uneltele necesare pentru a dezvolta o soluție orientată pe rezultate, și nu pe detaliile de implementare la nivel de limbaj de programare.

Aplicațiile lucrării s-au bazat pe toolboxurile *Fuzzy Logic Toolbox* și *Neural Network Toolbox*. Primul toolbox ne-a ajutat să generăm un sistem de inferență fuzzy de tip Sugeno, să clusterizăm datele de intrare și să putem construi o rețea ANFIS. Cel de-al doilea ne-a oferit posibilitatea de a construi și antrena o rețea neuronală folosită drept comparație pentru modelul neuro-fuzzy propus.

4 Seturi de date

4.1 Descriere

Setul de date folosit în această lucrare a fost compus din concatenarea a două seturi de date [5, 6]. Acestea sunt o reprezentare a mai multor proiecte în formatul propus de către modelul COCOMO [1], mai specific de forma sa COCOMO '81.

Modelul COCOMO este un model de estimare a costului de dezvoltare software procedural, parametrii săi fiind derivați dintr-o formulă de regresie aplicată peste date din proiecte istorice (un număr de 63 pentru COCOMO '81). Aceste date au fost extrase de la compania TRW Aerospace, unde Barry W. Boehm, autorul modelului, a ocupat poziția de director de cercetare în software și tehnologie. Studiul său a inclus proiecte cu dimensiuni între 2000 și 100000 de linii de cod, scrise în diverse limbaje de programare și dezvoltate folosind metodologia *waterfall*, care era cea mai prevalentă la nivelul anilor 80.

Primul set de date cuprinde 60 de proiecte dezvoltate în mai multe centre ale NASA din perioada anilor 1980 si 1990, având toate cele 17 atribute ale modelului. Al doilea set de date este constituit din 63 de de proiecte, fiind chiar datele după care s-a efectuat calibrarea modelului COCOMO.

4.2 Caracteristicile setului de date

Caracteristică	Very Low	Low	Nominal	High	Very High	Extra High
Fiabilitatea softului necesară	0.75	0.88	1.00	1.15	1.40	
Dimensiunea bazei de date		0.94	1.00	1.08	1.16	
Complexitatea produslui	0.70	0.85	1.00	1.15	1.30	1.65
Constrângeri de performanță la rulare			1.00	1.11	1.30	1.66
Constrâgeri de memorie			1.00	1.06	1.21	1.56
Volatilitatea mediului de mașini virtuale		0.87	1.00	1.15	1.30	
Timpul necesar pentru schimbări		0.87	1.00	1.07	1.15	
Capabilitatea analiștilor	1.46	1.19	1.00	0.86	0.71	
Experien;tâ în aplicații	1.29	1.13	1.00	0.91	0.82	
Capabilitatea inginerilor soft	1.42	1.17	1.00	0.86	0.70	
Experiență cu mașinile virtuale	1.21	1.10	1.00	0.90		
Experiență cu limbajul de programare	1.14	1.07	1.00	0.95		
Aplicarea metodelor de inginerie soft	1.24	1.10	1.00	0.91	0.82	
Folosirea uneltelor software	1.24	1.10	1.00	0.91	0.83	
Strictețea planificârii	1.23	1.08	1.00	1.04	1.10	

Tabela 4.1: Valorile posibile pentru multiplicatorii de efort.

Fiecare din cele 15 caracteristici prezente in tabelul de mai sus primesc un multiplicator de efort corespunzător cu încadrarea lor in tabel. Produsul tuturor

multiplicatorilor de efort se numește factor de ajustare a efortului (EAF); valorile tipice se află între 0.9 și 1.4.

Pornind de la aceste valori, formula modelului intermediar este exprimată prin:

$$E = a_i \cdot (KLoC)^{b_i} \cdot (EAF) \tag{4.1}$$

unde E este efortul aplicat in luni-persoană, KLoC este numărul estimat de mii de linii de cod livrate în proiect, iar EAF este factorul de ajustare a efortului. Tabela 4.2 oferă valorile pentru a_i și b_i.

Proiect software	a_i	b_i
Organic	3.2	1.05
Semi-detașat	3.0	1.12
Încorporat	2.8	1.20

Tabela 4.2: Valorile posibile pentru coeficientul a_i și exponentul b_i.

Cele trei clase de proiecte regăsite în tabel sunt explicate astfel:

- Proiect organic: echipe "mici" cu experiență "bună" ce lucrează cu cerințe "mai puțin rigide"
- Proiect semi-detașat: echipe "medii" ca dimensiune, cu experiență mixtă ce lucrează cu cerințe mixte din punct de vedere al rigidității
- Proiect încorporat: proiecte dezvoltate cu constrângeri "tari". Acestea sunt o combinație de proiecte organice și semi-organice.

Urmând ordinea din tabela 4.1 enunțăm în continuare abrevierile multiplicatorilor de efort. Acestea sunt: **RELY, DATA, CPLX, TIME, STOR, VIRT, TURN, ACAP, AEXP, PCAP, VEXP, LEXP, MODP, TOOL** și **SCED**.

Din tabela 4.1 putem deduce că multiplicatorii pot fi împărțiți în două categorii:

- Cre;sterea valorii implică scăderea efortului: ACAP, PCAP, AEXP, MODP, TOOL, VEXP și LEXP.
- Scăderea valorii implică creșterea efortului: STOR, DATA, TIME, TURN, VIRT, CPLX și RELY.

Multiplicatorul SCED este unic: el nu poate fi exprimat într-un mod "monoton" i.e. valorile numerice din jurul punctului nominal cresc atât atunci când factorii tind spre "Very Low" cât și atunci cand aceștia tind spre "Extra High" astfel încât corelarea cu efortul devine în forma de "U", deci, a le oferi programatorilor prea mult sau prea puțin timp pentru dezvoltare este dăunător pentru întreg sistemul.

5 Logică fuzzy

5.1 Preliminarii

În contrast cu mulțimile clasice (*crisp*), mulțimile fuzzy tratează apartenența la o mulțime printr-o funcție, numită funcție de apartenență, ce indică gradul de apartenență al unui element în raport cu mulțimea. În timp ce mulțimile clasice fac o distincție clară, folosind funcția caracteristică, între elementele ce sunt în respectiva mulțime și cele care nu, mulțimile fuzzy vor indica apartenența, de obicei printr-un grad din intervalul $[0, 1]$, în funcție de cât de mult elementul respectiv îndeplinește proprietățile mulțimii. Formal, fie $A = a_1, a_2, ..., a_n$ o mulțime. Funcția caracteristică a mulțimii a este:

$$\chi_A : A \to \{0, 1\},\ \chi(x) = \begin{cases} 1 & \text{dacă } x \in A \\ 0 & \text{dacă } x \notin A \end{cases} \tag{5.1}$$

În cazul logicii fuzzy, funcția de apartenență este definită ca:

$$\mu_A : X \to [0, 1] \tag{5.2}$$

Unde $\mu(x)$ va fi definitâ în funcție de particularitățile apartenenței la mulțimea X. Deci, observăm faptul că acum există o metodă de a obține gradele de apartenență datorită codomeniului fiind un interval, nu o mulțime discretă. Klir et al. [7] menționează că importanța variabilelor fuzzy constă în faptul că ele facilitează tranzițiile graduale între stări, și, prin urmare, posedă o capabilitate inerentă de a exprima și de a lucra cu incertitudini de observație și de măsurare, în contrast cu variabilele crisp, ce nu au această putere. Mai mult, deși variabilele crisp sunt definite corect din punct de vedere matematic, acestea nu pot fi realiste atunci când modelează situații din lumea reală, unde erorile de măsurare sunt imposibil de evitat.

Putem defini în continuare următoarele operații asupra unei mulțimi fuzzy:

$$\bar{A}(x) = 1 - A(x) \tag{5.3}$$

$$(A \cap B)(x) = min[A(x), B(x)] \tag{5.4}$$

$$(A \cup B)(x) = max[A(x), B(x)] \tag{5.5}$$

pentru toți $x \in X$. Aceste operații se numesc operații fuzzy standard; ele mai sunt cunoscute și sub numele de complement, intersecție, respectiv reuniune. Una dintre proprietățile remarcabile și dezirabile ale acestor operații este faptul că, având o eroare e asociată gradelor de apartenență $A(x)$ și $B(x)$, atunci eroarea maximă asociată cu gradul de apartenență a lui x în $\bar{A}(x)$, $A \cap B$ și $A \cup B$ rămâne e. Majoritatea celorlalte operații fuzzy nu au această proprietate (Klir, p. 51).

Din toată varietatea de mulțimi fuzzy, o importanță deosebită o au mulțimile fuzzy ce sunt definite peste mulțimea $\mathbb{R}$ a numerelor reale. Funcțiile de apartenență ale acestor mulțimi, care sunt de forma

$$A : \mathbb{R} \to [0, 1] \tag{5.6}$$

au, în mod evident, o interpretare cantitativă, și pot fi, în anumite cazuri, privite ca fiind numere fuzzy sau intervale fuzzy. Pentru a putea obține această viziune asupra lor, ele trebuie să modeleze concepția noastră intuitivă de aproximare a numerelor sau intervalelor, i.e. "numere care sunt "aproape" de un număr real dat" sau "numere care sunt "aproape" de un interval de numere reale dat".

Având conceptul de număr fuzzy putem acum introduce noțiunea de variabile fuzzy cantitative. Acestea sunt variabile ale căror stări sunt numere fuzzy. Dacă, în plus, aceste numere fuzzy modelează concepte lingvistice, precum "foarte mic", "mic", "mediu" etc., interpretate sub un anumit context, atunci construcția rezultantă se numește variabilă lingvistică.

Fiecare variabilă lingvistică are stările exprimate similar cu felul în care numerele fuzzy sunt exprimate în func;tie de o variabilă de bază, unde aceste numere fuzzy sunt numere reale încadrate într-un anumit interval. Astfel, termenii lingvistici ai unei variabile lingvistice reprezintă valori aproximative ale unei variabile de baza, și sunt capturate ca atare de către numere fuzzy.

Orice variabilă lingvistică este caracterizată de către un cvintuplu (v, T, X, g, m), unde v este numele variabilei, T este mulțimea termenilor lingvistici ai lui v care se referă la o variabilă de bază ale ca;rei valori acoperă o mulțime universală X, g este o regulă sintactică (o gramatică) pentru a genera termeni lingvistici, iar m este o regulă semantică ce atribuie fiecărui termen $t \in T$ semnificația sa, $m(t)$, care este o mulțime fuzzy peste X (i.e. $m : T \to \mathcal{F}(X)$), unde $\mathcal{F}(X)$ este analogul mulțimii părților din mulțimile clasice pentru mulțimi fuzzy.

Putem, așadar, observa cum modelul COCOMO poate fi reprezentat cu ușurință în raport cu mulțimi fuzzy, putând modela fiecare caracteristică sub formă de variabile lingvistice.

5.2 Sisteme fuzzy

În general, un sistem fuzzy este un sistem ce conține una sau mai multe variabile care primește valori peste stări care sunt mulțimi fuzzy. Pentru fiecare variabilă, mulțimile fuzzy sunt definite peste o mulțime universală relevantă, care este, de obicei, un interval de numere reale. În acest caz particular, dar totuși important, mulțimile fuzzy sunt numere fuzzy, iar variabilele asociate sunt variabile lingvistice.

Reprezentarea stărilor variabilelor este un mod de a cuantiza variabilele. Datorită preciziei finite pe care orice metodă de măsurare o are, cuantizarea potrivită, care poate reflecta limitarea măsurătorilor, este inevitabilă indiferent daca variabila modelează un atribut al lumii reale sau nu. Intervalul numerelor reale ce reprezintă un spectru de valori ale variabilei este partiționat în subintervale corespunzătoare. Valorile distincte din fiecare interval nu pot fi distinse de către instrumentele de măsură, și, prin urmare, sunt considerate echivalente. Subintervalele sunt etichetate cu numere reale corespunzătoare (numere reale relevante cu eroare de o cifră semnificativă, două cifre semnificative etc), iar aceste etichete sunt considerate stările variabilei; stările oricărei variabile cuantizate sunt deci reprezentanți ai unei clasde de echivalență pentru valoarea reală a variabilei. Fiecare stare a unei variabile cuantizate este asociată cu incertitudine

legată de valoarea reală a variabilei. Această incertitudine poate fi măsurată după dimensiunea clasei de echivalență.

5.3 Clusterizare subtractivă

Scopul clusterizării este de a produce grupări ale datelor dintr-o mulțime mare, astfel încât să se producă o reprezentare naturală a comportamentului unui sistem. Unul dintre cei mai studiați algoritmi de clusterizare este fuzzy C-means (FCM) [9]. Algoritmul FCM se bazează pe o optimizare iterativă de minimizare a funcției de cost:

$$J = \sum_{k=1}^{n} \sum_{i=1}^{c} \mu_{ik}^{m} ||x_k - v_i||^2 \tag{5.7}$$

unde n este numărul de puncte de date, c este numărul de clustere, x_k este cel de-al k-lea punct de date, v_i este al i-lea centru de cluster, μ_{ik} este gradul de apartenență al celui de-al k-lea punct de date în cel de-al i-lea cluster, iar m este o constantă mai mare decât 1 (de obicei se alege m = 2). Gradul de apartenență μ_{ik} este definit de:

$$\mu_{ik} = \frac{1}{\sum_{j=1}^{r} (\frac{||x_k - v_i||}{||x_k - vj||})^{2/(m-1)}} \tag{5.8}$$

Pornind cu numărul dorit de clustere c și o estimare inițială pentru fiecare centru de cluster $v_i,\ i = \overline{1, c}$ algoritmul FCM converge către o soluție pentru v_i care reprezintă fie un punct de minim local, fie un punct șa al funcției de cost. Calitatea soluției, la fel ca și în alte probleme de optimizare neliniară, depinde de valorile inițiale, numărul de clustere c și centrele inițiale.

Algoritmul de clusterizare subtractivă [10] nu are nevoie de o estimare a priori a numărului de clustere, fiind un algoritm rapid și robust pentru generarea clusterelor și identificarea modelelor fuzzy. Vom prezenta în continuare construirea acestuia.

Fie o mulțime de n puncte $\{x_1, .., x_n\}$ într-un spațiu M-dimensional. Presupunem că aceste puncte au fost normalizate în fiecare dimensiune astfel încât intervalele coordonatelor din fiecare dimensiune sunt de lungimi egale. Considerăm fiecare punct de date ca fiind un potențial candidat pentru a deveni un centru de cluster, și definim o măsură a potențialului punctului x_i ca:

$$P_i = \sum_{j=1}^{n} e^{-\alpha ||x_i - xj||^2} \tag{5.9}$$

unde

$$\alpha = \frac{4}{r_a^2} \tag{5.10}$$

iar r_a este o constantă pozitivă. Astfel, măsura potențialului pentru un punct este funcția de distanță față de toate celelalte puncte. Un punct cu foarte multe valori în vecinătate va avea un potențial mare. Constanta r_a este raza ce

definește vecinătatea, punctele din afara vecinătății având o influență aproape neglijabilă asupra potențialului.

După ce am calculat potențialul fiecărui punct, selectăm punctul cu cel mai mare potențial ca fiind primul centru de cluster. Fie x_1^* poziția primului cluster și P_1^* valoarea potențialului său. Putem recalcula potențialul fiecărui punct x_i după formula:

$$P_i \leftarrow P_i - P_1^* e^{-\beta ||x_i - x_1^*||^2} \tag{5.11}$$

unde

$$\beta = \frac{4}{r_h^2} \tag{5.12}$$

și r_h este o constantă pozitivă. Astfel, se scade potențialul tuturor punctelor din vecinătatea centrului în funcție de distanța lor față de el, reducând posibilitatea ca acestea să devină la rândul lor centre. r_h definește raza vecinătății care va avea influență asupra reducerii de potențial ale punctelor.

Când potențialul tuturor punctelor a fost actualizat se selectează următorul punct cu cel mai mare potențial ca fiind cel de-al doilea centru de cluster, repetând operația de la (5.11) asupra acestuia. În general, actualizarea celui de al k-lea centru de cluster va fi

$$P_i \leftarrow P_i - P_k^* e^{-\beta ||x_i - x_k^*||^2} \tag{5.13}$$

unde x_k^* este poziția celui de-al k-lea centru de cluster și P_k^* este valoarea potențialului său.

6 Rețele neuronale

6.1 Perceptronul

Perceptronul este un model matematic inspirat din biologia neuronului uman, folosit în învățarea supervizată a clasificatorilor binari. Acesta permite atât învățare on-line, cât și off-line. Formal, definim perceptronul ca fiind o funcție f care pentru un vector de numere reale x oferă un rezultat $f(x)$, $f(x)$ fiind o valoare singulară.

$$f(x) = \begin{cases} 1 & \text{dacă} \quad w \cdot x > 0 \\ 0 & \text{altfel} \end{cases} \tag{6.1}$$

unde w este vectorul de ponderi, $w \cdot x$ este produsul $\sum_{i=1}^{m} w_i x_i$, unde m este numărul de intrări pentru perceptron iar b este biasul. Biasul deplasează linia de separare față de origine și nu depinde de valorile de intrare.

Având vectorul x definit ca $\{x^1, d^1\}, ..., \{x^m, d^m\}$, unde $x^k \in \mathbb{R}^{n+1}$ este al k-lea punct de intrare, iar $d^k \in \{-1, +1\}, k = 1, \bar{m}$ este ținta pentru cea de-a k-a intrare, x se numește mulțimea de antrenare.

Prin urmare, dorim ca perceptronul să obțină ieșirea y^k pentru fiecare x^k astfel încât $y^k = d^k$. O regulă posibilă pentru a ajunge la o soluție w^* este regula învățării perceptronului [11]:

$$\begin{cases} w^1 \quad \text{arbitrar} \\ w^{k+1} = w^k + \rho(d^k - y^k)x^k \quad k = 1, 2, ... \end{cases} \tag{6.2}$$

unde ρ este o constantă poziti;va numită rată de învățare. Această regulă poate fi generalizată să includă o variabilă incrementală ρ^k și o margine pozitivă fixată b. Utilizând această regulă, vectorul de ponderi se actualizează doar dacă $(z^k)^T w^k$ nu depășește pragul b.

$$\begin{cases} w^1 \quad \text{arbitrar} \\ w^{k+1} = w^k + \rho^k z^k \quad \text{dacă } (z^k)^T w^k \leq b \\ w^{k+1} = w^k \quad \text{altfel} \end{cases} \tag{6.3}$$

6.2 Rețele multi-strat

O rețea multi-strat este compusă dintr-o mulțime de perceptroni, dispuși astfel încât, până la ultimul strat (numit stratul de ieșire), rezultatele unui perceptron să devină intrările altui perceptron de pe stratul următor. Această arhitectură permite, printre altele, aplicarea algoritmului de backpropagare a erorilor, ajustând astfel ponderile fiecărui perceptron din rețea.

Algoritmul backprop calculează gradientul funcției de eroare. Funcția de eroare transformă valori formate din una sau mai multe variabile într-un număr real ce reprezintă, la nivel intuitiv, un "cost" asociat acestor valori. În cazul particular al backprop-ului, funcția de eroare calculează diferența dintre rezultatul oferit de către rețea într-un anumit punct și valoarea reală.

Algoritmul de backprop are următoarea formă:
Fie N o rețea neuronală cu e conexiuni, m intrări și n ieșiri.
x un vector din $\mathbb{R}^m$, y un vector din $\mathbb{R}^n$, w un vector din $\mathbb{R}^e$, denumiți intrări, ieșiri, respectiv ponderi.
Rețeaua neuronală corespunde unei funcții $y = f_N(w, x)$ care, date fiind ponderile w, oferă o etichetă y pentru intrările x.
Optimizarea primește perechi de forma (x_i, y_i) și produce ponderi de forma w_i, pornind de la o pondere inițială w_0.
Aceste ponderi sunt calculate iterativ: calculează w_i pornind de la (x_i, y_i, w_{i-1}); algoritmul oferă un vector nou, w_p, și o funcție nouă $x \mapsto f_N(w_p, x)$. Acest pas se repetă pentru toate perechile (x_i, y_i).
Calculul lui w_1 din (x_1, y_1, w_0) are loc prin presupunerea unei variabile w, și aplicarea gradientului descendent pe funcția $w \mapsto E(f_N(w, x_1), y_1)$ pentru a găsi un minim local, pornind de la $w = w_0$; E fiind funcția de eroare. Acest pas îl face pe w_1 ponderea minimizată găsită de către gradientul descendent.

7 Rețele de tip ANFIS

Vom continua lucrarea prin a prezenta modelul de rețele ANFIS [2]. Acesta introduce conceptul de sistem de inferență fuzzy bazat pe o rețea adaptivă, care poate construi o mulțime de reguli fuzzy *if-then* cu funcțiile de apartenență potrivita așa încât să genereze perechi intrare-ieșire pentru datele oferite.

7.1 Reguli fuzzy if-then

Regulile fuzzy if-then (sau expresiile fuzzy condiționale) sunt de forma *IF A THEN B*, unde A și B sunt etichete ale unor mulțimi fuzzy, caracterizate de către funcțiile lor de apartenență. Datorită formei lor compacte, ele sunt folosite pentru a modela moduri imprecise de gândire care joacă un rol esențial în abilitatea umană de a lua decizii într-un mediu incert și imprecis. Jang oferă un exemplu clar și concis pentru a ilustra o regulă fuzzy if-then:

If pressure is high, then volume is small [2]

unde *pressure* și *volume* sunt variabile lingvistice, iar *high* și *small* sunt valori lingvistice caracterizate de către funcțiile de apartenență.

Un alt fel de reguli fuzzy if-then sunt cele de tip Takagi-Sugeno [8] de forma:

$$if\ x_1 = small\ and\ x_2 = big\ then\ y = x_1 + x_2$$

unde, din nou, small și high sunt valori lingvistice caracterizate de către funcțiile lor de apartenență. Diferența apare în partea consecventă, rezultatul fiind de această dată o ecuație non-fuzzy ce depinde de variabilele x_1 și x_2.

Definim în continuare un sistem de inferență fuzzy (cunoscut și ca model fuzzy sau controller fuzzy când este folosit pe post de controller). Un sistem de inferență fuzzy este compus din cinci unități funcționale:

- o bază de reguli ce conține reguli fuzzy if-then;
- o bază de date ce definește funcțiile de apartenență ale mulțimilor fuzzy folosite în regulile fuzzy;
- o unitate de decizie care execută operațiile de inferență peste reguli;
- o interfață de fuzzificare ce transformă intrările crisp în grade de apartenență la valori lingvistice;
- o interfață de defuzzificare ce transformă rezultatele fuzzy ale inferenței înapoi în date crisp

De obicei baza de reguli si baza de date sunt grupate împreună sub denumirea de "bază de cunoaștere".

Pașii inferenței fuzzy sunt următorii:

1. Variabilele de intrare sunt comparate prin funcțiile de apartenență în partea de premisă pentru a li se oferi valorile de apartenență (fuzzificare)

2. Valorile de apartenență sunt combinate pentru a obține ponderea fiecărei reguli (combinarea are loc printr-un operator T-normat[1], de obicei multiplicare)

3. Este generat consecventul fiecărei reguli (fie el fuzzy sau crisp), în funcție de ponderi

4. Consecvenții sunt agregați pentru a produce o ieșire crisp (defuzzificare)

7.2 Rețele adaptive

Rețelele adaptive sunt un superset al mulțimii rețelelor *feedforward* cu capacitate de învățare supervizată. După cum numele sugerează, rețelele adaptive sunt structuri ce conțin noduri și legături direcționale pentru interconectarea nodurilor, rezultând într-o rețea. Cel puțin o parte dintre noduri sunt adaptive, ceea ce înseamnă că ieșirile acestora depind de parametrii pe care îi primesc, iar regulile de învățare dictează cum acești parametri se schimbă pentru a optimiza o funcție de eroare.

Regula de învățare de bază a rețelelor adaptive se bazează pe pe gradientul descendent și pe regula înlănțuirii. Dar, deoarece metoda gradientului descendent este cunoscută ca fiind foarte lentă și riscă să ajungă blocată în minime locale, arhitectura ANFIS propune un algoritm de învățare hibrid, care este mai rapid, și care funcționează atât în mod *batch* (off-line) cât și *pattern* (on-line).

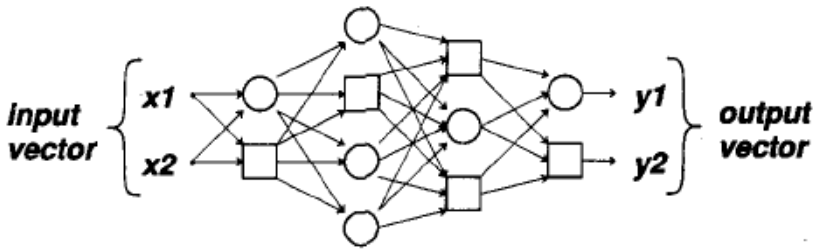

Figura 7.1: Arhitectura unei rețele adaptive cu două intrări și două ieșiri

[1]Un operator T-normat este o funcție $T : [0,1] \times [0,1] \to [0,1]$ care satisface condițiile:
Comutativitate: $T(a,b) = T(b,a)$
Monotonie: $T(a,b) \leq T(c,d)$ dacă $a \leq c$ și $b \leq d$
Asociativitate: $T(a,T(b,c)) = T(T(a,b),c)$
Numărul 1 are comportamentul elementului identitate: $T(a,1) = a$

Așa cum am enunțat înainte, o rețea adaptivă este o rețea multi-strat feedforward în care fiecare nod evaluează o anumită funcție (numită și funcție de nod) pe semnalele de intrare, și, în plus, fiecare nod are și o mulțime de parametri. Formulele pentru funcțiile de nod pot varia de la nod la nod, iar alegerea lor depinde de funcția pe care rețeaua trebuie să o modeleze. O particularitate importantă este că legăturile dintre noduri indică doar direcția pe care o parcurge semnalul; nu există niciun fel de ponderi asociate legăturilor dintre noduri.

Figura 7.1 ilustrează o posibilă configurație a unei rețele cu doi parametri de intrare și doi parametri de ieșire. Pentru a diferenția tipurile de noduri dintr-o rețea adaptivă, sunt folosite noduri în forma rotundă pentru a indica un nod fixat, ce nu primește parametri, și noduri în forma pătrată pentru a indica noduri adaptive, ce primesc parametri. Mulțimea parametrilor unei rețele adaptive este reuniunea tuturor mulțimilor de parametri ale tuturor nodurilor adaptive. Pentru a obține o modelare a unei funcții după intrări și ieșiri, acești parametri sunt actualizați folosind datele de antrenare și o regulă de învățare bazată pe gradient prezentată în continuare.

Fie o rețea adaptivă cu L straturi și cu al k-lea strat având $\#(k)$ noduri. Notăm al i-lea nod de pe stratul k cu (k, i) și funcția de nod cu O_i^k. Din moment ce semnalul de ieșire al unui nod depinde de semnalele de intrare și de parametrii săi, avem

$$O_i^k = O_i^k(O_1^{k-1}, ..., O_{\#(k-1)}^{k-1}, a, b, c, ...) \tag{7.1}$$

unde $a, b, c, ...$ sunt parametrii corespunzători acestui nod. (Notă: am folosit O_i^k atât pentru a ne referi la semnalul de ieșire al nodului cât și la funcția sa.)

Având un set de date antrenare cu P intrări, putem defini măsura de eroare pentru a p-a intrare ($1 \leq p \leq P$) din setul de date de antrenare ca fiind suma pătratelor erorilor:

$$E_p = \sum_{m=1}^{\#(L)} (T_{m,p} - O_{m,p}^L)^2 \tag{7.2}$$

unde $T_{m,p}$ este a m-a componentă a celui de al p-lea vector țintă de rezultate, iar $O_{m,p}^L$ este a m-a componentă a vectorului real de rezultate, produs prin prezentarea celui de-al p-lea vector de intrări.

Prin urmare, măsura totală a erorii este

$$E = \sum_{p=1}^{P} E_p \tag{7.3}$$

Pentru a obține o procedură de învățare care să implementeze gradientul descendent în E peste spațiul parametrilor, avem nevoie în primul rând de rata de eroare $\frac{\partial E_p}{\partial O}$ pentru al p-lea punct din datele de antrenare și pentru fiecare ieșire de nod O. Rata de eroare pentru nodul (L, i) devine (obținută din 7.2)

$$\frac{\partial E_p}{\partial O_{i,p}^k} = -2(T_{i,p} - O_{i,p}^L) \tag{7.4}$$

Pentru nodul intern de la (k, i), rata de eroare poate fi derivată folosind regula înlănțuirii [2]

$$\frac{\partial E_p}{\partial O_{i,p}^k} = \sum_{m=1}^{\#(k+1)} \frac{\partial E_p}{\partial O_{m,p}^{k+1}} \cdot \frac{\partial O_{m,p}^{k+1}}{\partial O_{i,p}^k} \tag{7.5}$$

unde $1 \leq k \leq L-1$. Deci, rata de eroare a unui nod intern poate fi explicitată sub formă de combinație liniară a ratelor de eroare ale nodurilor de pe stratul următor. Prin urmare, pentru toți $1 \leq k \leq L$ și $1 \leq i \leq \#(k)$ putem găsi $\frac{\partial E_p}{\partial O_{i,p}^k}$ conform cu 7.4 și 7.5.

Să presupunem α un parametru al rețelei adaptive. Avem așadar

$$\frac{\partial E_p}{\partial \alpha} = \sum_{O^* \in S} \frac{\partial E_p}{\partial O^*} \cdot \frac{\partial O^*}{\partial \alpha} \tag{7.6}$$

unde S este mulțimea nodurilor ale căror ieșiri depind de α. Derivata măsurii totale a erorii E în raport cu α devine:

$$\frac{\partial E_p}{\partial \alpha} = \sum_{p=1}^{P} \frac{\partial E_p}{\partial \alpha} \tag{7.7}$$

Formula de actualizare pentru parametrul generic α este:

$$\Delta\alpha = -\eta \frac{\partial E}{\partial \alpha} \tag{7.8}$$

unde η este rata de învățare care poate fi exprimată și ca:

$$\eta = -\frac{k}{\sqrt{\sum_{\alpha} \left(\frac{\partial E}{\partial \alpha}\right)^2}} \tag{7.9}$$

unde k este dimensiunea pasului, adică lungimea fiecărei tranziții a gradientului în spațiul parametrilor. De obicei, valorea lui k poate fi schimbată pentru a varia viteza de convergență.

Există două metode de învățare pentru rețelele adaptive. Prin învățarea de tip batch (off-line) formula de actualizare a parametrului α este bazată pe (7.7), iar acțiunea de actualizare are loc doar atunci când intreg setul de date de antrenare a fost parcurs, i.e. după o epocă.

În contrast, dacă dorim ca parametrii să fie actualizați imediat cum o pereche intrare-ieșire i-a fost prezentată rețelei, atunci formula de învățare este bazată pe (7.6), și ne vom referi la ea ca învățare pattern (sau on-line). În continuare, prezentăm o derivare a unei reguli hibride și rapide sub ambele forme: off-line si on-line.

[2] Regula înlănțuirii este definită prin $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$; i.e. având trei variabile x, y, z, cu z dependentă de y, iar y dependentă de x astfel încât y și z sunt variabile dependente, atunci z, prin intermediul lui y depinde de x.

7.3 Regula de învățare hibridă off-line

Regula de învățare hibridă off-line propune o combinație între metoda gradientului și metodă estimării pătratelor minime pentru a identifica parametrii. Deși s-ar putea aplica metoda gradientului, aceasta este lentă și există posibilitatea ca ea să rămanâ blocată în minime locale.

Să presupunem că rețeaua adaptivă pe care vom aplica regula are o singură ieșire:

$$ieșire = F(\overrightarrow{I}, S) \tag{7.10}$$

unde $\overrightarrow{I}$ este mulțimea variabilelor de intrare și S este mulțimea parametrilor. Dacă există o funcție H astfel încât compunerea $H \circ F$ este liniară în unele elemente din S, atunci aceste elemente pot fi identificate prin metoda pătratelor minime. Formal, daca descompunem S în

$$S = S_1 \oplus S_2 \tag{7.11}$$

unde $\oplus$ reprezintă însumarea directă, astfel încât $H \circ F$ este liniară în S_2, atunci aplicând H pe (7.10) avem

$$H(ieșire) = H \circ F(\overrightarrow{I}, S) \tag{7.12}$$

care este liniară în S_2. Date fiind valorile elementelor din S_1 putem introduce datele de antrenare P în (7.12) obținând o ecuație matriceală:

$$AX = B \tag{7.13}$$

unde X este un vector de necunoscute ale cărui elemente sunt parametri în S_2. Fie $|S_2| = M$, atunci dimensiunile lui A, X și B sunt $P \times M, M \times 1$, respectiv $P \times 1$. Din moment de P este, de obicei, mai mare decât M (numărul de puncte de date de antrenare este mai mare decât numărul de parametri liniari), aceasta este o problemă compatibil nedeterminată, și, în general, nu există o soluție exactă pentru (7.13). Prin urmare, o estimare a celor mai mici pătrate ale lui X, X^*, este căutată pentru a minimiza eroarea pătrată $||AX - B||^2$. Aceasta este o problemă standard ce stă la baza regresiei liniare, filtrării adaptive și procesării de semnale. Cea mai cunoscută formulă pentru X^* se folosește de pseudo-inversa lui X:

$$X^* = (A^T A)^{-1} A^T B \tag{7.14}$$

unde A^T este transpusa lui A, iar $(A^T A)^{-1} A^T$ este pseudo-transpusa lui A, dacă A este non-singulară. Deși (7.14) are o notație clară, și ea este ușor de construit intuitiv, calcularea inversei este un proces foarte costisitor din punct de vedere computațional. Mai mult, ecuația nu este bine definită dacă $A^T A$ este singulară. Prin urmare, [2] prezintă o metodă secvențială de a calcula estimarea celor mai mici pătrate ale lui X. Această metodă secvențială este eficientă, și poate fi modificată într-o variantă on-line.

Fie al i-lea vector rând din matricea A definită în (7.13) notat cu a_i^T, și al i-lea element din B notat cu b_i^T. Atunci X poate fi calculat iterativ folosind

formule secvențiale adoptate din literatură:

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1}a_{i+1}(b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} &= S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{1 + a_{i+1}^T S_i a_{i+1}},\ i = 0, 1, ..., P-1 \end{aligned} \tag{7.15}$$

unde S_i este matricea de covarianță și estimarea celor mai mici pătrate este egală cu X_P. Condițiile inițiale pentru a aplica *bootstrapping*[3] sunt $X_0 = 0$ și $S_0 = \gamma I$, unde γ este un număr pozitiv mare, iar I este matricea identitate de dimensiune $M \times M$. Atunci când rețeaua adaptivă are mai multe ie;siri, (*ieșire* din (7.10) este un vector coloană), atunci (7.15) rămâne valabilă, dar b_i^T devine al i-lea rând din B.

Putem acum să combinăm metoda gradientului și metoda estimării pătratelor minime pentru a actualiza parametrii dintr-o rețea adaptivă. Fiecare epocă a acestei proceduri hibride de învățare este compusă dintr-o propagare înainte și o propagare înapoi. În propagarea înainte oferim datele de intrare și semnalele funcționale, iar acestea se propagă înainte pentru a calcula ieșirea fiecărui nod până se obțin matricele A și B din (7.13), iar parametrii din S_2 sunt identificați după formula secvențială a celor mai mici pătrate din (7.15). După ce au fost identificați parametrii din S_2, semnalele funcționale continuă să se propage înainte până când măsura erorii este calculată. În propagarea înapoi ratele de eroare (derivatele măsurii erorii în raport cu fiecare ieșire de nod) se propagă dinspre ieșiri către intrări, iar parametrii din S_1 sunt actualizați după metoda gradientului din (7.9).

Pentru valori fixate date parametrilor din S_1, este garantat că valorile parametrilor din S_2 sunt puncte globale de optim din spațiul parametrilor lui S_2 datorită alegerii măsurii de eroare a pătratelor. Nu numai că această regulă hibridă de învățare descrește dimensiunea spațiului de căutare din metoda gradientului, dar, în general, va avea un timp de convergență substanțial mai mic.

Luând ca exemplu o rețea neuronală cu propagare în spate și funcția de activare sigmoid, dacă ea are p unități de ieșire, atunci *ieșire* din (7.10) este un vector coloană. Fie $H(\cdot)$ inversa funcției sigmoid:

$$H(x) = ln(\frac{x}{1-x}) \tag{7.16}$$

atunci (7.12) devine o funcție liniară astfel încât fiecare element din $H(ieșire)$ este o combinație liniară între parametrii (ponderi și praguri) ce țin de stratul cu numărul 2. În alte cuvinte,
S_1 = ponderile și pragurile stratului ascuns
S_2 = ponderile și pragurile stratului de ieșire
Prin urmare, putem aplica regula de învățare back-propagation pentru a regla parametrii stratului ascuns, și parametrii stratului de ieșire pot fi identificați de

[3]Bootstrapping este o metodă de inferență statistică bazată pe folosirea unei singure eșantionări și înlocuirea aleatorie a unor puncte din datele eșantionate cu alte puncte din același eșantion, în mod repetat.

către metoda celor mai mici pătrate. Trebuie totuși notat că folosind metoda celor mai mici pătrate peste datele transformate de către $H(\cdot)$, parametrii obținuți sunt optimi în funcție de măsura erorii cu eroarea la pătrat, în loc de cea originală. De obicei, acest lucru nu ridică o problemă reală cât timp $H(\cdot)$ este monoton crescătoare.

7.4 Regula de învățare hibridă on-line

Dacă parametrii sunt actualizați după fiecare prezentare a datelorf atunci avem de a face cu paradigma învățarii on-line. Această paradigmă este crucială pentru identificarea parametrilor în sistemele cu caracteristici care sunt în schimbare. Pentru a modifica regula off-line într-o variantă off-line este suficient să observăm că gradientul descendent trebuie să fie bazat pe E_p (vezi (7.6)) în loc de E. Formal, aceasta nu este o procedură în gradient adevărată de minimizare a lui E, dar ea o va aproxima dacă rata de învățare este mică.

Pentru ca formula secvențială a celor mai mici pătrate să poată fi adaptată la caracteristicile variabile în funcție de timp ale datelor de intrare, este necesar ca perechile de date vechi să aibă o pondere din ce în ce mai mică odată ce apar date noi. Încă odată, această problemă este studiată și rezolvată în literatura controlului adaptiv. O metodă este formularea măsurii erorii pătratelor într-o variantă ponderată, unde datele mai vechi primesc factori mai mici de pondere față de cele mai noi. Acest lucru este echivalent cu adăugarea unui factor de uitare λ în formula secvențială originală:

$$\begin{aligned} X_{i+1} &= X_i + S_{i+1}a_{i+1}(b_{i+1}^T - a_{i+1}^T X_i) \\ S_{i+1} &= \frac{1}{\lambda}[S_i - \frac{S_i a_{i+1} a_{i+1}^T S_i}{\lambda + a_{i+1}^T S_i a_{i+1}}],\ i = 0, 1, ..., P-1 \end{aligned} \tag{7.17}$$

unde λ este între 0 și 1. Cu cât λ descrește, cu atât mai mult se observă efectele slăbirii datelor vechi. Dar, uneori, un λ mic poate duce la instabilitate numerică, deci ar trebui evitat.

7.5 Arhitectura ANFIS

Știind acum definiția, arhitectura și regulile de învățare ale unei rețele adaptive, putem deduce faptul că funcțional, nu există aproape nicio constrângere asupra funcțiilor de nod în afară de diferențiabilitatea pe porțiuni[4], iar structural, singura limitare a configurației rețelei este că aceasta trebuie sa fie de tip feedforward. Datorită acestor restricții minimale, observăm că aplicațiile rețelelor adaptive sunt vaste și imediate în diverse domenii. Vom prezenta în continuare o arhitectură de rețele care sunt echivalente funcțional cu un sistem de inferență fuzzy. Vom descrie o metodă de descompunere a mulțimii parametrilor pentru a aplica regula de învățare hibridă.

Să presupunem un sistem de inferență fuzzy cu două variabile de intrare x, y și una de ieșire z. Presupunem că baza de reguli conține două reguli fuzzy if-then de tip Takagi-Sugeno:

1. if x is A_1 and y is B_1 then $f_1 = p_1x + q_1y + r_1$
2. if x is A_2 and y is B_2 then $f_2 = p_2x + q_2y + r_2$

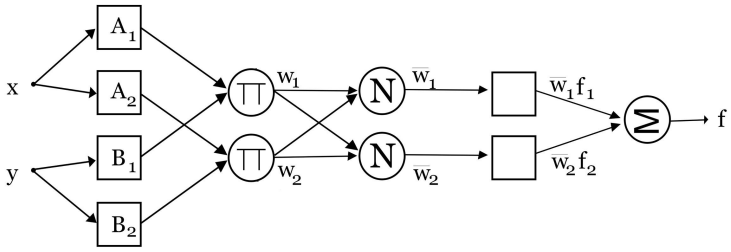

Figura 7.2: Arhitectură ANFIS pentru două reguli fuzzy if-then Takagi-Sugeno.

[4]O funcție f se numește diferențiabilă pe porțiuni dacă:
f este definită pe porțiuni (există mai multe sub-funcții ale lui f, fiecare aplicată pentru un anumit interval din domeniul lui f)
Pentru fiecare porțiune a lui f, subfuncția corespunzătoare acelei porțiuni este diferențiabilă de-a lungul întregului subdomeniu

Toate funcțiile de node aparțin uneia din familiile de funcții definite în continuare, în funcție de strat:

Stratul 1: Fiecare nod i din acest strat este un nod adaptiv cu o funcție de nod

$$O_i^1 = \mu_{A_i}(x) \tag{7.18}$$

unde x este intrarea pentru nodul i, și A_i este eticheta în limbaj natural (i.e. mic, mare, etc.) asociată acestei funcții de nod. Altfel spus, O_i^1 este funcția de apartenență a lui A_i, specificând gradul cu care un x dat satisface cuantificatorul A_i. De obicei, se alege $\mu_{A_i}(x)$ astfel încât să fie gaussiană cu maximul egal cu 1 și minimul egal cu 0, cu formele:

$$\mu_{A_i}(x) = \frac{1}{1 + [(\frac{x-c_i}{a_i})^2]b_i} \tag{7.19}$$

sau

$$\mu_{A_i}(x) = exp[-(\frac{x-c_i}{a_i})^2] \tag{7.20}$$

unde $\{a_i, b_i, c_i\}$ este mulțimea de parametri. În funcție de schimbările parametrilor, funcțiile gaussiene variază corespunzător, arătând astfel diefirte forme de funcții de apartenență peste eticheta în limbaj natural A_i. De fapt, orice funcții continue și diferențiabile pe porțiuni, cum ar fi funcțiile de apartenență trapezoidale sau triunghiulare, sunt candidați buni pentru acest strat. Parametrii acestui strat se mai numesc și parametri de premise.

Stratul 2: Fiecare nod din acest strat este un nod fix (i.e. nu are parametri) etichetat cu Π, care înmulțește semnalele primite și trimite produsul acestora. De exemplu,

$$w_i = \mu_{A_i}(x) \times \mu_{B_i}(y), \ i = 1, 2. \tag{7.21}$$

Fiecare ieșire de nod reprezintă puterea de "aprindere" a regulii. Mai mult, orice alți operatori T-normați care calculează un "ȘI" generalizat pot fi folosite ca funcții de nod pe acest strat.

Stratul 3: Fiecare nod de pe acest strat este un nod fix etichetat cu "N". Cel de-al i-lea nod calculează raportul puterii de "aprindere" a celei de a i-a regulă la suma tuturor puterilor de "aprindere" ale regulilor.

$$\bar{w}_i = \frac{w_i}{w_1 + w_2}, \ i = 1, 2 \tag{7.22}$$

Ieșirile acestui strat se mai numesc și puterile de "aprindere" normalizate.

Stratul 4: Fiecare nod i din acest strat este un nod adaptiv, cu funcția

$$O_i^4 = \bar{w}_i f_i = \bar{w}_i(p_i x + q_i y + r_i) \tag{7.23}$$

unde $\bar{w}_i$ este ieșirea stratului 3, iar $\{p_i, q_i, r_i\}$ este mulțimea parametrilor. Parametrii acestui strat se mai numesc și parametrii consecvenți.

Stratul 5: Singurul nod din acest strat este un nod fix etichetat cu σ care calculează semnalul general de ieșire ca fiind suma tuturor semnalelor care ajung la el

$$O_1^5 = \text{ieșire generală} = \sum_i \bar{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i} \tag{7.24}$$

Având toate straturile definite obținem construcția unei rețele adaptive ce modelează un sistem de inferență fuzzy format din reguli if-then de tipul Takagi-Sugeno.

7.6 Aplicarea algoritmului de învățare hibrid

Continuând cu același exemplu ca în secțiunea 7.5, avem un sistem de inferență fuzzy cu două reguli:

$$\begin{aligned} f_1 &= p_1 x + q_1 y + r_1 \\ f_2 &= p_2 x + q_2 y + r_2 \end{aligned} \tag{7.25}$$

Se poate observa că dacă avem valorile parametrilor de premise, putem exprima ieșirea finală sub formă de combinație liniară a parametrilor consecvenți. Mai precis, putem scrie ieșirea f ca:

$$\begin{aligned} f &= \frac{w_1}{w_1 + w_2} f_1 + \frac{w_2}{w_1 + w_2} f_2 \\ &= \bar{w}_1 f_1 + \bar{w}_2 + f_2 \\ &= (\bar{w}_1 x) p_1 + (\bar{w}_1 y) q_1 + (\bar{w}_1) r_1 + \\ &\quad (\bar{w}_2 x) p_2 + (\bar{w}_2 y) q_2 + (\bar{w}_2) r_2 \end{aligned} \tag{7.26}$$

Observăm că f este liniară în parametrii consecvenți $p_1, q_1, r_1, p_2, q_2, r_2$. Obținem

$$S = S_1 \cup S_2 \tag{7.27}$$

unde $S =$ mulțimea tuturor parametrilor, $S_1 =$ mulțimea parametrilor de premise și $S_2 =$ mulțimea parametrilor consecvenți pentru ecuația (7.11). $H(\cdot)$ și $F(\cdot, \cdot)$ sunt funcțiile identitate, respectiv funcția sistemului de inferență fuzzy. Prin urmare, regula de învățare hibridă poate fi aplicată direct; în direcția înainte a regulii semnalele funcțiilor ajung până în stratul 4, iar parametrii consecvenți sunt identificați de estimarea celor mai mici pătrate. În direcția înapoi, ratele de eroare se propagă înapoi, iar parametrii de premise sunt actualizați după regula gradientului descendent.

8 Măsura performanței modelelor

8.1 Root mean squared error - RMSE

RMSE este des folosită și este o metrică generală de erori pentru predicțiile numerice. Comparată cu alte metrici, cum ar fi cea a mediei absolute a erorilor, RMSE amplifică și scoate în evidență toate erorile mari.

Formula pentru RMSE este:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{8.1}$$

8.2 Mean Magnitude Relative Error - MMRE

Metrica MMRE este una dintre cele mai folosite în interpretarea acurateței unui model estimativ de cost. Pentru a ajunge la calculul MMRE este nevoie ca inițial să fie calculată pentru fiecare punct din date metrica MRE (mean relative error), cu formula

$$MRE_i = \frac{|y_i - \hat{y}_i|}{y_i} \tag{8.2}$$

Având MRE calculat pentru toate punctele, MMRE este definit ca

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i \tag{8.3}$$

8.3 PRED(25)

PRED(25) este o altă metrică utilizată pentru a interpreta acuratețea unui model estimativ de cost. Această măsoară câte din estimările făcute se încadrează într-o rată de +/- 25% față de valorea reală.

$$PRED(25) = \frac{1}{N}\sum_{i=1}^{N} \begin{cases} 1 & \text{dacă } MRE_i \leq x \\ 0 & \text{altfel} \end{cases} \tag{8.4}$$

9 Rezultate experimentale

Având definite rețelele de tip ANFIS și cunoscând acum modelul COCOMO, reamintim scopul experimentului: predicția efortului dezvoltării unui proiect software plecând de la factorii de cost COCOMO și estimarea proiectului în mii de linii de cod sursă livrate.

9.1 Metodologia de lucru

Datele pe care le-am folosit au fost prezentate în capitolul 4. Acestea au fost împărțite, în mod randomizat, în 85% date de antrenare și 15% date de testare. Următorul pas a fost aplicarea normalizării min-max peste acestea, astfel încât valorile extreme să nu aibă o pondere exagerată față de restul datelor.

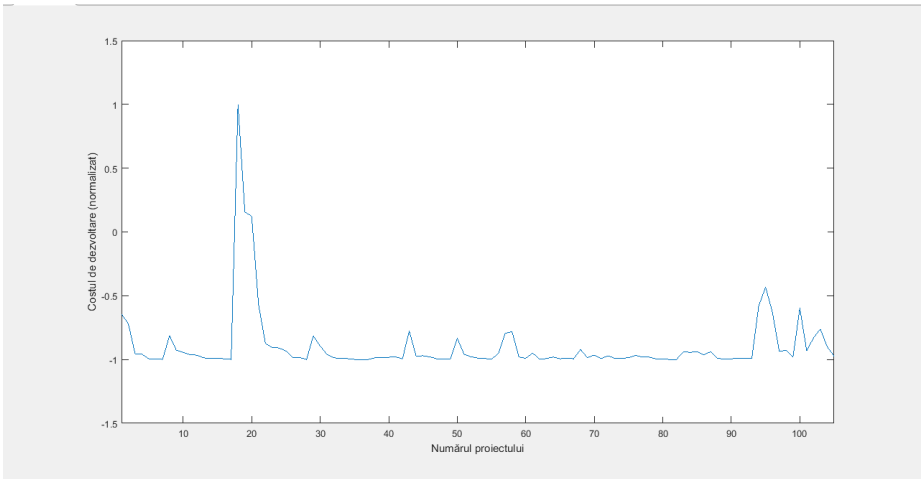

Figura 9.1: Repartiția costurilor din mulțimea de antrenare pentru fiecare proiect

Putem observa ușor din Figura 9.1 că există valori extreme în date, astfel justificând normalizarea datelor.

Odată normalizate datele am creat un sistem de inferență fuzzy pentru datele de intrare, folosind algoritmul de clustering subtractiv, cu valorile 0.8 pentru "Cluster Influence Range", 0.4 pentru "Squash Factor", 0.2 pentru "Accept Ratio" și 0.07 pentru "Reject Ratio". Astfel s-a obținut un sistem de inferență fuzzy cu 64 de clustere, fiecare cu câte 16 reguli de apartenență pe antecedent.

Sistemul de inferență fuzzy a fost antrenat folosind algoritmul ANFIS pentru 100 de epoci, astfel ajungând să obțină o eroare RMSE de învățare de 0.0008. Eroarea MMRE este de 0.000270 peste mulțimea de antrenare. Rezultatele pot fi vizualizate în figura 9.2.

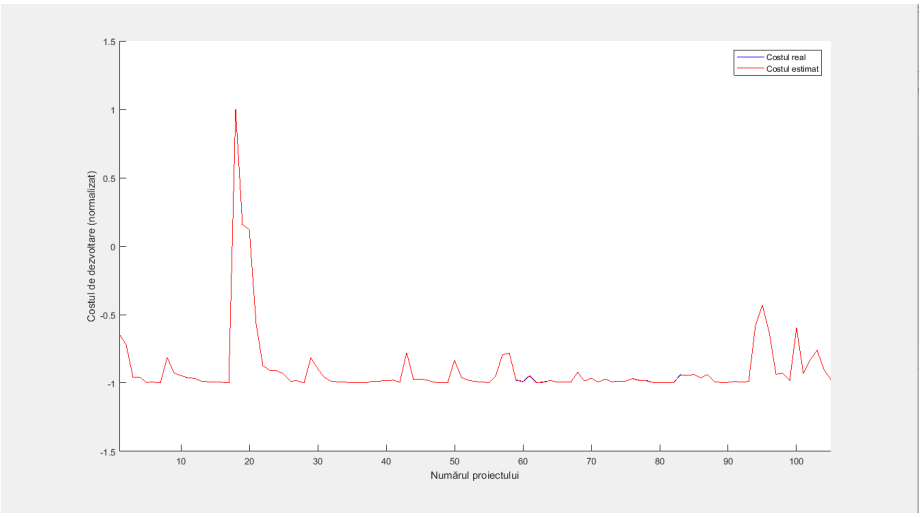

Figura 9.2: Rezultatele antrenării vs valorile reale

Putem observa o învățare aproape completă a datelor de antrenare. Acest lucru nu este neapărat dezirabil, fiind un indicator că modelul suferă de overfitting. Ne vom pronun;ta asupra acestui lucru după observarea comportamentului peste datele de testare.

9.2 Rezultatele evaluării rețelei

Păstrând rețeaua antrenată mai devreme, o vom supune simulării peste cele 18 puncte de test pe care le avem la dispoziție.

Rezultatele evaluării sunt urmâtoarele:

- O eroare RMSE de 0.059
- O eroare MMRE de 0.036
- O estimare PRED(25) de 1

Astfel, este evident că fenomenul de overfitting nu a avut loc, sau dacă acesta a apărut se manifestă insesizabil. Prin urmare, considerăm că modelul propus ne oferă o rată de învățare foarte bună, fiind capabil să ofere predicții cu acuratețe foarte bună pentru proiectele modelate sub COCOMO.

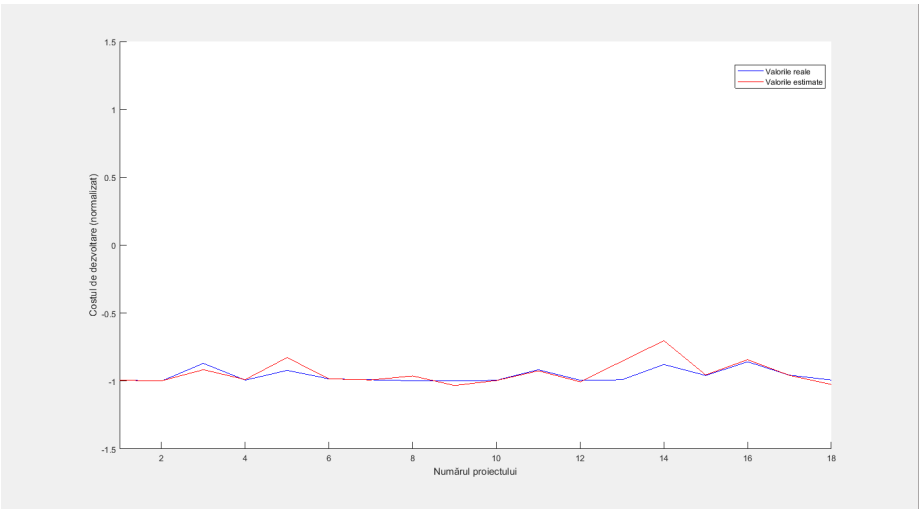

Figura 9.3: Rezultatele evaluării datelor de test vs valorile reale

10 Rezultatele modelului cu rețele neuronale

10.1 Metodologia de lucru

Am folosit aceleași date ca și în capitolul 9, împărțite în 85% date de antrenare, și 15% date de testare. La fel ca înainte, am aplicat aceeași normalizare min-max asupra lor înainte de a începe antrenarea rețelei.

Rețeaua a fost antrenată folosind algoritmul Levenberg-Marquardt de backpropagare, cu un strat ascuns de 10 perceptroni și un număr maxim de 1000 de epoci. Arhitectura rețelei poate fi observată în figura 10.1.

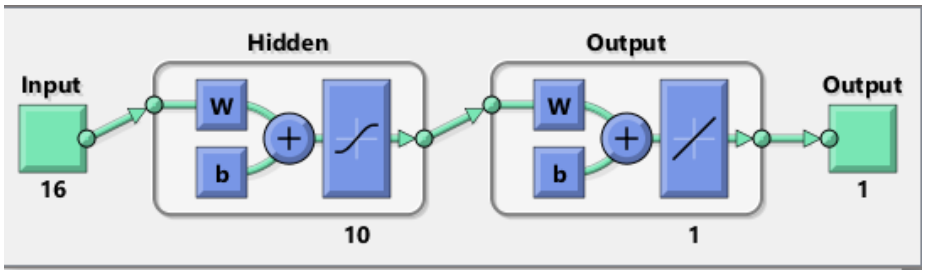

Figura 10.1: Arhitectura rețelei neuronale propuse

10.2 Rezultatele evaluării rețelei

Simularea peste cele 18 puncte de test au oferit următoarele rezultate:

- O eroare RMSE de 0.227
- O eroare MMRE de 0.146
- O estimare PRED(25) de 0.83

Figura 10.2 ne oferă o viziune asupra felului în care rețeaua construiește predicțiile peste datele de test. Putem observa că unele puncte sunt prezise cu o eroare semnificativă, lucru parțial datorat unui set de date de antrenare restrâns.

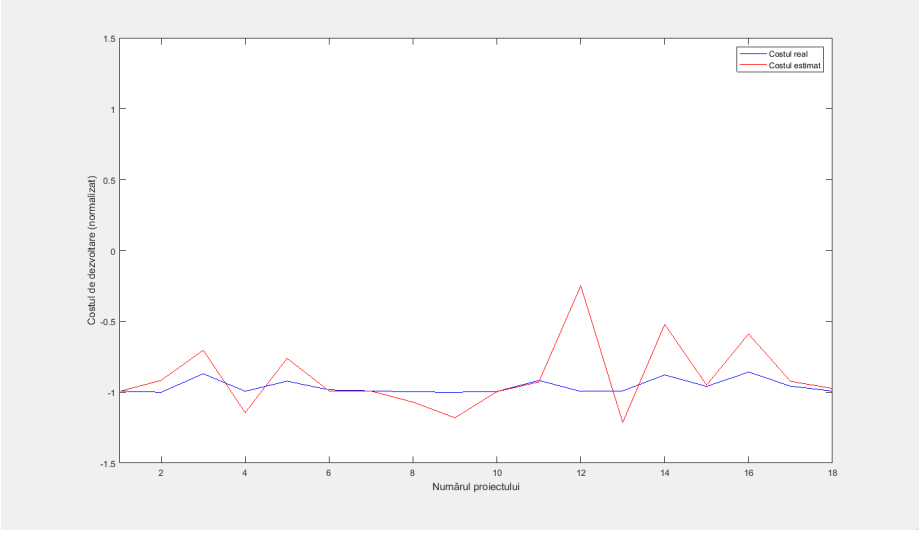

Figura 10.2: Rezultatele evaluării datelor de test folosind rețeaua neuronală propusă

11 Compararea rezultatelor

Observând rezultatele obținute prin evaluarea celor două modele, observăm că modelul bazat pe ANFIS obține rezultate mai bune decât cel bazat pe rețele neuronale prin toate metricile. Acest lucru este parțial datorat faptului că ANFIS este construit astfel încât să obțină rezultate robuste peste date cu o oarecare incertitudine.

Un alt motiv pentru care în acest experiment ANFIS obține erori mai mici este acela că nu au existat suficiente date de antrenare și testare, acesta fiind un punct slab pentru rețelele neuronale, care au un sistem de inferență foarte puternic atunci când li se pot prezenta baze de date cu un număr de intrări cu cel puțin un ordin de magnitudine mai mare decât am avut disponibile pentru acest studiu.

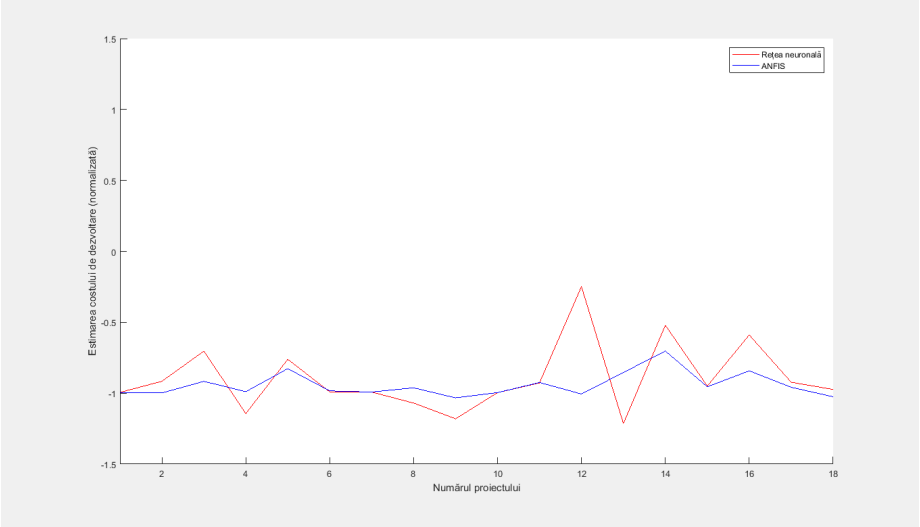

Figura 11.1: ANFIS vs rețeaua neuronală peste datele de test

Din figura 11.1 putem observa faptul că în afară de proiectele 12 și 13, unde rețeaua neuronală oferă o estimare complet diferită față de ANFIS, în general, rețeaua neuronală urmează trendul ANFIS dar mult mai exagerat, întocmai datorită unor ponderi prea mari.

12 Concluzii

Această lucrare a fost o prezentare a unui model de rețele neuro-fuzzy, și anume ANFIS, cât și aplicarea acestuia asupra unui set de date bazat pe COCOMO. Pentru a obține o viziune mai bună asupra performanțelor acestui model, rezultatele acestuia au fost comparate cu cele obținute dintr-o rețea neuronală clasică, fiind un model cunoscut și bine studiat.

Rezultatele inițiale ne arată faptul că ANFIS se descurcă foarte bine, cel puțin pentru acest set de date, obținând o predicție cu eroare sub 25% pentru toate proiectele repartizate în mulțimea de testare. În cealaltă parte, modelul bazat pe rețele neuronale obține aceeași predicție doar în proporție de 83%, fiind aproape la limita de a fi un predictor acceptabil.

Considerăm că unul dintre cele mai puternice puncte ale acestei lucrări a fost învățarea peste toate cele 16 caracteristici ale modelului COCOMO, față de alte articole din literatură care se restrâng la un număr mai mic de caracteristici. În plus, ANFIS poate fi în mod manual ajustat cu cunoștințe expert, fiind până la urmă un sistem de inferență fuzzy. Acest lucru permite și analizarea foarte ușoară în cazul în care estimările au erori foarte mari, fiind posibilă găsirea regulii și/sau a clusterului care generează aceste defecte.

O viitoare direcție de cercetare poate consta în a găsi metode cât mai eficiente de integrare a cunoștințelor expert încă din momentul antrenării rețelelor ANFIS, astfel îmbunătățind eficien;ta învățarii. În plus, comparația ar putea fi reluată și îmbunătățită atât pentru rețelele ANFIS, cât și pentru cele neuronale dacă ar exista o metodă verosimilă de a genera mulțimi noi de proiecte sintetice sub modelul COCOMO.

În concluzie, noi considerăm modelul ANFIS ca având un domeniu de aplicație foarte bun în predicția efortului de dezvoltare ale proiectelor software. În plus, păstrând aceeași arhitectură a rețelei, scopul acesteia poate fi schimbat pentru a prezice viitoarele defecte dintr-un proiect software (desigur, date fiind datele de antrenare corespunzătoare) sau puncte în seriile de timp. Aplicabilitatea ANFIS a fost deja extinsă cu succes și in domeniul controlului automat, și cu siguranță acesta este doar începutul.

Bibliografie

[1] Boehm, Barry W. (1981). *Software Engineering Economics.* Prentice-Hall.

[2] Jang, J.-S.R. (1993) ANFIS: Adaptive-Network-Based Fuzzy Inference System. *IEEE Transactions On Systems, Man, And Cybernetics*, 23(3), pp. 665-685.

[3] Kaushik, A. et al. (2012) COCOMO Estimates Using Neural Networks *I.J. Intelligent Systems and Applications*, 9, pp. 22-8.

[4] Mokri, F. D. et al. (2016) Software Cost Estimation using Adaptive Neuro Fuzzy Inference System *International Journal of Academic Research in Computer Engineering*, 1(1), pp. 34-39.

[5] Promise Software Engineering Repository (2004), *COCOMO Nasa/Software Cost Estimation.* [online] Disponibil la http://promise.site.uottawa.ca/SERepository/datasets/cocomonasa_v1.arff (30 Aug. 2017)

[6] Promise Software Engineering Repository (2004), *COCOMO 81/Software Cost Estimation.* [online] Disponibil la http://promise.site.uottawa.ca/SERepository/datasets/cocomo81.arff (30 Aug. 2017)

[7] Klir, George J. et al. (1995). *Fuzzy Sets and Fuzzy Logic.* Pretince-Hall.

[8] Takagi, T. et al. (1983) Derivation of fuzzy control rules from human operatorș control actions. *Proc. IFAC Symp. Fuzzy Inform., Knowledge Representation and Decision Analysis*, pp. 55-60.

[9] Dunn, J. (1974). A fuzzy relative of the ISODATA process and its use in detecting compact, well separated cluster. *J. Cybernetics*, 3(3), pp. 32-57.

[10] Chiu, S. (1994). Fuzzy Model Identification Based on Cluster Estimation. *Journal of the Intelligent and Fuzzy Systems*, 1(2), pp. 267-278.

[11] Enachescu, D. (2003) *Elements of Statistical Learning. Applications in Data Mining.* Cooperativa Libraria Editrice Universita di Padova.

13 Anexă

Sursele folosite pentru această lucrare, seturile de date și documentația LaTeX pot fi găsite la adresa https://github.com/nfmadar/finalver-unibuc2017 .